

in a step 906. The parent facilitator responds to this service request and at a later time, the client agent receives the results of the request in a step 908, operations of Figure 9 being complete in a done step 910.

FIGURE 10 depicts operations involved in a client agent responding to a service request in accordance with a preferred embodiment of the present invention. Once started in a step 1000, the client agent receives the service request in a step 1002. In a next step 1004, the client agent parses the received request from ICL. The client agent then determines if the service is available in a step 1006. If it is not, the client agent returns a status report to that effect in a step 1008. If the service is available, control is passed to a step 1010 where the client performs the requested service. Note that in completing step 1010 the client may form complex goal expressions, requesting results for these solvables from the facilitator agent. For example, a fax agent might fax a document to a certain person only after requesting and receiving a fax number for that person. Subsequently, the client agent either returns the results of the service and/or a status report in a step 1012. The operations of Figure 10 are complete in a done step 1014.

FIGURE 11 depicts operations involved in a facilitator agent response to a service request in accordance with a preferred embodiment of the present invention. The start of such operations in step 1100 leads to the reception of a goal request in a step 1102 by the facilitator. This request is then parsed and interpreted by the facilitator in a step 1104. The facilitator then proceeds to construct a goal satisfaction plan in a next step 1106. In steps 1108 and 1110, respectively, the facilitator determines the required sub-goals and then selects agents suitable for performing the required sub-goals. The facilitator then transmits the sub-goal requests to the selected agents in a step 1112 and receives the results of these transmitted requests in a step 1114. It should be noted that the actual implementation of steps 1112 and 1114 are dependent upon the specific goal satisfaction plan. For instance, certain sub-goals may be sent to separate agents in parallel, while transmission of other sub-goals may be postponed until receipt of particular answers. Further, certain requests may generate multiple responses that generate additional sub-goals. Once the responses have been received, the facilitator determines whether the original requested goal has been completed in a step 1118. If the original requested goal has not been completed,

0165010 036152260

the facilitator recursively repeats the operations 1106 through 1116. Once the original requested goal is completed, the facilitator returns the results to the requesting agent 1118 and the operations are done at 1120.

5 A further preferred embodiment of the present invention incorporates
transparent delegation, which means that a requesting agent can generate a request, and a facilitator can manage the satisfaction of that request, without the requester needing to have any knowledge of the identities or locations of the satisfying agents. In some cases, such as when the request is a data query, the requesting agent may also be oblivious to the *number* of agents involved in satisfying a request. Transparent
10 delegation is possible because agents' capabilities (solvable) are treated as an abstract description of a service, rather than as an entry point into a library or body of code.

A further preferred embodiment of the present invention incorporates facilitator handling of compound goals, preferably involving three types of processing: delegation, optimization and interpretation.

15 *Delegation* processing preferably supports facilitator determination of which specific agents will execute a compound goal and how such a compound goal's sub-goals will be combined and the sub-goal results routed. *Delegation* involves selective application of global and local constraint and advice parameters onto the specific sub-goals. *Delegation* results in a goal that is unambiguous as to its meaning and as to the
20 agents that will participate in satisfying it.

Optimization processing of the completed goal preferably includes the facilitator using sub-goal parallelization where appropriate. *Optimization* results in a goal whose interpretation will require as few exchanges as possible, between the facilitator and the satisfying agents, and can exploit parallel efforts of the satisfying
25 agents, wherever this does not affect the goal's meaning.

Interpretation processing of the optimized goal. Completing the addressing of a goal involves the selection of one or more agents to handle each of its sub-goals (that is, each sub-goal for which this selection has not been specified by the requester). In doing this, the facilitator uses its knowledge of the capabilities of its
30 client agents (and possibly of other facilitators, in a multi-facilitator system). It may also use strategies or advice specified by the requester, as explained below. The

665070-86752260

A data solvable is conceptually similar to a relation in a relational database. The facts associated with each solvable are maintained by the agent library, which also handles incoming messages containing queries of data solvables. The default behavior of an agent library in managing these facts may preferably be refined, using parameters specified with the solvable's declaration. For example, the parameter *single_value* preferably indicates that the solvable should only contain a single fact at any given point in time. The parameter *unique_values* preferably indicates that no duplicate values should be stored.

Other parameters preferably allow data solvables use of the concepts of ownership and persistence. For implementing shared repositories, it is often preferable to maintain a record of which agent created each fact of a data solvable with the creating agent being preferably considered the fact's owner. In many applications, it is preferable to remove an agent's facts when that agent goes offline (for instance, when the agent is no longer participating in the agent community, whether by deliberate termination or by malfunction). When a data solvable is declared to be non-persistent, its facts are automatically maintained in this way, whereas a persistent data solvable preferably retains its facts until they are explicitly removed.

A further preferred embodiment of present invention supports an agent library through procedures by which agents can update (add, remove, and replace) facts belonging to data solvables, either locally or on other agents, given that they have preferably the required permissions. These procedures may preferably be refined using many of the same parameters that apply to service requests. For example, the *address* parameter preferably specifies one or more particular agents to which the update request applies. In its absence, just as with service requests, the update request preferably goes to *all* agents providing the relevant data solvable. This default behavior can be used to maintain coordinated "mirror" copies of a data set within multiple agents, and can be useful in support of distributed, collaborative activities.

Similarly, the *feedback* parameters, described in connection with *oaa_Solve*, are preferably available for use with data maintenance requests.

A further preferred embodiment of present invention supports ability to provide data solvables not just to client agents, but also to facilitator agents. Data solvables can preferably created, maintained and used by a facilitator. The facilitator preferably can, at the request of a client of the facilitator, create, maintain and share the use of data solvables with all the facilitator's clients. This can be useful with relatively stable collections of agents, where the facilitator's workload is predictable.

Using a Blackboard Style of Communication

In a further preferred embodiment of present invention, when a data solvable is publicly readable and writable, it acts essentially as a global data repository and can be used cooperatively by a group of agents. In combination with the use of triggers, this allows the agents to organize their efforts around a "blackboard" style of communication.

As an example, the "DCG-NL" agent (one of several existing natural language processing agents), provides natural language processing services for a variety of its peer agents, expects those other agents to record, on the facilitator, the vocabulary to which they are prepared to respond, with an indication of each word's part of speech, and of the logical form (*ICL* sub-goal) that should result from the use of that word. In a further preferred embodiment of present invention, the NL agent, preferably when it comes online, preferably installs a data solvable for each basic part of speech on its facilitator. For instance, one such solvable would be:

```
solvable(noun(Meaning, Syntax), [], [])
```

Note that the empty lists for the solvable's permissions and parameters are acceptable here, since the default permissions and parameters provide appropriate functionality.

A further preferred embodiment of present invention incorporating an Office Assistant system as discussed herein or similar to the discussion here supports several agents making use of these or similar services. For instance, the database agent uses the following call, to library procedure *oaa_AddData*, to post the noun `boss', and to indicate that the "meaning" of boss is the concept `manager':

```
oaa_AddData(noun(manager, atom(boss)), [address(parent)])
```

Autonomous Monitoring with Triggers

A further preferred embodiment of present invention includes support for triggers, providing a general mechanism for requesting some action be taken when a set of conditions is met. Each agent can preferably install triggers either locally, for
5 itself, or remotely, on its facilitator or peer agents. There are preferably at least four types of triggers: communication, data, task, and time. In addition to a type, each trigger preferably specifies at least a condition and an action, both preferably expressed in *ICL*. The condition indicates under what circumstances the trigger should fire, and the action indicates what should happen when it fires. In addition, each
10 trigger can be set to fire either an unlimited number of times, or a specified number of times, which can be any positive integer.

Triggers can be used in a variety of ways within preferred embodiments of the present invention. For example, triggers can be used for monitoring external sensors in the execution environment, tracking the progress of complex tasks, or coordinating
15 communications between agents that are essential for the synchronization of related tasks. The installation of a trigger within an agent can be thought of as a representation of that agent's *commitment* to carry out the specified action, whenever the specified condition holds true.

Communication triggers preferably allow any incoming or outgoing event
20 (message) to be monitored. For instance, a simple communication trigger may say something like: "Whenever a solution to a goal is returned from the facilitator, send the result to the presentation manager to be displayed to the user."

Data triggers preferably monitor the state of a data repository (which can be maintained on a facilitator or a client agent). Data triggers' conditions may be tested
25 upon the addition, removal, or replacement of a fact belonging to a data solvable. An example data trigger is: "When 15 users are simultaneously logged on to a machine, send an alert message to the system administrator."

Task triggers preferably contain conditions that are tested after the processing of each incoming event and whenever a timeout occurs in the event polling. These
30 conditions may specify any goal executable by the local *ICL* interpreter, and most often are used to test when some solvable becomes satisfiable. Task triggers are

useful in checking for task-specific internal conditions. Although in many cases such conditions are captured by solvables, in other cases they may not be. For example, a mail agent might watch for new incoming mail, or an airline database agent may monitor which flights will arrive later than scheduled. An example task trigger is:
5 "When mail arrives for me about security, notify me immediately."

Time triggers preferably monitor time conditions. For instance, an alarm trigger can be set to fire at a single fixed point in time (e.g., "On December 23rd at 3pm"), or on a recurring basis (e.g., "Every three minutes from now until noon").

Triggers are preferably implemented as data solvables, declared implicitly for every agent. When requesting that a trigger be installed, an agent may use many of the
10 same parameters that apply to service and data maintenance requests.

A further preferred embodiment of present invention incorporates semantic support, in contrast with most programming methodologies, of the agent on which the trigger is installed only having to know how to evaluate the conditional part of the
15 trigger, not the consequence. When the trigger fires, the action is delegated to the facilitator for execution. Whereas many commercial mail programs allow rules of the form "When mail arrives about XXX, [forward it, delete it, archive it]", the possible actions are hard-coded and the user must select from a fixed set.

A further preferred embodiment of present invention, the consequence of a
20 trigger may be any compound goal executable by the dynamic community of agents. Since new agents preferably define both functionality and vocabulary, when an unanticipated agent (for example, a fax agent) joins the community, no modifications to existing code is required for a user to make use of it - "When mail arrives, fax it to Bill Smith."

25

The Agent Library

In a preferred embodiment of present invention, the agent library provides the infrastructure for constructing an agent-based system. The essential elements of protocol (involving the details of the messages that encapsulate a service request and
30 its response) are preferably made transparent to simplify the programming applications. This enables the developer to focus functionality, rather than message

construction details and communication details. For example, to request a service of another agent, an agent preferably calls the library procedure *oaa_Solve*. This call results in a message to a facilitator, which will exchange messages with one or more service providers, and then send a message containing the desired results to the requesting agent. These results are returned via one of the arguments of *oaa_Solve*. None of the messages involved in this scenario is explicitly constructed by the agent developer. Note that this describes the *synchronous* use of *oaa_Solve*.

In another preferred embodiment of present invention, an agent library provides both *intraagent* and *interagent* infrastructure; that is, mechanisms supporting the internal structure of individual agents, on the one hand, and mechanisms of cooperative interoperation between agents, on the other. Note that most of the infrastructure cuts across this boundary with many of the same mechanisms supporting both agent internals and agent interactions in an integrated fashion. For example, services provided by an agent preferably can be accessed by that agent through the same procedure (*oaa_Solve*) that it would employ to request a service of another agent (the only difference being in the *address* parameter accompanying the request). This helps the developer to reuse code and avoid redundant entry points into the same functionality.

Both of the preferred characteristics described above (transparent construction of messages and integration of *intraagent* with *interagent* mechanisms) apply to most other library functionality as well, including but not limited to data management and temporal control mechanisms.

Source Code Appendix

Source code for version 2.0 of the *OAA* software product is included as an appendix hereto, and is incorporated herein by reference. The code includes an agent library, which provides infrastructure for constructing an agent-based system. The library's several families of procedures provide the functionalities discussed above, as well as others that have not been discussed here but that will be sufficiently clear to the interested practitioner. For example, declarations of an agent's solvables, and their registration with a facilitator, are managed using procedures such as *oaa_Declare*, *oaa_Undeclare*, and *oaa_Redeclare*. Updates to data solvables can be accomplished with a family of procedures including *oaa_AddData*, *oaa_RemoveData*, and

oaa_ReplaceData. Similarly, triggers are maintained using procedures such as *oaa_AddTrigger*, *oaa_RemoveTrigger*, and *oaa_ReplaceTrigger*. The provided source code also includes source code for an OAA Facilitator Agent.

5 The source code appendix is offered solely as a means of further helping practitioners to construct a preferred embodiment of the invention. By no means is the source code intended to limit the scope of the present invention.

Illustrative Applications

To further illustrate the technology of the preferred embodiment, we will next present and discuss two sample applications of the present inventions.

10 **Unified Messaging**

A further preferred embodiment of present invention incorporates a Unified Messaging application extending the Automated Office application presented previously herein with an emphasis on ubiquitous access and dynamic presentation of the information and services supported by the agent community. The agents used in
15 this application are depicted in Figure 12.

A hypothetical example of realistic dialog using a preferred embodiment of the present invention can provide insight into how systems may preferably be built using the present invention. In this scenario, the user, with only a telephone as an interface, is planning a trip to Boston where he will soon give a presentation.
20 Capitalized sentences are phrases spoken by the user into the telephone and processed by a phone agent 452.

Responses, unless otherwise indicated, are spoken by the system using text-to-speech generation agent 454.

25 1.1 Welcome to SRI International. Please enter your user ID and password.

<User enters touchtone ID and password>

Good to see you again Adam Cheyer. I am listening to you.

30 Every user interface agent 408, including the telephone agent 452, should know the identify of its user. This information is used in resolving anaphoric

references such as "me" and "I", and allows multiple user interfaces operated by the same user to work together.

1.2 WHAT IS TODAY'S SCHEDULE?

Here is today's schedule for Adam Cheyer:

5 At 10am for 1 hour, meeting with Dave.

 At 3pm for 1 hour, presentation about software agents.

 End of schedule.

 If the user is operating both a graphical user interface and a telephone, as
10 described in conjunction with the Automated Office application, the result of this
spoken request is to display a calendar window containing the current schedule. In
this case, with no graphical display available, the GEN_NL agent 1202 is tasked to
produce a spoken response that can be played over the phone. GEN_NL shares the
same dynamic vocabulary and phrasal rules as the natural language parser DCG_NL
15 426, and contains strategies for producing responses to queries using either simple or
list-based multimedia utterances.

1.3 FIND FRIDAY'S WEATHER IN BOSTON.

The weather in Boston for Friday is as follows:

20 Sunny in the morning. Partly cloudy in the
 afternoon with a 20
 percent chance of thunderstorms late. Highs in the
 mid 70s.

 In addition to data accessible from legacy applications, content may be
retrieved by web-reading agents which provide wrappers around useful websites.

25 1.4 FIND ALL NEW MAIL MESSAGES.

 There are 2 messages available.

 Message 1, from Mark Tierny, entitled "OAA meeting."

1.5 NEXT MESSAGE

30 Message 2, from Jennifer Schwefler, entitled
 "Presentation Summary."

1.6 PLAY IT.

 This message is a multipart MIME-encoded message.
 There are two parts.

35 Part 1. (Voicemail message, not text-to speech):
 Thanks for taking part as a speaker in our
 conference.

 The schedule will be posted soon on our homepage.

1.7 NEXT PART

40 Part 2. (read using text-to-speech):
 The presentation home page is <http://www...>

1.8 PRINT MESSAGE

 Command executed.

Mail messages are no longer just simple text documents, but often consist of multiple subparts containing audio files, pictures, webpages, attachments and so forth. When a user asks to play a complex email message over the telephone, many different agents may be implicated in the translation process, which would be quite different given the request "print it." The challenge is to develop a system which will enable agents to cooperate in an extensible, flexible manner that alleviates explicit coding of agent interactions for every possible input/output combination.

In a preferred embodiment of the present invention, each agent concentrates only on what it can do and on what it knows, and leaves other work to be delegated to the agent community. For instance, a printer agent 1204, defining the solvable print(Object,Parameters), can be defined by the following pseudo-code, which basically says, "If someone can get me a document, in either POSTSCRIPT or text form, I can print it."

```
15 print(Object, Parameters) {
    ' If Object is reference to "it", find an appropriate
    document
    if (Object = "ref(it)")
        oaa_Solve(resolve_reference(the, document, Params,
20 Object), []);
    ' Given a reference to some document, ask for the
    document in POSTSCRIPT
    if (Object = "id(Pointer)")
        oaa_Solve(resolve_id_as(id(Pointer), postscript,
25 [], Object), []);
    ' If Object is of type text or POSTSCRIPT, we can
    print it.
    if ((Object is of type Text) or (Object is of type
    Postscript))
30     do_print(Object);
}
```

In the above example, since an email message is the salient document, the mail agent 442 will receive a request to produce the message as POSTSCRIPT. Whereas the mail agent 442 may know how to save a text message as POSTSCRIPT, it will not know what to do with a webpage or voicemail message. For these parts of the message, it will simply send oaa_Solve requests to see if another agent knows how to accomplish the task.

Until now, the user has been using only a telephone as user interface. Now, he moves to his desktop, starts a web browser 436, and accesses the URL referenced by the mail message.

1.9 RECORD MESSAGE

5 Recording voice message. Start speaking now.

1.10 THIS IS THE UPDATED WEB PAGE CONTAINING THE PRESENTATION SCHEDULE.

Message one recorded.

1.11 IF THIS WEB PAGE CHANGES, GET IT TO ME WITH NOTE
10 ONE.

Trigger added as requested.

In this example, a local agent 436 which interfaces with the web browser can return the current page as a solution to the request "oaa_Solve(resolve_reference(this, web_page, [], Ref),[])", sent by the NL agent 426. A trigger is installed on a web
15 agent 436 to monitor changes to the page, and when the page is updated, the notify agent 446 can find the user and transmit the webpage and voicemail message using the most appropriate media transfer mechanism.

This example based on the Unified Messaging application is intended to show how concepts in accordance with the present invention can be used to produce a
20 simple yet extensible solution to a multi-agent problem that would be difficult to implement using a more rigid framework. The application supports adaptable presentation for queries across dynamically changing, complex information; shared context and reference resolution among applications; and flexible translation of multimedia data. In the next section, we will present an application which highlights
25 the use of parallel competition and cooperation among agents during multi-modal fusion.

Multimodal Map

A further preferred embodiment of present invention incorporates the Multimodal Map application. This application demonstrates natural ways of
30 communicating with a community of agents, providing an interactive interface on which the user may draw, write or speak. In a travel-planning domain illustrated by Figure 13, available information includes hotel, restaurant, and tourist-site data retrieved by distributed software agents from commercial Internet sites. Some preferred types of user interactions and multimodal issues handled by the application

are illustrated by a brief scenario featuring working examples taken from the current system.

Sara is planning a business trip to San Francisco, but would like to schedule some activities for the weekend while she is there. She turns on her laptop PC,

5 executes a map application, and selects San Francisco.

2.1 [Speaking] Where is downtown?
Map scrolls to appropriate area.

2.2 [Speaking and drawing region] Show me all hotels near here.

10 Icons representing hotels appear.

2.3 [Writes on a hotel] Info?
A textual description (price, attributes, etc.) appears.

2.4 [Speaking] I only want hotels with a pool.
15 Some hotels disappear.

2.5 [Draws a crossout on a hotel that is too close to a highway]

Hotel disappears

2.6 [Speaking and circling] Show me a photo of this
20 hotel.

Photo appears.

2.7 [Points to another hotel]
Photo appears.

2.8 [Speaking] Price of the other hotel?
25 Price appears for previous hotel.

2.9 [Speaking and drawing an arrow] Scroll down.
Display adjusted.

2.10 [Speaking and drawing an arrow toward a hotel]
30 What is the distance from this hotel to Fisherman's Wharf?

Distance displayed.

2.11 [Pointing to another place and speaking] And the distance to here?

Distance displayed.

35 Sara decides she could use some human advice. She picks up the phone, calls Bob, her travel agent, and writes Start collaboration to synchronize his display with hers. At this point, both are presented with identical maps, and the input and actions of one will be remotely seen by the other.

40 3.1 [Sara speaks and circles two hotels]
Bob, I'm trying to choose between these two hotels. Any opinions?

3.2 [Bob draws an arrow, speaks, and points]
Well, this area is really nice to visit. You can

45 walk there from

this hotel.

Map scrolls to indicated area. Hotel selected.

3.3 [Sara speaks] Do you think I should visit Alcatraz?

3.4 [Bob speaks] Map, show video of Alcatraz.

5 Video appears.

3.5 [Bob speaks] Yes, Alcatraz is a lot of fun.

A further preferred embodiment of present invention generates the most appropriate interpretation for the incoming streams of multimodal input. Besides providing a user interface to a dynamic set of distributed agents, the application is preferably built using an agent framework. The present invention also contemplates aiding the coordinate competition and cooperation among information sources, which in turn works in parallel to resolve the ambiguities arising at every level of the interpretation process: *low-level processing of the data stream, anaphora resolution, cross-modality influences and addressee.*

15 *Low-level processing of the data stream:* Pen input may be preferably interpreted as a gesture (e.g., 2.5: cross-out) by one algorithm, or as handwriting by a separate recognition process (e.g., 2.3: "info?"). Multiple hypotheses may preferably be returned by a modality recognition component.

Anaphora resolution: When resolving anaphoric references, separate information sources may contribute to resolving the reference: context by object type, deictic, visual context, database queries, discourse analysis. An example of information provided through context by object type is found in interpreting an utterance such as "show photo of the hotel", where the natural language component can return a list of the last hotels talked about. Deictic information in combination with a spoken utterance like "show photo of this hotel" may preferably include pointing, circling, or arrow gestures which might indicate the desired object (e.g., 2.7). Deictic references may preferably occur before, during, or after an accompanying verbal command. Information provided in a visual context, given for the request "display photo of the hotel" may preferably include the user interface agent might determine that only one hotel is currently visible on the map, and therefore this might be the desired reference object. Database queries preferably involving information from a database agent combined with results from other resolution strategies. Examples are "show me a photo of the hotel in Menlo Park" and

2.2. Discourse analysis preferably provides a source of information for phrases such as "No, the other one" (or 2.8).

The above list of preferred anaphora resolution mechanisms is not exhaustive. Examples of other preferred resolution methods include but are not limited to spatial reasoning ("the hotel between Fisherman's Wharf and Lombard Street") and user preferences ("near my favorite restaurant").

Cross-modality influences: When multiple modalities are used together, one modality may preferably reinforce or remove or diminish ambiguity from the interpretation of another. For instance, the interpretation of an arrow gesture may vary when accompanied by different verbal commands (e.g., "scroll left" vs. "show info about this hotel"). In the latter example, the system must take into account how accurately and unambiguously an arrow selects a single hotel.

Addressee: With the addition of collaboration technology, humans and automated agents all share the same workspace. A pen doodle or a spoken utterance may be meant for either another human, the system (3.1), or both (3.2).

The implementation of the Multimodal Map application illustrates and exploits several preferred features of the present invention: reference resolution and task delegation by parallel parameters of oaa_Solve, basic multi-user collaboration handled through built-in data management services, additional functionality readily achieved by adding new agents to the community, domain-specific code cleanly separated from other agents.

A further preferred embodiment of present invention provides reference resolution and task delegation handled in a distributed fashion by the parallel parameters of oaa_Solve, with meta-agents encoding rules to help the facilitator make context- or user-specific decisions about priorities among knowledge sources.

A further preferred embodiment of present invention provides basic multi-user collaboration handled through at least one built-in data management service. The map user interface preferably publishes data solvables for elements such as icons, screen position, and viewers, and preferably defines these elements to have the attribute "shareable". For every update to this public data, the changes are preferably

automatically replicated to all members of the collaborative session, with associated callbacks producing the visible effect of the data change (e.g., adding or removing an icon).

Functionality for recording and playback of a session is preferably implemented by adding agents as members of the collaborative community. These agents either record the data changes to disk, or read a log file and replicate the changes in the shared environment.

The domain-specific code for interpreting travel planning dialog is preferably separated from the speech, natural language, pen recognition, database and map user interface agents. These components were preferably reused without modification to add multimodal map capabilities to other applications for activities such as crisis management, multi-robot control, and the MVIEWWS tools for the video analyst.

Improved Scalability and Fault Tolerance

Implementations of a preferred embodiment of present invention which rely upon simple, single facilitator architectures may face certain limitations with respect to scalability, because the single facilitator may become a communications bottleneck and may also represent a single, critical point for system failure.

Multiple facilitator systems as disclosed in the preferred embodiments to this point can be used to construct peer-to-peer agent networks as illustrated in Figure 14. While such embodiments are scalable, they do possess the potential for communication bottlenecks as discussed in the previous paragraph and they further possess the potential for reliability problems as central, critical points of vulnerability to systems failure.

A further embodiment of present invention supports a facilitator implemented as an agent like any other, whereby multiple facilitator network topologies can be readily constructed. One example configuration (but not the only possibility) is a hierarchical topology as depicted in Figure 15, where a top level Facilitator manages collections of both client agents 1508 and other Facilitators, 1504 and 1506. Facilitator agents could be installed for individual users, for a group of users, or as appropriate for the task.

Note further, that network work topologies of facilitators can be seen as graphs where each node corresponds to an instance of a facilitator and each edge connecting two or more nodes corresponds to a transmission path across one or more physical transport mechanisms. Some nodes may represent facilitators and some nodes may represent clients. Each node can be further annotated with attributes corresponding to include triggers, data, capabilities but not limited to these attributes.

A further embodiment of present invention provides enhanced scalability and robustness by separating the planning and execution components of the facilitator. In contrast with the centralized facilitation schemes described above, the facilitator system 1600 of Figure 16 separates the registry/planning component from the execution component. As a result, no single facilitator agent must carry all communications nor does the failure of a single facilitator agent shut down the entire system.

Turning directly to Figure 16, the facilitator system 1600 includes a registry/planner 1602 and a plurality of client agents 1612-1616. The registry/planner 1604 is typically replicated in one or more locations accessible by the client agents. Thus if the registry/planner 1604 becomes unavailable, the client agents can access the replicated registry/planner(s).

This system operates, for example, as follows. An agent transmits a goal 1610 to the registry planner 1602. The registry/planner 1604 translates the goal into an unambiguous execution plan detailing how to accomplish any sub-goals developed from the compound goal, as well as specifying the agents selected for performing the sub-goals. This execution plan is provided to the requesting agent which in turn initiates peer-to-peer interactions 1618 in order to implement the detailed execution plan, routing and combining information as specified within the execution plan. Communication is distributed thus decreasing sensitivity of the system to bandwidth limitations of a single facilitator agent. Execution state is likewise distributed thus enabling system operation even when a facilitator agent fails.

Further embodiments of present invention incorporate into the facilitator functionality such as load-balancing, resource management, and dynamic configuration of agent locations and numbers, using (for example) any of the topologies discussed. Other embodiments incorporate into a facilitator the ability to aid agents in establishing peer-to-peer communications. That is, for tasks requiring a

0925191052260

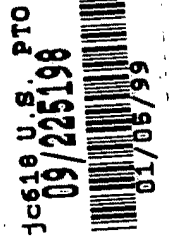
sequence of exchanges between two agents, the facilitator assists the agents in finding one another and establishing communication, stepping out of the way while the agents communicate peer-to-peer over a direct, perhaps dedicated channel.

Further preferred embodiments of the present invention incorporate
5 mechanisms for basic transaction management, such as periodically saving the state of agents (both facilitator and client) and rolling back to the latest saved state in the event of the failure of an agent.

665070-66752260

APPENDIX A.I

Source code file named compound.pl.



```

*****
%   File       : compound.pl
%   Primary Authors  : David Martin, Adam Cheyer
%   Purpose    : Provides handling of compound goals by the facilitator.
%
% -----
%   Unpublished-rights reserved under the copyright laws of the United States.
%
% -----
%   Unpublished Copyright (c) 1998, SRI International.
%   "Open Agent Architecture" and "OAA" are Trademarks of SRI International.
%   -----

```

```

*****

```

```

% This is just here so this file can be compiled separately (but its
% official declaration is in oaa.pl):
:- op(599,yfx,::).

```

```

:- dynamic
   binding_num/1,
   ks_num/1,
   multiple_continuation/7

```

```

% This file is loaded by facilitator code, and thus no
% module imports are needed here.

```

```

*****
% OVERVIEW
*****

```

```

/*\

```

These facilitator routines support the use of compound "ICL goals". An ICLGoal is of the form Sources:Goal::Params, where both Sources and Params are optional. Each subgoal of ICLGoal is also of that form.

When an agent calls solve/2, it may specify an ICL goal which is "incomplete"; that is, ambiguous as to which agents are to solve the various subgoals. The facilitator then completes the ICL goal, if necessary, and executes it. Execution involves having all the subgoals solved by the appropriate agents, assembling the solutions, and returning them to the requesting agent.

If a agent wants to construct a complete ICL goal, and is willing to guarantee that it's complete and that all solvers mentioned in it are currently valid, then that agent (usually a "meta-agent") may call execute_goal directly. @@ We haven't yet provided library calls for this.

IMPORTANT NOTE: : has higher precedence than ::. This means that a:b::c will unify with X:Y and X:Y::Z, but NOT with Y::Z.

Wherever a Sources field appears, it may be any of the following:

```

    built_in
    facilitator

```



```
parent
KS
[KS1, KS2, ...]
```

'built_in' isn't normally specified by a requesting agent - although there's no harm in doing so - but is used internally by the facilitator. KS, KS1, KS2, etc. may be either the name or address of an agent (client or facilitator). 'facilitator' or 'parent' may also appear in a list of KS's. If Sources is an empty list or a var, it is handled just as if there were no Sources field, in which case the facilitator determines what sources are relevant.

Note that when an ICL goal includes a Sources field, there should not be Sources fields for any of its subgoals. If there are, they will be ignored. (@@Need to make sure this works ok.) However, Params fields may be usefully nested within goals that have Params fields. Certain nested parameters, such as solution_limit/1, can be used by the solving agent.

If an ICL goal has parameters, some of them are "inherited" by subgoals. If there's a conflicting parameter on a subgoal, however, it overrides an inherited parameter.

PARAMETERS

address(+A) [embedded or global] - Used precisely as if A: prefixes the relevant goal.

get_address(-S) [embedded] - bind S to indicate who provided the solution. Solver identities will be given as numeric ids. Currently only works when attached to non-compound (sub)goals.

get_address(-S) [global] - bind S to indicate all sources that were queried in finding solutions (even if they returned none).

*/

```
*****
% GOAL COMPLETION
*****
```

/*\

complete_goal(RequestingKS, Goal, GlobalParams, CompletedGoal).

complete_goal takes in an ICL goal and produces a "complete ICL goal" (sometimes known as a "plan", but I think we'll reserve that term for future developments). The goal and the complete goal have precisely the same variables - but are not necessarily unifiable.

*/

```
complete_goal(RequestingKS, Goal, GlobalParams, CompletedGoal) :-
  complete_addressing(RequestingKS, Goal, GlobalParams, AddressedGoal),
  complete_concurrency(AddressedGoal, CompletedGoal).
```

```
/*\
```

```
complete_addressing(+RequestingKS, +ICLGoal, +GlobalParams, -AddressedGoal).
```

AddressedGoal has more-or-less the same form as ICLGoal, but possibly with some regrouping of subgoals, and the addition of Sources fields to ICLGoal or its subgoals. The idea is that AddressedGoal contains complete information as to where its various subgoals are to be sent, so that no further analysis is needed. Any regrouping of subgoals is done as an optimization. AddressedGoal shares all variables with ICLGoal.

@@What other operators (e.g., negation) might we want to support?

```
\*/
```

```
complete_addressing(RequestingKS, ICLGoal, GlobalParams, AddressedGoal) :-  
    % @@ verify_params(GlobalParams, global, Verified),  
    complete_sources(RequestingKS, ICLGoal, GlobalParams,  
        AddressedGoalWithParamsEverywhere),  
    % @@Here, propagate params, instantiate address request in GlobalParams. ?  
    remove_empty_params(AddressedGoalWithParamsEverywhere, AddressedGoal).
```

```
/*\
```

```
complete_sources(+RequestingKS, +ICLGoal, +GlobalParams, -AddressedGoal).
```

Ensures that every subgoal is explicitly covered by one or more sources. Determines the largest subgoals that can be "chunked"; that is, grouped together for submission to a source.

In the process, every goal acquires a Params field (wherever there was no Params field before, the empty list is added). This is done just to make the definition of complete_sources more readable.

```
\*/
```

```
    % Here we assume that the goal-writer didn't really mean to put a var,  
    % because it's not meaningful to do so:  
complete_sources(KS, Sources:Goal, GlobalParams, AddressedGoal) :-  
    var(Sources),  
    !,  
    complete_sources(KS, Goal, GlobalParams, AddressedGoal).
```

```
/*
```

```
( AddressedGoal = A:_ ->  
    Sources = A  
| otherwise ->  
    findall(A, sub_term(A:_, AddressedGoal), SubSources),  
    % @@More work needed here:  
    Sources = SubSources  
).
```

```
*/
```

```
    % Here we assume that the goal-writer didn't really mean to put [],  
    % because it's not meaningful to do so:
```

```

complete_sources(KS, []:Goal, GlobalParams, AddressedGoal) :-
    !,
    complete_sources(KS, Goal, GlobalParams, AddressedGoal).

    % Sources and Params already specified; we're done:
    % @@But let's verify the sources are valid!
complete_sources(_KS, Sources:Goal::Params, _GlobalParams,
    Sources:Goal::Params) :-
    !.

    % Sources already specified; add empty Params list:
complete_sources(_KS, Sources:Goal, _GlobalParams, Sources:Goal::[]) :-
    !.

    % Sure, we'll continue to support an address in Params or GlobalParams:
complete_sources(KS, Goal::Params, GlobalParams, AddressedGoal) :-
    % @@ verify_params(...),
    ( memberchk(address(Sources), Params) ;
      memberchk(address(Sources), GlobalParams) ),
    \+ var(Sources),
    !,
    complete_sources(KS, Sources:Goal::Params, GlobalParams, AddressedGoal).

    % No Sources or Params specified; add empty Params list before
    % proceeding:
complete_sources(KS, Goal, GlobalParams, AddressedGoal) :-
    \+ (Goal = _::_),
    !,
    complete_sources(KS, Goal::[], GlobalParams, AddressedGoal).

    % Here we get down to the real work: determining solvers and
    % chunking of subgoals:

complete_sources(KS, (\+ Goal1)::Params, GlobalParams, AddressedGoal) :-
    !,
    oaa_Name(Facilitator),
    complete_sources(KS, Goal1, GlobalParams, AddressedGoal1),
    % If S1 is a SINGLE source, it's OK to send the negation to the source.
    % This case also works if S1 == built_in.
    ( (AddressedGoal1 = [S1]:G1::P1,
      S1 \== Facilitator,
      S1 \== facilitator) ->
      AddressedGoal = S1:(\+ G1)::P1)::Params
    | otherwise ->
      AddressedGoal = (\+ AddressedGoal1)::Params
    ).

complete_sources(KS, (Goal1, Goal2, Goal3)::Params, GlobalParams,
    AddressedGoal) :-
    % This clause is needed because we want built_in pred's to be grouped
    % with what comes before, not after.
    !,
    complete_sources(KS, Goal1, GlobalParams, AddressedGoal1),
    complete_sources(KS, Goal2, GlobalParams, AddressedGoal2),
    complete_sources(KS, Goal3, GlobalParams, AddressedGoal3),
    ( (AddressedGoal1 = S1:G1::P1,
      AddressedGoal2 = S2:G2::P2,

```

```

    AddressedGoal3 = S3:G3::P3,
    chunkable_sources([S1, S2, S3], Sources),
    compatible_params([P1, P2, P3])) ->
    AddressedGoal = Sources:(G1::P1, G2::P2, G3::P3)::Params
| (AddressedGoal1 = S1:G1::P1,
    AddressedGoal2 = S2:G2::P2,
    AddressedGoal3 = (S3A:G3A::P3A, Goal3B)::P3,
    % Goal3B may or may not begin with Source:. icl_GoalComponents
    % deals with the precedence issues.
    icl_GoalComponents(Goal3B, _, G3B, P3B),
    chunkable_sources([S1, S2, S3A], Sources),
    append(P3A, P3, NewP3A),
    append(P3B, P3, NewP3B),
    compatible_params([P1, P2, NewP3A])) ->
    AddressedGoal = (Sources:(G1::P1, G2::P2, G3A::NewP3A)::[],
                    G3B::NewP3B)::Params
| (AddressedGoal1 = S1:G1::P1,
    AddressedGoal2 = S2:G2::P2,
    chunkable_sources(S1, S2, Sources),
    compatible_params([P1, P2])) ->
    AddressedGoal = (Sources:(G1::P1, G2::P2)::[], AddressedGoal3)::Params
| (AddressedGoal2 = S2:G2::P2,
    AddressedGoal3 = S3:G3::P3,
    chunkable_sources(S2, S3, Sources),
    compatible_params([P2, P3])) ->
    AddressedGoal = (AddressedGoal1, Sources:(G2::P2, G3::P3)::[])::Params
| (AddressedGoal2 = S2:G2::P2,
    AddressedGoal3 = (S3A:G3A::P3A, Goal3B)::P3,
    icl_GoalComponents(Goal3B, _, G3B, P3B),
    chunkable_sources([S2, S3A], Sources),
    append(P3A, P3, NewP3A),
    append(P3B, P3, NewP3B),
    compatible_params([P2, NewP3A])) ->
    AddressedGoal = (AddressedGoal1, Sources:(G2::P2, G3A::NewP3A)::[],
                    G3B:NewP3B)::Params
| otherwise ->
    AddressedGoal =
        (AddressedGoal1, AddressedGoal2, AddressedGoal3)::Params
).
complete_sources(KS, (Goal1, Goal2)::Params, GlobalParams, AddressedGoal) :-
!,
complete_sources(KS, Goal1, GlobalParams, AddressedGoal1),
complete_sources(KS, Goal2, GlobalParams, AddressedGoal2),
( (AddressedGoal1 = S1:G1::P1,
    AddressedGoal2 = S2:G2::P2,
    chunkable_sources(S1, S2, Sources),
    compatible_params([P1, P2])) ->
    AddressedGoal = Sources:(G1::P1, G2::P2)::Params
| otherwise ->
    AddressedGoal = (AddressedGoal1, AddressedGoal2)::Params
).
% Note: this clause must precede that for disjunction.
complete_sources(KS, (Goal1 -> Goal2 ; Goal3)::Params, GlobalParams,
    AddressedGoal) :-
!,
complete_sources(KS, Goal1, GlobalParams, AddressedGoal1),
complete_sources(KS, Goal2, GlobalParams, AddressedGoal2),

```

```

complete_sources(KS, Goal3, GlobalParams, AddressedGoal3),
( (AddressedGoal1 = S1:G1::P1,
  AddressedGoal2 = S2:G2::P2,
  AddressedGoal3 = S3:G3::P3,
  chunkable_sources([S1, S2, S3], Sources),
  compatible_params([P1, P2, P3])) ->
  AddressedGoal = Sources:(G1::P1 -> G2::P2 | G3::P3)::Params
| otherwise ->
  AddressedGoal =
    (AddressedGoal1 -> AddressedGoal2 | AddressedGoal3)::Params
).
complete_sources(KS, (Goal1 -> Goal2)::Params, GlobalParams, AddressedGoal) :-
!,
complete_sources(KS, Goal1, GlobalParams, AddressedGoal1),
complete_sources(KS, Goal2, GlobalParams, AddressedGoal2),
( (AddressedGoal1 = S1:G1::P1,
  AddressedGoal2 = S2:G2::P2,
  chunkable_sources([S1, S2], Sources),
  compatible_params([P1, P2])) ->
  AddressedGoal = Sources:(G1::P1 -> G2::P2)::Params
| otherwise ->
  AddressedGoal =
    (AddressedGoal1 -> AddressedGoal2)::Params
).
complete_sources(KS, (Goal1 ; Goal2)::Params, GlobalParams, AddressedGoal) :-
!,
complete_sources(KS, Goal1, GlobalParams, AddressedGoal1),
complete_sources(KS, Goal2, GlobalParams, AddressedGoal2),
( (AddressedGoal1 = S1:G1::P1,
  AddressedGoal2 = S2:G2::P2,
  chunkable_sources(S1, S2, Sources),
  compatible_params([P1, P2])) ->
  AddressedGoal = Sources:(G1::P1; G2::P2)::Params
| otherwise ->
  AddressedGoal = (AddressedGoal1; AddressedGoal2)::Params
).
% To be complete, we will allow for this nonstandard goal form:
complete_sources(KS, Goal::Params1::Params2, GlobalParams,
  AddressedGoal::Params2) :-
!,
complete_sources(KS, Goal::Params1, GlobalParams, AddressedGoal).
complete_sources(_KS, Goal::Params, _GlobalParams, built_in:Goal::Params) :-
icl_BuiltIn(Goal),
!.
% Here, finally, we determine the agents (or parent facilitator) that
% can solve a non-compound Goal:
complete_sources(KS, Goal, GlobalParams, Sources:Goal) :-
sources_for_goal(KS, Goal, GlobalParams, Sources).

remove_empty_params(Addr:Goal::[], Addr:NewGoal) :-
!,
remove_empty_params(Goal, NewGoal).
remove_empty_params(Addr:Goal::Params, Addr:NewGoal::Params) :-
!,
remove_empty_params(Goal, NewGoal).
remove_empty_params(Goal::[], NewGoal) :-
!,

```

```

    remove_empty_params(Goal, NewGoal).
remove_empty_params(Goal::Params, NewGoal::Params) :-
    !,
    remove_empty_params(Goal, NewGoal).
remove_empty_params(Sources:Goal, Sources:NewGoal) :-
    !,
    remove_empty_params(Goal, NewGoal).
remove_empty_params((\+ Goal)::[], (\+ NewGoal)) :-
    !,
    remove_empty_params(Goal, NewGoal).
remove_empty_params((Goal1, Goal2), (NewGoal1, NewGoal2)) :-
    !,
    remove_empty_params(Goal1, NewGoal1),
    remove_empty_params(Goal2, NewGoal2).
remove_empty_params((Goal1 ; Goal2), (NewGoal1 ; NewGoal2)) :-
    !,
    remove_empty_params(Goal1, NewGoal1),
    remove_empty_params(Goal2, NewGoal2).
remove_empty_params((Goal1 -> Goal2), (NewGoal1 -> NewGoal2)) :-
    !,
    remove_empty_params(Goal1, NewGoal1),
    remove_empty_params(Goal2, NewGoal2).
% Primitive (non-compound) goal:
remove_empty_params(Goal, Goal).

remove_addresses(_Sources:Goal, NewGoal) :-
    !,
    remove_addresses(Goal, NewGoal).
remove_addresses((Goal1, Goal2), (NewGoal1, NewGoal2)) :-
    !,
    remove_addresses(Goal1, NewGoal1),
    remove_addresses(Goal2, NewGoal2).
remove_addresses((Goal1 ; Goal2), (NewGoal1 ; NewGoal2)) :-
    !,
    remove_addresses(Goal1, NewGoal1),
    remove_addresses(Goal2, NewGoal2).
remove_addresses((Goal1 -> Goal2), (NewGoal1 -> NewGoal2)) :-
    !,
    remove_addresses(Goal1, NewGoal1),
    remove_addresses(Goal2, NewGoal2).
% Primitive (non-compound) goal:
remove_addresses(Goal, Goal).

/*\

```

```

chunkable_sources(+Sources1, +Sources2, -Sources).

```

Each argument is either: a single KS name (or numeric id); a list of KS names (where 'facilitator' or 'parent' also count as KS names), or the atom 'built_in'. (Empty list is OK.)

Sources1 gives the sources that can solve some goal, Sources2 gives the sources that can solve some other goal, and if this pred. succeeds, Sources gives a set of sources that can solve both together.

NOTES ON CHUNKING:

%1 A chunk is a sub-goal SG of a Goal such that
 (1) There is a nonempty set S of client agents each of which can solve the entire chunk (that is, every predicate in the chunk is either an icl_BuiltIn or one of the agent's solvables), and
 (2) Performing the subgoal as (ks1:SQ ; ks2:SQ ; ... ; ksN:SQ), where ks1 ... ksN are all the agents in S, does not in any way violate the intended semantics of the overall Goal.

NOTE: chunking is done "conservatively", so as to preserve Prolog semantics. So, for example, the following Goal:

```
(a(1), b(2)),
```

where a and b are both solvable by ks1 and ks2, will be chunked as follows:

```
chunk(a(1), [ks1, ks2]), chunk(b(2), [ks1, ks2])
```

which amounts to no chunking at all, instead of

```
chunk((a(1), b(2)), [ks1, ks2]).
```

The former results in execution

```
(ks1:a(1) ; ks2:a2), (ks1:b(2) ; ks2:b(2))
```

whereas the latter would result in execution

```
ks1:(a(1), b(2)) ; ks2:(a(1), b(2))
```

We might want to explore under what conditions more extensive chunking can be done.

```
\*/
```

```
% This just allows for single sources, not in a list:
```

```
chunkable_sources(Source1, Source2, Sources) :-
```

```
( atomic(Source1) ->
```

```
  S1 = [Source1]
```

```
| otherwise ->
```

```
  S1 = Source1
```

```
),
```

```
( atomic(Source2) ->
```

```
  S2 = [Source2]
```

```
| otherwise ->
```

```
  S2 = Source2
```

```
),
```

```
chunkable_srcs(S1, S2, Sources).
```

```
chunkable_srcs(built_in, Sources, Sources) :-
```

```
% at least one element:
```

```
Sources = [_ | _],
```

```
!.
```

```
chunkable_srcs(Sources, built_in, Sources) :-
```

```
Sources = [_ | _],
```

```
!.
```

```
chunkable_srcs([], [], []) :-
```

```
!.
```

```
chunkable_srcs([Source], [Source], [Source]) :-
```

```
!.
```

```
chunkable_srcs([Source1], [Source2], [Source1]) :-
```

```
( number(Source1), atom(Source2) ;
```

```
  number(Source2), atom(Source1) ),
```

```
!,
```

```
find_address(Source1, Source),
```

```
find_address(Source2, Source).
```

```

% chunkable_sources(+SourcesIn, -SourcesOut).
%   Does the same as chunkable_sources/3, but allows for a list
% of sources (length >= 1) as arg 1.

chunkable_sources([Sources], Sources).
chunkable_sources([Sources1, Sources2 | RestSources], SourcesOut) :-
    chunkable_sources(Sources1, Sources2, SourcesTemp),
    chunkable_sources([SourcesTemp | RestSources], SourcesOut).

% compatible_params(+ParamLists).
%   ParamLists is a list of 2 or more ParamLists. This predicate
% succeeds IFF the ParamLists are compatible for purposes of
% chunking.
compatible_params(_).

% sources_for_goal(+RequestingKS, +Goal, +Params, -Sources).
% @@ Here, depending on how the treatment of multiple facilitators evolves,
% we may need to revisit the default use of the facilitator.

sources_for_goal(RequestingKS, ICLGoal, GlobalParams, Sources) :-
    icl_GoalComponents(ICLGoal, _, Goal, Params),
    append(Params, GlobalParams, AllParams),
    findall(SomeKS,
        choose_ks_for_goal(RequestingKS, Goal, _, AllParams, SomeKS, _),
        KSList),
    ( KSList = [] ->
        % @@Determine if there's a parent facilitator that can handle
        % the goal. This needs work; probably should have a local record
        % of what the parent can handle.
        find_level(AllParams, Level, _NewParams),
        ( (on_exception(_, com:com_GetInfo(parent, fac_id(ParentBB)), fail), Level
        > 0) ->
            Sources = [ParentBB]
        | otherwise ->
            Sources = []
        )
    | otherwise ->
        Sources = KSList
    ).

% If Sources is bound, VERIFIES that all the Sources can be used
% on the ICLGoal. If var(Sources), finds all the Sources that can
% be used.

% sources_for_compound_goal(RKS, ICLGoal, GlobalParams, Sources) :-

/*\
complete_concurrency(+Goal, -ConcurrentGoal).

TBD.

\*/
complete_concurrency(Goal, Goal).

```



```
*****
% GOAL EXECUTION: TOP LEVEL
*****
```

```
/*\
    execute_goal(+RequestingKS, +OrigGoal, +OrigParams, +CompleteGoal).
```

OrigGoal are OrigParams are exactly as submitted by some client agent (RequestingKS). CompleteGoal is the rewriting of OrigGoal that ensures complete addressing. OrigGoal and ICLGoal contain precisely the same var's.

See global comments near the top of this file.

Note: the meaning of variable "Goal" and other variables ending in "Goal" varies with context. In some places they indicate an ICL goal Source:Goal::Params (where Source and Params are both optional); in other places, they indicate just the Goal part of an ICL goal.

```
\*/
```

```
execute_goal(RKS, OrigGoal, OrigParams, ICLGoal) :-
    % Here, ICLGoal may or may not include a Sources component. Either
    % way, it gets handled by execute/7.
    % @@ What if OrigGoal's Params or GlobalParams has vars?
    % We remove addresses before calling term_vars only so as to avoid
    % a syntax error exception that comes up when ICLGoal = Addr:\+Goal
    remove_addresses(ICLGoal, TempGoal),
    term_vars(TempGoal, AllVars, _Singletons, _NonSingletons),
    new_goal_id(Id),
    % This means simply, "When the Solvers and solutions (in the form of
    % Bindings for AllVars) are known for Goal, call
    % unify_and_return_solutions(...)."
    assert(continuation(Id, Requestees, Solvers, Bindings,
        unify_and_return_solutions(Id,RKS,OrigGoal,OrigParams,AllVars,
            Requestees,Solvers,Bindings))),
    % This means: Find the Solvers and solutions:
    execute(Id, RKS, [], [], ICLGoal, OrigParams, AllVars).
```

```
/*\
*    execute(Id, RKS, Requestees, Solvers, Goal, InheritedParams, Vars).
```

execute/7 satisfies the ICL goal Goal. Id is an integer that identifies a continuation assertion. When the satisfaction of Goal has been completed, the continuation assertion tells what to do next. The satisfaction of Goal may be very simple, or may involve a number of steps, depending on the form of Goal.

Requestees is a list of source id's of all sources asked to participate in the satisfaction of whatever request contained Goal, and Solvers is a list of source id's of sources that succeeded in satisfying some part of the request (so Solvers is a subset of Requestees. These lists are being accumulated for return to the agent that submitted the request.

Conceptually, execute/7 does this:

```

    findall(Vars, Goal, Bindings),
    append(Requestees, <list of KSs called on in the findall>, NewRequestees),
    append(Solvers, <list of KSs providing solutions in the findall>,
           NewSolvers),
    continue_execution(Id, RKS, NewRequestees, NewSolvers, Bindings)

```

The behavior of `continue_execution`, then, depends on a continuation/5 assertion, with `Id` as the first arg.

The important details have to do with how the satisfaction of the "findall" part of this strategy may be delayed.

```

*
\*/

```

```

execute(Id, RKS, Requestees, Solvers, built_in:ICLGoal, InheritedParams, Vars) :-

```

```

    % This handles ICL built-ins, such as <, >, =, member/2, true, false, ...
    !,
    icl_GoalComponents(ICLGoal, _, Goal, Params),
    append(Params, InheritedParams, AllParams),
    oaa_Name(Facilitator),
    add_element(Facilitator, Requestees, NewRequestees),
    % If the requestor wants to know the solver, bind it here:
    ( memberchk(get_address(Facilitator), Params) -> true | true),

    ( oaa:passes_tests(Params) ->
      % @@The use of solution_limit and elsewhere here needs a close look:
      ( memberchk(solution_limit(N), AllParams) ->
        oaa:findNSolutions(N, Vars, call(Goal), Bindings)
      | otherwise ->
        findall(Vars, call(Goal), Bindings)
      )
    | otherwise ->
      Bindings = []
    ),
    ( Bindings == [] ->
      NewSolvers = Solvers
    | otherwise ->
      add_element(Facilitator, Solvers, NewSolvers)
    ),
    ( memberchk(reply(none), AllParams) ->
      continue_execution(Id, RKS, NewRequestees, NewSolvers, [Vars])
    | otherwise ->
      continue_execution(Id, RKS, NewRequestees, NewSolvers, Bindings)
    ).

```

```

    % Empty list of sources:
execute(Id, RKS, Requestees, Solvers, []:ICLGoal, _InheritedParams, _Vars) :-
    format('WARNING: No solvers for ICL goal or subgoal:-n ~q-n',
           ICLGoal),
    continue_execution(Id, RKS, Requestees, Solvers, []).

```

```

    % Single KS in a list:
execute(Id, RKS, Requestees, Solvers, [KS]:G, Params, Vars) :-
    !,

```

```

execute(Id, RKS, Requestees, Solvers, KS:G, Params, Vars).

% Multiple KSs in a list:
execute(Id, RKS, Requestees, Solvers, [KS | Rest]:G, Params, Vars) :-
!,
execute_for_each_ks(Id, RKS, Requestees, Solvers, G, Params,
Vars, [KS | Rest]).

% Solver is facilitator (me):
execute(Id, RKS, Requestees, Solvers, Source:ICLGoal, InheritedParams, Vars) :-
oaa_Name(Facilitator),
(Source = facilitator ; Source = Facilitator),
!,
icl_GoalComponents(ICLGoal, _, Goal, Params),
% If the requestor wants to know the solver, bind it here:
( memberchk(get_address(Facilitator), Params) -> true | true),
append(Params, InheritedParams, AllParams),
findall(Vars,
oaa:oaa_solve_local(Goal, InheritedParams,
Bindings),
( memberchk(reply(none), AllParams) ->
true
| otherwise ->
oaa_Name(KSName),
add_element(KSName, Requestees, NewRequestees),
( Bindings == [] ->
NewSolvers = Solvers
| otherwise ->
add_element(KSName, Solvers, NewSolvers)
),
continue_execution(Id, RKS, NewRequestees, NewSolvers, Bindings)
).

% Note: this code was inherited from pre-compound-query facilitator.
% One significant change: when a goal is sent to a parent, we used to
% automatically include local blackboard solutions also. We don't
% do this anymore.
%
% @@ Strategy should be re-evaluated at some point. For instance,
% the use of var P2 might now cause things to break (the requesting
% agent might try to unify its copy of Params with P2).

execute(Id, RKS, Requestees, Solvers, Sources:ICLGoal, InheritedParams, Vars) :-
on_exception(_, com:com_GetInfo(parent, fac_id(ParentBB)), fail),
(Sources == parent ; Sources == ParentBB),
!,
icl_GoalComponents(ICLGoal, _, _Goal, Params),
% If the requestor wants to know the solver, bind it here:
% NO - it gets bound by the parent facilitator.
% ( memberchk(get_address(ParentBB), Params) -> true | true),

append(Params, InheritedParams, AllParams),
% We don't need to check the level here (that's already been done),
% but we do need to decrement its value by 1:
find_level(AllParams, _Level, NewParams),
oaa_TraceMsg('-nRouting goal "solve(-p)" to parent -p.-n',

```

```

[ICLGoal, ParentBB]),
new_goal_id(NewId),
oaa_PostEvent(ev_post_solve_from_bb(NewId, ICLGoal, NewParams),
              [address(ParentBB)]),
( memberchk(reply(none), NewParams) ->
  unify_and_continue_execution(Id, RKS, ICLGoal, Vars,
    ParentBB, Requestees, Solvers, [ICLGoal])
| otherwise ->
  % @@Shouldn't there be a time-check here?
  oaa:oaa_add_trigger_local(
    comm,
    event(ev_reply_solved_by_bb(NewId, _KS, ICLGoal, _P2,
      Solutions),
      _),
    ev_unify_and_continue_execution(Id, RKS, ICLGoal, Vars,
      ParentBB, Requestees, Solvers, Solutions),
    [recurrence(when), on(receive)])
).

% Send the goal to an agent:
execute(Id, RKS, Requestees, Solvers, KS:ICLGoal, InheritedParams, Vars) :-
!,
icl_GoalComponents(ICLGoal, _, Goal, Params),
append(Params, InheritedParams, AllParams),
% @@What if the KS' status has changed since it was specified?
% find_address allows for KS to be either numeric or symbolic.
find_address(KS, KSId),
% If the requestor wants to know the solver, bind it here:
( memberchk(get_address(KSId), Params) -> true | true),
% Could do another check of the agent's validity:
% ks_ready(KSId, _),

% relevant_vars(Vars, Goal, GVars),
% OptimizedG = findall(GVars, Goal, All),

% Output trace message:
( oaa:oaa_trace(on) ->
  copy_term(ICLGoal, TraceCopy),
  numbervars(TraceCopy, 0, _),
  copy_term(InheritedParams, ParamsCopy),
  numbervars(ParamsCopy, 0, _),
  oaa_TraceMsg(
    '% Routing goal to -w:-n%   -w ~w~n~n',
    [KS, TraceCopy, ParamsCopy])
| otherwise ->
  true
),

new_goal_id(NewId),
% oaa_PostEvent(KS, RKS, solve(NewId, OptimizedG::Params, [])),
oaa_PostEvent(ev_solve(NewId, ICLGoal, InheritedParams),
              [from(RKS), address(KSId)]),

( memberchk(reply(none), AllParams) ->
  unify_and_continue_execution(Id, RKS, ICLGoal, Vars,
    KSId, Requestees, Solvers, [ICLGoal])
  % If time_limit specified in parameters, setup

```

```

    % time_trigger to wakeup if solutions hasn't been returned
    % in specified time.
| otherwise ->
  ( memberchk(time_limit(NSecs), AllParams) ->
    add_time_check(NSecs, NewId, RKS, Goal, AllParams)
  | true),
oaa:oaa_add_trigger_local(
  comm,
  event(ev_solved(NewId, _KS, ICLGoal, _P2, Solutions), _),
  ev_unify_and_continue_execution(Id, RKS, ICLGoal, Vars,
    KSId, Requestees, Solvers, Solutions),
  [recurrence(when), on(receive)])
% poll_until_all_events([solved(Id, _KS, OptimizedG, P2, Solutions)]),
% Solutions = [findall(GVars, Goal, All)],
% respond_query(Id, RKS, Solvers, KS, Goal, P2, Solutions)
% Backtrack over solutions:
% member(GVars, All).
).

% Negation:
execute(Id, RKS, Requestees, Solvers, ICLGoal, InheritedParams, Vars) :-
  icl_GoalComponents(ICLGoal, _, (\+ G1), Params),
  !,
  append(Params, InheritedParams, NewIParams),
  new_goal_id(NewId),
  assert(
    continuation(NewId, NewRequestees, NewSolvers, Bindings,
      continue_negation(Id, RKS, NewRequestees, NewSolvers, NewIParams,
        Vars, Bindings))),
  execute(NewId, RKS, Requestees, Solvers, G1, NewIParams, Vars).

% Conjunction:
execute(Id, RKS, Requestees, Solvers, ICLGoal, InheritedParams, Vars) :-
  icl_GoalComponents(ICLGoal, _, (G1, G2), Params),
  !,
  append(Params, InheritedParams, NewIParams),
  new_goal_id(NewId),
  assert(
    continuation(NewId, NewRequestees, NewSolvers, Bindings,
      continue_conjunction(Id, RKS, NewRequestees, NewSolvers, G2,
NewIParams,
        Vars, Bindings))),
  execute(NewId, RKS, Requestees, Solvers, G1, NewIParams, Vars).

% Local cut with alternative. Note: this clause must precede
% that for disjunction.
execute(Id, RKS, Requestees, Solvers, ICLGoal, InheritedParams, Vars) :-
  icl_GoalComponents(ICLGoal, _, (G1 -> G2 | G3), Params),
  !,
  append(Params, InheritedParams, NewIParams),
  new_goal_id(NewId),
  assert(
    continuation(NewId, NewRequestees, NewSolvers, Bindings,
      continue_local_cut(Id, RKS, NewRequestees, NewSolvers, G2, G3,
NewIParams,
        Vars, Bindings))),
  execute(NewId, RKS, Requestees, Solvers, G1, NewIParams, Vars).

```

```

% Local cut:
execute(Id, RKS, Requestees, Solvers, ICLGoal, InheritedParams, Vars) :-
    icl_GoalComponents(ICLGoal, _, (G1 -> G2), Params),
    !,
    append(Params, InheritedParams, NewIPParams),
    new_goal_id(NewId),
    assert(
        continuation(NewId, NewRequestees, NewSolvers, Bindings,
            continue_local_cut(Id, RKS, NewRequestees, NewSolvers, G2, false,
NewIPParams,
                Vars, Bindings))),
    execute(NewId, RKS, Requestees, Solvers, G1, NewIPParams, Vars).

% Disjunction:
execute(Id, RKS, Requestees, Solvers, ICLGoal, InheritedParams, Vars) :-
    icl_GoalComponents(ICLGoal, _, (G1; G2), Params),
    !,
    append(Params, InheritedParams, NewIPParams),
    new_goal_id(Id1),
    new_goal_id(Id2),
    assert(
        multiple_continuation([Id1, Id2], Requestees, AllRequestees,
            Solvers, AllSolvers,
            [], AllBindings,
            continue_execution(Id, RKS, AllRequestees, AllSolvers, AllBindings))),
    execute(Id1, RKS, Requestees, Solvers, G1, NewIPParams, Vars),
    execute(Id2, RKS, Requestees, Solvers, G2, NewIPParams, Vars).

% Occasionally, a goal may have the form G::P (that is, no
% address, and P is not compound), but it is still valid, so
% long as G is valid.
%
% Ex.: ([7]:a1(1)::[...])::[...]
execute(Id, RKS, Requestees, Solvers, Goal::Params, InheritedParams, Vars) :-
    !,
    append(Params, InheritedParams, NewIPParams),
    execute(Id, RKS, Requestees, Solvers, Goal, NewIPParams, Vars).

execute(Id, RKS, Requestees, Solvers, G, _Params, _Vars) :-
    format('WARNING (execute/7): unrecognized goal form:-n    -w-n', [G]),
    continue_execution(Id, RKS, Requestees, Solvers, []).

execute_for_each_ks(Id, RKS, Requestees, Solvers, Goal, Params, Vars, KSs) :-
    length(KSs, NumKSs),
    new_goal_ids(NumKSs, Ids),
    assert(
        multiple_continuation(Ids, Requestees, AllRequestees, Solvers,
AllSolvers, [], AllBindings,
            continue_execution(Id, RKS, AllRequestees, AllSolvers, AllBindings))),
    exec_for_each_ks(NumKSs, Ids, KSs, RKS, Requestees, Solvers, Goal,
        Params, Vars).

*****
% GOAL EXECUTION: INTERMEDIATE STEPS
% The predicates in this group define intermediate steps in the satisfaction
% of various ICL goal forms.

```

```

%
% Note: intermediate steps in handling of DISJUNCTION are handled by
% continue_execution, using the multiple_continuation assertion.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This is used in satisfying [KS1, KS2, ...]:Goal. Note that this is
% equivalent to a disjunction (KS1:Goal ; KS2:Goal ; ...). So we
% are able to use the multiple_continuation assertion to accumulate
% the solutions.
%
% We don't need Solvers, because ...

exec_for_each_ks(NumKSSs, Ids, KSs, RKS, _Requestees, _Solvers,
                Goal, Params, Vars) :-
    retractall( ks_num(_) ),
    assert( ks_num(1) ),
    repeat,
    ks_num(Num),
    ( Num > NumKSSs ->
      !
      | otherwise ->
        nth1(Num, KSs, KS),
        nth1(Num, Ids, Id),
        % We use a local cut to prevent some (harmless) backtracking.
        % This is one place where we don't need to pass Requestees and
        % Solvers through to execute (3rd and 4th args), because they are
        % filled in by handle_multiple_continuation.

        ( execute(Id, RKS, [], [], KS:Goal, Params, Vars) -> true ),
        NextNum is Num + 1,
        retractall( ks_num(_) ),
        assert( ks_num(NextNum) ),
        fail
    ).

% This is used in satisfying (\+ Goal). When this
% pred. is called, Goal has just been completed. Bindings gives
% the solutions to Goal.

continue_negation(Id, RKS, Requestees, Solvers, _Params, Vars, []) :-
    !,
    continue_execution(Id, RKS, Requestees, Solvers, [Vars]).
continue_negation(Id, RKS, Requestees, Solvers, _Params, _Vars, _Bindings) :-
    continue_execution(Id, RKS, Requestees, Solvers, []).

% This is used in satisfying (Goal1, Goal2). When this
% pred. is called, Goal1 has just been completed. Bindings gives
% the solutions to Goal1.

continue_conjunction(Id, RKS, Requestees, Solvers, _Goal2, _Params, _Vars, [])
:-
    !,
    continue_execution(Id, RKS, Requestees, Solvers, []).
continue_conjunction(Id, RKS, Requestees, Solvers, Goal2, Params, Vars,
Bindings) :-
    length(Bindings, NumBindings),
    new_goal_ids(NumBindings, Ids),

```

```

    assert(
        multiple_continuation(Ids, Requestees, AllRequestees, Solvers,
            AllSolvers, [], AllBindings,
            continue_execution(Id, RKS, AllRequestees, AllSolvers, AllBindings))),
        exec_for_each_binding(NumBindings, Ids, Bindings, RKS, Requestees, Solvers,
            Goal2,
                Params, Vars).

    % We don't need Requestees or Solvers, because they are filled in
    % by handle_multiple_continuation.

exec_for_each_binding(NumBindings, Ids, Bindings, RKS, _Requestees, _Solvers,
    Goal, Params, Vars) :-
    retractall( binding_num(_) ),
    assert( binding_num(1) ),
    repeat,
    binding_num(Num),
    ( Num > NumBindings ->
        !
    | otherwise ->
        nth1(Num, Bindings, Binding),
        nth1(Num, Ids, Id),
        Vars = Binding,
        % We use a local cut to prevent some (harmless) backtracking.
        % This is one place where we don't need to pass Solvers through
        % to execute (3rd arg):
        ( execute(Id, RKS, [], [], Goal, Params, Binding) -> true ),
        NextNum is Num + 1,
        retractall( binding_num(_) ),
        assert( binding_num(NextNum) ),
        fail
    ).

    % This is used in satisfying Goal1 -> Goal2 | Goal3. When this
    % pred. is called, Goal1 has just been completed. Bindings gives
    % the solutions to Goal1.

    % No solutions to Goal1:
continue_local_cut(Id, RKS, Requestees, Solvers, _Goal2, Goal3, Params,
    Vars, []) :-
    !,
    ( Goal3 = false ->
        continue_execution(Id, RKS, Requestees, Solvers, [])
    | otherwise ->
        execute(Id, RKS, Requestees, Solvers, Goal3, Params, Vars)
    ).
    % Some solutions:
continue_local_cut(Id, RKS, Requestees, Solvers, Goal2, _Goal3, Params,
    Vars, [Binding1 | _]) :-
    new_goal_id(NewId),
    assert(
        continuation(NewId, NewRequestees, NewSolvers, Bindings,
            continue_execution(Id, RKS, NewRequestees, NewSolvers, Bindings))),
    Vars = Binding1,
    % local cut to prevent some (harmless) backtracking:
    ( execute(NewId, RKS, Requestees, Solvers, Goal2, Params, Binding1) -> true
    ).

```



```

*****
% GOAL EXECUTION: COMPLETION
*****

```

```

% This is called when the goal associated with Id has been completely
% satisfied.

```

```

continue_execution(Id, _RKS, Requestees, Solvers, Bindings) :-
    % Here we are BINDING the Solvers and Bindings var's. in the
    % continuation assertion. The var. also appears in Continuation:
    ( retract(continuation(Id, Requestees, Solvers, Bindings, Continuation)) ->
      call(Continuation)
    | multiple_continuation(Ids, _, _, _, _, _),
      memberchk(Id, Ids) ->
      handle_multiple_continuation(Id, Requestees, Solvers, Bindings, Ids)
    | otherwise ->
      format('Internal Error: no continuation with id -w-n', [Id])
    ).

```

```

handle_multiple_continuation(Id, Requestees, Solvers, Bindings, Ids) :-
    retract(multiple_continuation(Ids, PrevRequestees,
                                   AllRequestees, PrevSolvers, AllSolvers,
                                   PrevBindings, AllBindings,
                                   Continuation)),
    del_element(Id, Ids, NewIds),
    append(PrevBindings, Bindings, NewBindings),
    append(PrevRequestees, Requestees, NewRequestees),
    append(PrevSolvers, Solvers, NewSolvers),
    ( NewIds = [] ->
      AllBindings = NewBindings,
      AllRequestees = NewRequestees,
      AllSolvers = NewSolvers,
      call(Continuation)
    | otherwise ->
      assert(multiple_continuation(NewIds, NewRequestees, AllRequestees,
                                   NewSolvers, AllSolvers,
                                   NewBindings, AllBindings,
                                   Continuation))
    ).

```

```

% @@Let's see, if these args included the vars for any
% nested solvers params, we could probably instantiate solvers
% params in Goal...

```

```

unify_and_continue_execution(Id, RKS, Goal, Vars, Requestee, Requestees,
                             Solvers, Solutions) :-
    add_element(Requestee, Requestees, NewRequestees),
    ( Solutions == [] ->
      NewSolvers = Solvers
    | otherwise ->
      add_element(Requestee, Solvers, NewSolvers)
    ),
    findall(Vars,
            member(Goal, Solutions),
            Bindings),
    continue_execution(Id, RKS, NewRequestees, NewSolvers, Bindings).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% GENERAL UTILITIES
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

term_vars(Term, AllVars, SingletonVars, NonSingletonVars) :-
    with_output_to_chars(portray_clause(Term), Chars),
    with_input_from_chars(
        read_term([variable_names(Names), singletons(Singletons)],
            Term1),
        Chars),
    extract_vars(Names, Singletons, AllVars, SingletonVars, NonSingletonVars),
    Term = Term1.

extract_vars([], _Singletons, [], [], []).
extract_vars([Name = Var | RestNames], Singletons, [Var | RestVars],
    [Var | RestSV], NonSingletonVars) :-
    memberchk(Name = Var, Singletons),
    !,
    extract_vars(RestNames, Singletons, RestVars, RestSV, NonSingletonVars).
extract_vars([_Name = Var | RestNames], Singletons, [Var | RestVars],
    RestSV, [Var | NonSingletonVars]) :-
    extract_vars(RestNames, Singletons, RestVars, RestSV, NonSingletonVars).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DEBUGGING UTILITIES
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% static_test :-
%     Class = root,
%     KSName = dontcare,
%     BBName = dontcare,
%     oaa_read_setup_file,
%     oaa_init_flags,
%     assert(oaa_class(Class)),
%     oaa_SetupCommunication(Class, KSName, BBName, []),
%     on_exception(_, oaa_AppInit, true),
%     oaa_Ready(true).
%
% connect :-
%     % go(leaf, shell, root).
%     static_test.
%
% ce :-
%     repeat,
%     oaa_GetEvent(CallingKS, Event, 0),
%     ( Event = timeout ->
%     !,
%     format('No events-n', [])
%     | otherwise ->
%     oaa_process_event(CallingKS, Event),
%     fail
%     ).
% ce :-
%     format('No events-n', []).
%

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% OrigGoal must be used in the return event, so that the
% requesting KS will identify it correctly.
```

```
unify_and_return_solutions(Id,RKS,OrigGoal,OrigParams,Vars,Requestees,Solvers,Bi
ndings) :-
    findall(OrigGoal,
            member(Vars, Bindings),
            Solutions),
    oaa_TraceMsg('~nRouting answers back to -p:-n  -p~n',
                [RKS,Solutions]),
    cancel_time_check(Id),
    remove_dups(Requestees, RequesteesSet),
    remove_dups(Solvers, SolversSet),
    % If present, bind solvers request in OrigParams:
    ( memberchk(get_address(RequesteesSet), OrigParams) -> true | true ),
    ( memberchk(get_satisfiers(SolversSet), OrigParams) -> true | true ),
    oaa_PostEvent(ev_reply_solved(RequesteesSet, SolversSet, OrigGoal,
OrigParams, Solutions),
                [address(RKS)]).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

APPENDIX A.II

Source code file named fac.pl.



```

%*****
% File      : fac.pl
% Primary Authors  : Adam Cheyer, David Martin
% Purpose   : Provides communications and coordination of the activities
%             of a dynamic collection of client agents.
% Updated   : 12/98
%
% -----
% Unpublished-rights reserved under the copyright laws of the United States.
%
% Unpublished Copyright (c) 1998, SRI International.
% "Open Agent Architecture" and "OAA" are Trademarks of SRI International.
% -----
%
%*****
% fac.pl : the facilitator agent          Adam Cheyer
%                                             David Martin
%
% Provides communications and coordination of the activities of a
% dynamic collection of client agents.
%
% The blackboard can respond to the following external requests:
%
%   ev_post_event(AgentID, Cmd)   : sends event to the agent
%   ev_post_event(Cmd)           : sends event to all
%   ev_post_declare(Mode, Solvables, Params)
%                               : adds, removes or replaces solvables ON
%                               : the facilitator
%   ev_post_update(Mode, Clause, Params)
%                               : adds, removes, or replaces data
%                               : on appropriate agents
%   ev_post_trigger_update(Mode, TriggerType, Condition, Action, Params)
%                               : adds or removes a trigger
%                               : on appropriate agents
%   ev_post_solve(Goal, Params) : finds agent(s) to solve Goal
%   connected(Connection)       : records that a client agent has connected
%   ev_connect(AgentInfo)
%                               : additional information from a client
%                               : agent (having version > 3.0)
%   end_of_file(Connection)     : records that a client has closed its
%                               : connection
%   ev_register_solvables       : records the goals that an agent can solve.
%
% A facilitator uses the following events internally as trigger actions:
%
%   ev_respond_query(Id, ToKS, ByKS, G, OrigParams, Params, S)
%                               : Sends the result of a query back to KS
%
%*****
:- use_module(library(lists)).
:- use_module(library(basics)).
:- use_module(library(strings)).
:- use_module(library(charsio)).

```

```

:- use_module(library(sets)).
:- use_module(library(samsort)). % for samsort(Ordered,Raw,Sort)
:- use_module(library(tcp), [tcp_now/1, tcp_time_plus/3,
                             tcp_schedule_wakeup/2, tcp_cancel_wakeup/2]).

% The file containing the com module is normally specified here. For
% more info, see comments near the top of oaa.pl.
:- use_module(com_tcp, all).
:- use_module(oaa, all).

% Whether or not to load translations and compound query code
% is determined right here:
% :- [compound].
:- [translations].

:- multifile oaa_AppDoEvent/2.

:- dynamic time_limit_trigger/5. % time_limit_trigger(Id,When,KS,Goal,Params)
:- dynamic goal_count/10.      % goal_count(GoalId,Goal,Params,EvParams,
                                   % ToBeCalled,Called,Responders,Solvers,
                                   % Answers,NumAnswers)
:- dynamic update_count/4.     % update_count(GoalId,NumAgentsRequested,
                                   % KSs, Updaters)
initial_solvables([
  solvable(agent_data(_Id, _Status, _Solvables, _Name), [type(data)],
            [write(true)]),
  % Locations of all facilitators (currently maintained only by the 'root'
  % facilitator:
  solvable(agent_location(_Id2, _Name2, _Host2, _Port2), [type(data)],
            [write(true)]),
  % Host (if known) of each client agent:
  solvable(agent_host(_Id3, _Name3, _Host3), [type(data)], [write(true)]),
  agent_version(_Id1, _Language1, _Version1),
  can_solve(_Goal4, _IdList4),
  % For backwards compatibility. In translations.pl, some events
  % (write_bb, etc.) specify updates to this solvable. Also, old-style
  % data triggers refer to it:
  solvable(data(_Item, _Data), [type(data)], [write(true)])
]).

/* Agent specific declarations */

oaa_AppInit :-
  oaa_SetTimeout(0).

/* This is the event generated by the TCP library. Will be followed
   immediately by ev_connect/4, which is constructed by the client agent */
oaa_AppDoEvent(connected(Connection), _) :-
  !,
  format('-nKnowledge source connected: -p-n-n', [Connection]),
  Id = Connection,
  oaa:oaa_add_data_local(agent_data(Id, open, [], Id), []),
  %% Maintain information of currently connected data.
  add_connected(Id, Connection).

```

```

/* For now, the ID of a client agent is the same as its connection (socket).
   This could change in the future, so we store Id and Connection
   as two separate entities. */
oaa_AppDoEvent(ev_connect(AgentInfoList), Params) :-
    memberchk( connection_id(Id), Params),
    oaa_Name(MyName),
    oaa_Id(MyId),
    MyLanguage = prolog,
    oaa_LibraryVersion(MyVersion),

    update_connected(Id, AgentInfoList),

    % preferred TCP transfer mechanism
    MyFormat = quintus_binary,

    % Inform the client of his Id, and info about me.
    com_SendData(Id,
        event(ev_connected([oaa_id(Id), fac_id(MyId), fac_name(MyName),
            fac_lang(MyLanguage), fac_version(MyVersion),
            format(MyFormat)]),
            [])).

/* Removes meta-data for KS when the KS disconnects */
oaa_AppDoEvent(end_of_file(Connection), _) :-
    Id = Connection,
    remove_connected(Id),
    oaa:oaa_remove_data_local(agent_data(Id, _Status, _Solvable, AgentName),
[]),
    format('-nKnowledge source disconnected: -p (-p)-n-n', [Id,AgentName]),
    % remove all facts written by the agent
    % TBD: Is this getting all relevant triggers (see commented code below)?
    oaa:oaa_remove_data_owned_by(Id),

% Do we really want to do this? I think clients who are interested could
% register a trigger on the agent_data predicate.
% Rather, I think we should check to see if any agents are currently waiting
% for this agent to solve some goal -- if the agent disconnects, we can assume
% that it won't be solving the goal anytime soon, and we should send back
% failure to the requesting agent. See OAA 1.0 Facilitator, end_of_file()
% method. [AJC, 11/24/97]
    post_to_all_clients(ev_agent_disconnected(Id)).

% fail.
% TBD: This needs update to look at the persistence param.
% oaa_AppDoEvent(end_of_file(KS), _) :-
%     % remove all triggers for KS
%     on_exception(_, trigger(KS, Type, Kind, OpMask, Template, Cond, Action),
fail),
%     retract(trigger(KS, Type, Kind, OpMask, Template, Cond, Action)),
%     fail.
% oaa_AppDoEvent(end_of_file(_KS), _) :- !.

oaa_AppDoEvent(ev_ready(Name), Params) :-
    memberchk(from(Id), Params),
    % TBD: Let's have an error message if this fails:
    oaa:oaa_remove_data_local(agent_data(Id, _OldStatus, Solvables, _Name),
Params),

```

```

    oaa:oaa_add_data_local(agent_data(Id, ready, Solvables, Name), Params).

/* Stores the goals that a KS knows how to solve */
% Is this obsolete?
oaa_AppDoEvent(ev_register_solvables(Goals), Params) :-
    memberchk(from(KS), Params),
    oaa_AppDoEvent(ev_register_solvables(add,Goals,KS,[]), Params), !.

% IMPORTANT: We assume the Solvables are in standard form and can
% legally be added/removed/replaced for this agent. Also, we take
% care to keep the facilitator's copy of each client's solvables
% identical to that stored at the client. (Compare to code in
% liboaa.pl, pred. oaa_declare_local).
oaa_AppDoEvent(ev_register_solvables(Mode,Solvs,AgentName,EvParams), Params) :-
    memberchk(from(KS), Params),
    oaa_Name(KSName),
    (oaa:oaa_remove_data_local(agent_data(KS, Status, List, _AgentName),
Params)
    ;
    format('STRANGE! register_solvables called by unknown KS!!!: -p-n',
    [KS]),
    Status = ready,
    List = []
    ),
    icl_ConvertSolvables(PrettySolvs, Solvs),
    ( Mode == add, memberchk(if_exists(overwrite), EvParams) ->
        NewList = Solvs,
        format('-p (-p) can solve: -n -p-n-n', [KS, AgentName,
        PrettySolvs])
    | Mode == add ->
        append(List, Solvs, NewList),
        format('-p (-p) has added solvables: -n -p-n-n',
        [KS, AgentName, PrettySolvs])
    | Mode == remove ->
        subtract(List, Solvs, NewList),
        format('-p (-p) has removed solvables: -n -p-n-n',
        [KS, AgentName, PrettySolvs])
    | Mode == replace ->
        memberchk(with(NewSolvable), EvParams),
        Solvs = [Solvable],
        oaa:replace_element(Solvable, List, NewSolvable, NewList),
        format('-p (-p) has replaced solvable:-n -p-nwith solvable:-n
-p-n-n',
        [KS, AgentName, Solvable, NewSolvable])
    ),
    oaa:oaa_add_data_local(agent_data(KS, Status, NewList, AgentName),
Params),

% if a parent exists (not root), pass goals upward.
(com:com_GetInfo(parent, connection(_C)) ->
    oaa_PostEvent( ev_register_solvables(Mode,Solvs,EvParams,KSName),
    [address(parent)])
    |
    true),
    !.

```



```

/* A client has requested that I declare certain solvables.
   TBD: This is still sketchy; should include some validation of the
   request, and should ensure the perms and params are right. */
oaa_AppDoEvent(ev_post_declare(Mode, Solvables, Params), EvParams) :-
    memberchk(from(RequestingKS), EvParams),
    oaa:ooo_declare_local(Mode, Solvables, Params, NewSolvables),
    icl_ConvertSolvables(PrettySolvs, NewSolvables),
    oaa_Id(MyId),
    oaa_Name(MyName),
    format('-p (-p) has added solvables: -n -p-n-n',
           [MyId, MyName, PrettySolvs]),
    oaa_PostEvent(
        ev_reply_declared(Mode, Solvables, Params, NewSolvables),
        [address(RequestingKS)]).

% A client requests a data solvable update operation (add, remove, replace)
% on the appropriate agents.
oaa_AppDoEvent(ev_post_update(Mode, Clause, Params), EvParams) :-
    ( Clause = (Head :- _Body) ->
      true
    | otherwise ->
      Head = Clause
    ),
    memberchk(from(RequestingKS), EvParams),
    % see if the query is addressed using address(KS) in Params
    check_address(Params, AddrKS),
    choose_agents_for_data(RequestingKS, Head, AddrKS, write, false, KSList),
    dispatch_update_request(RequestingKS, Mode, Clause, Params, KSList).

% A client requests a trigger update operation (Mode = add, remove, replace)
% on the appropriate agents. For triggers of type comm' and time', the
% address parameter must be present (otherwise, the request should not
% have come to the facilitator). For the other types, the address is
% optional.

oaa_AppDoEvent(ev_post_trigger_update(Mode, data, Condition,
                                       Action, Params), EvParams) :-
    !,
    memberchk(from(RequestingKS), EvParams),
    % see if the query is addressed using address(KS) in Params
    check_address(Params, AddrKS),
    choose_agents_for_data(RequestingKS, Condition, AddrKS, call, false, KSList),
    append(Params, EvParams, AllParams),
    dispatch_trigger_request(RequestingKS, Mode, data, Condition, Action,
                             AllParams, KSList).

oaa_AppDoEvent(ev_post_trigger_update(Mode, task, Condition,
                                       Action, Params), EvParams) :-
    !,
    memberchk(from(RequestingKS), EvParams),
    % see if the query is addressed using address(KS) in Params
    check_address(Params, AddrKS),
    choose_agents_for_goal(RequestingKS, Condition, AddrKS, Params, false, KSList),
    append(Params, EvParams, AllParams),
    dispatch_trigger_request(RequestingKS, Mode, task, Condition, Action,
                             AllParams, KSList).

oaa_AppDoEvent(ev_post_trigger_update(Mode, Type, Condition,

```

```

        Action, Params), EvParams) :-
memberchk(from(RequestingKS), EvParams),
check_address(Params, KSList),
is_list(KSList),
append(Params, EvParams, AllParams),
dispatch_trigger_request(RequestingKS, Mode, Type, Condition, Action,
        AllParams, KSList).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TBD: New for compound goals:

% If satisfaction of a compound goal is requested, and the compound query
% interpreter is not included, signal error condition:
oaa_AppDoEvent(ev_post_solve(Goal, Params), EvParams) :-
    \+ current_predicate(complete_goal(_,_,_,_)),
    \+ icl_BasicGoal(Goal),
    !,
    format('ERROR: This facilitator does not support compound goals-n', []),
    format(' Returning 0 solutions for goal:-n -w-n', [Goal]),
    oaa_Id(Facilitator),
    memberchk(from(RequestingKS), EvParams),
    oaa_PostEvent(
        ev_reply_solved([Facilitator], [], Goal, Params, []),
    [address(RequestingKS)]).

% If compound goal capabilities are included, ALL ev_post_solve events are
% handled here. Otherwise, they fall through to later clauses.
oaa_AppDoEvent(ev_post_solve(Goal, Params), EvParams) :-
    current_predicate(complete_goal(_,_,_,_)),
    !,
    memberchk(from(RequestingKS), EvParams),
    complete_goal(RequestingKS, Goal, Params, CompletedGoal),
    execute_goal(RequestingKS, Goal, Params, CompletedGoal).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

/* Finds all KSs for a goal, asks them to solve it, then returns */
/* the answers to the calling KS */
oaa_AppDoEvent(ev_post_solve(Goal, Params), EvParams) :-
    memberchk(from(RequestingKS), EvParams),
    % see if the query is addressed using address(KS) in Params
    check_address(Params, AddrKS),
    choose_agents_for_goal(RequestingKS, Goal, AddrKS, Params, true, KSList),

    % if none of my agents know how to solve goal, send to parent
    (KSList = [] ->

        find_level(Params, Level, NewParams),
        ((com:com_GetInfo(parent, fac_name(ParentName)),
        Level > 0) ->
            oaa_TraceMsg('-nRouting goal "ev_solve(-p)" to parent -p.-n',
                [Goal, ParentName]),

            new_goal_id(Id),
            oaa_PostEvent( ev_post_solve_from_bb(Id, Goal, NewParams),
                [address(parent)]),

```

```

% if answers requested,
% send parent's answers directly back to requestingKS
% as well as blackboard solutions
(memberchk(reply(none), NewParams) -> true |
% No longer valid:
% send_blackboard_solutions(RequestingKS, Goal, Params),
oaa:oaa_add_trigger_local(
    comm,
    event(ev_reply_solved_by_bb(Id,SomeKS,Goal,Params2,Solutions),
        _),
    ev_respond_query(Id,RequestingKS,SomeKS,Goal,Params,Params2,
        Solutions),
    [recurrence(when), on(receive)])
)
|
% root blackboard: doesn't know anyone who can solve goal
(memberchk(reply(none), NewParams) -> true |
    oaa_Id(KSID),
    oaa_PostEvent(
        ev_reply_solved([KSID], [],Goal,Params, []),
        [address(RequestingKS)])
    )
)
| otherwise ->
    dispatch_solve_request(RequestingKS, Goal, Params, EvParams, KSList)
).

/* Finds all KSs for a goal, asks them to solve it, then returns */
/* the answers to the calling BB */
oaa_AppDoEvent(ev_post_solve_from_bb(Id, Goal, Params), EvParams) :-
    memberchk(from(RequestingKS), EvParams),
    % see if the query is addressed using address(KS) in Params
    check_address(Params, AddrKS),
    choose_agents_for_goal(RequestingKS,Goal,AddrKS,Params,true,KSList),

    % if none of my agents know how to solve goal, send to parent
    (KSList = [] ->

        find_level(Params, Level, NewParams),
        % try to ask parent
        ((com:com_GetInfo(parent, fac_name(ParentName)),
            com:com_GetInfo(parent, fac_id(ParentId)), Level > 0) ->
            oaa_TraceMsg('-nRouting goal "ev_solve(-p)" to parent -p.-n',
                [Goal, ParentName])),

        oaa_PostEvent( ev_post_solve_from_bb(Id, Goal, NewParams),
            [address(parent)]),

        (memberchk(reply(none), NewParams) -> true |
            oaa:oaa_add_trigger_local(
                comm,
                event(ev_reply_solved_by_bb(Id, _SomeKS, Goal, P2, Solutions),
                    _),
                ev_respond_bb_query(RequestingKS,ParentId,Id,Goal,Params,
                    P2, Solutions),
            )
        )
    )
)

```

```

        [recurrence(when), on(receive)])
    )
    |
    % root blackboard : knows no solvers
    (memberchk(reply(none), Params) -> true |
      oaa_Name(KSName),
      oaa_PostEvent(
        ev_reply_solved_by_bb(Id, KSName, Goal, Params, []),
        [address(RequestingKS)])
      )
    )
  |
  member(SomeKS, KSList),      % backtrack over all KSs.
  oaa_TraceMsg('-nRouting goal to ~p: ~p~n',
    [SomeKS, Goal]),

  oaa_PostEvent( ev_solve(Id, Goal, Params),
    [address(SomeKS), from(RequestingKS)]),
  (memberchk(reply(none), Params) -> fail |
    oaa:oaa_add_trigger_local(
      comm,
      event(ev_solved(Id, _SomeKS, Goal, P2, Solutions), _),
      ev_respond_bb_or_post_higher(RequestingKS, SomeKS, Id,
        Goal, P2, Solutions),
      [recurrence(when), on(receive)])
    ),
  fail      % send events to all KSs that can solve goal.
  ).

oaa_AppDoEvent(wakeup(time_limit(Id)), _EvParams) :-
  retract(time_limit_trigger(Id, _When, RequestingKS, Goal, Params)),
  oaa_TraceMsg('-nTime limit expired. Goal failed:-n ~p~n', [Goal]),
  oaa_Id(KSId),      % get local ksId
  %   interpret(KSId,
  %   ev_respond_query(-1, RequestingKS, KSId, Goal, Params, Params, [])).
  oaa_Interpret(
    ev_respond_query(-1, RequestingKS, KSId, Goal, Params, Params, []),
    [from(KSId)]).

% When asked by parent blackboard to solve a goal,
% route all answers back using "ev_solved(Id, KS, Goal, Params, Solutions)".
oaa_AppDoEvent(ev_solve(Id, Goal, Params), EvParams) :-
  memberchk(from(ParentBB), EvParams),
  oaa_Name(KSName),

  % see if the query is addressed using address(KS) in Params
  check_address(Params, AddrKS),
  choose_agents_for_goal(KSName, Goal, AddrKS, Params, true, KSList),

  % if none of my agents know how to solve goal, send empty solutions
  (KSList = [] ->
    (memberchk(reply(none), Params) -> true |
      oaa_PostEvent( ev_solved(Id, KSName, Goal, Params, []),
        [address(ParentBB)])
    )
  )

```

```

)
|
member(SomeKS, KSList),    % backtrack over all KSs.
    oaa_TraceMsg('-nRouting goal "ev_solve(-p)" to -p.-n', [Goal,
SomeKS]),

    oaa_PostEvent( ev_solve(Id, Goal, Params),
                    [address(SomeKS), from(ParentBB)]),
(memberchk(reply(none), Params) -> fail |
    oaa:oaa_add_trigger_local(
        comm,
        event(ev_solved(Id, _SomeKS, Goal, P2, Solutions), _),
        ev_respond_to_parent(ParentBB,KSName,Id,Goal,Params,
                              P2, Solutions),
        [recurrence(when), on(receive)])
    ),
fail    % send events to all KSs that can solve goal.
).

/* If a KS is available, send it the message */
oaa_AppDoEvent(ev_post_event(Event), EvParams) :-
    memberchk(from(KS), EvParams),
    choose_ks_for_goal(KS, Event, _, [], SomeKS, _),
    oaa_PostEvent(Event, [address(SomeKS), from(KS)]),
    fail.

/* If a KS is available, send it the message */
oaa_AppDoEvent(ev_post_event(KSName, Event), EvParams) :-
    oaa_Name(KSName), !,
    % interpret(KS, Event).
    oaa_Interpret(Event, EvParams).
oaa_AppDoEvent(ev_post_event(KSName, Event), EvParams) :-
    memberchk(from(KS), EvParams),
    % agent must be "ready" to receive messages, or just
    % open if it is an agent compiled with old agentlib.
    (oaa:oaa_solve_local(agent_data(RealKS, ready, _Solvable,AgentName), []))
;
    oaa:oaa_solve_local(agent_data(RealKS, open, _Solvable,AgentName), []),
    oaa_Version(RealKS, _Language, Version),
    Version < 2.0),
    (match_ks(KSName, RealKS) ; KSName = AgentName),
    oaa_PostEvent(Event, [address(RealKS), from(KS)] ),
    fail.
% oaa_AppDoEvent(ev_post_event(_KS, _Event), _KS) :- !.
oaa_AppDoEvent(ev_post_event(_KS, _Event), _EvParams) :- !.

% Send back solutions to KS who originally requested them (with ev_post_solve)
%
% 970219: DLM: Added arg. OrigParams. There is now a requirement that
% the params returned in a ev_reply_solved event must be unifiable with the
original
% params (from the corresponding solve event).
oaa_AppDoEvent(ev_respond_query(Id,RequestingKS, Requestee, Goal, OrigParams,
                                Params,Solutions), _EvParams) :-
    oaa_TraceMsg('-nRouting answers back to -p:-n -p-n',

```

```

        [RequestingKS,Solutions]),
cancel_time_check(Id),
unify_params(OrigParams, Params, UParams),
( Solutions == [] ->
    Solvers = []
| otherwise ->
    Solvers = [Requestee]
),
oaa_PostEvent( ev_reply_solved([Requestee], Solvers, Goal, UParams,
Solutions),
                [address(RequestingKS)]), !.

% Send back solutions to KS who originally requested them (with ev_post_solve)
% If no solutions, ask a higher blackboard
oaa_AppDoEvent(
    ev_respond_or_post_higher(RequestingKS, Solver,Id,Goal,P,Solutions),
    _EvParams) :-
    ((Solutions \== [] ; oaa:oaa_class(root)) ->
        cancel_time_check(Id), !,
        return_solutions(RequestingKS, Solver, Id, Goal,P,Solutions)
    |
        % @@DLM: The following needs work. Must check goal_count status
        % before posting higher
        % sub-agents found no solutions: post higher
        com:com_GetInfo(parent, fac_id(ParentId)),
        find_level(P, Level, NewParams),
        Level > 0,
        oaa_PostEvent( ev_post_solve_from_bb(Id, Goal, NewParams),
            [address(parent)]),
        oaa:oaa_add_trigger_local(
            comm,
            event(ev_reply_solved_by_bb(Id, _SomeKS, Goal, P2, Solutions),
                _),
            ev_respond_query(Id,RequestingKS,ParentId,Goal,P,P2, Solutions),
            [recurrence(when), on(receive)])
    ).

% Send back acknowledgement to agent that originally requested an update.
oaa_AppDoEvent(
    ev_return_update(RequestingKS, Mode, Solver, Id, Clause, Params, Updaters),
    _EvParams) :-
    return_update(RequestingKS, Mode, Solver, Id, Clause, Params, Updaters).
% Send back acknowledgement to agent that originally requested a trigger
% update.
oaa_AppDoEvent(
    ev_return_trigger_update(RequestingKS, Mode, Solver, Id, Type, Condition,
        Action, Params, Updaters),
    _EvParams) :-
    oaa_TraceMsg('-nRouting trigger updaters back to -p:-n -p-n',
        [RequestingKS,Updaters]),
    return_trigger_update(RequestingKS, Mode, Solver, Id, Type, Condition,
        Action, Params, Updaters).

% Send back solutions to a blackboard who requested them
% (with ev_post_solve_from_bb)
%
```

```

% 970219: DLM: Added arg. OrigP. There is now a requirement that
% the params returned in a ev_solved event must be unifiable with the original
% params (from the corresponding solve event).
oaa_AppDoEvent(ev_respond_bb_query(RequestingBB, Solver, Id,Goal,
                                OrigP, P,Solutions), _EvParams) :-
    unify_params(OrigP, P, UP),
    oaa_TraceMsg('-nRouting answers back to blackboard ~p:~n  ~p~n',
                [RequestingBB,Solutions]),
    oaa_PostEvent( ev_reply_solved_by_bb(Id,Solver,Goal,UP,Solutions),
                  [address(RequestingBB)]), !.

% Send back solutions to a blackboard who requested them
oaa_AppDoEvent(
    ev_respond_bb_or_post_higher(RequestingBB,Solver,Id,Goal,P,Solutions),
    _EvParams) :-
    ((Solutions \== [] ; oaa:oaa_class(root)) ->
        oaa_TraceMsg('-nRouting answers back to blackboard ~p:~n  ~p~n',
                    [RequestingBB,Solutions]),
        oaa_PostEvent( ev_reply_solved_by_bb(Id, Solver, Goal, P,Solutions),
                      [address(RequestingBB)]))
    |
    % sub-agents found no solutions: post higher
    com:com_GetInfo(parent, fac_id(ParentId)),
    find_level(P, Level, NewParams),
    Level > 0,
    oaa_PostEvent( ev_post_solve_from_bb(Id, Goal, NewParams),
                  [address(parent)]),
    oaa:oaa_add_trigger_local(
        comm,
        event(ev_reply_solved_by_bb(Id, _SomeKS, Goal, P2, Solutions),
            _),
        ev_respond_bb_query(RequestingBB,ParentId,Id,Goal,P,P2,Solutions),
        [recurrence(when), on(receive)])
    ).

% Send back solutions to KS who originally requested them (with ev_post_solve)
%
% 970219: DLM: Added arg. OrigP. There is now a requirement that
% the params returned in a ev_solved event must be unifiable with the original
% params (from the corresponding solve event).
oaa_AppDoEvent(ev_respond_to_parent(ParentBB,Solver,Id,Goal, OrigP,
                                    P, Solutions), _EvParams) :-
    unify_params(OrigP, P, UP),
    oaa_TraceMsg('-nRouting answers back to parent bb ~p:~n  ~p~n',
                [ParentBB,Solutions]),
    oaa_PostEvent( ev_solved(Id, Solver, Goal, UP, Solutions),
                  [address(ParentBB)]), !.

oaa_AppDoEvent(ev_check_agent_name(KSName), EvParams):-
    memberchk(from(KS), EvParams),
    findall(KSName, oaa:oaa_solve_local(agent_location(_KSID, KSName ,_,_)),
    []), L),
    (L==[] ->
        % @@tcp_send shouldn't be used:
        tcp_send(KS, 'UNIQUE');
        findall(KS1, oaa:oaa_solve_local(agent_location(_, KS1, _,_), [], R),

```

```

tcp_send(KS, R)),!.

oaa_AppDoEvent(ev_register_port_number(Name,Address), EvParams) :- %+KS, +Port,
+Host
    memberchk(from(KS), EvParams),
    Address =.. [address, Port, Host],
    oaa:oaa_remove_data_local(agent_location(KS, _Name, _Port, _Host),
[]),!,
    oaa:oaa_add_data_local(agent_location(KS, Name, Port, Host), []),
    format('Agent -p has Port: -p , Host: -p -n', [KS, Port, Host]),
    !.

oaa_AppDoEvent(ev_register_port_number(Name,Address), EvParams) :- %+KS, +Port,
+Host
    memberchk(from(KS), EvParams),
    Address =.. [address, Port, Host],
    oaa:oaa_add_data_local(agent_location(KS, Name, Port, Host), []),
    format('Agent -p has Port: -p , Host: -p -n', [KS, Port, Host]),
    !.

oaa_AppDoEvent(ev_continue_execution(Id, RKS, Requestees, Solvers, Solutions),
    _EvParams) :-
    continue_execution(Id, RKS, Requestees, Solvers, Solutions).

% This is called from a trigger set in compound.pl.
oaa_AppDoEvent(
    ev_unify_and_continue_execution(Id, RKS, Goal, Vars, Requestee, Requestees,
Solvers, Solutions),
    _) :-
    unify_and_continue_execution(Id, RKS, Goal, Vars, Requestee, Requestees,
Solvers, Solutions).

/* Facilitator solvable: report the version and language of some
connected agent. */
oaa_AppDoEvent(agent_version(Id, Language, Version), _EvParams) :-
    !,
    oaa_Version(Id, Language, Version).

/* Facilitator solvable: Find all agents who can solve goal */
oaa_AppDoEvent(can_solve(Goal, KSList), EvParams) :-
    ( memberchk(from(KS), EvParams) -> true | oaa_Id(KS) ),
    findall(SomeKS, choose_ks_for_goal(KS, Goal, _, [], SomeKS, _), KSList).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% choose_agents_for_goal(RequestingKS,Goal,AddrKS,Params,Sort,Agents) .
%
% The first 4 arguments are exactly as expected by choose_ks_for_goal.
% Sort, a boolean, tells whether to sort on utility.
choose_agents_for_goal(RequestingKS,Goal,AddrKS,Params,Sort,Agents) :-
    findall(
        p(Agent,Utility),
        choose_ks_for_goal(RequestingKS,Goal,AddrKS,Params,Agent,Utility),
        Pairs
    ),
    ( Sort ->
        samsort(oaa_utility_compare, Pairs, SortedPairs)
    | otherwise ->

```



```

        SortedPairs = Pairs
    ),
    findall(Agent, member(p(Agent,_Utility), SortedPairs), Agents).

% choose_agents_for_data(RequestingKS,Goal,AddrKS,Perm,Sort,Agents).
%
% The first 4 arguments are exactly as expected by choose_ks_for_data.
% Sort, a boolean, tells whether to sort on utility.
choose_agents_for_data(RequestingKS,Goal,AddrKS,Perm,Sort,Agents) :-
    findall(
        p(Agent,Utility),
        choose_ks_for_data(RequestingKS,Goal,AddrKS,Perm,Agent,Utility),
        Pairs
    ),
    ( Sort ->
        samsort(oaa_utility_compare, Pairs, SortedPairs)
    | otherwise ->
        SortedPairs = Pairs
    ),
    findall(Agent, member(p(Agent,_Utility), SortedPairs), Agents).

oaa_utility_compare(p(_Agent1,Utility1), p(_Agent2,Utility2)) :-
    Utility1 >= Utility2.

/* Finds a KS that knows how to solve Goal */

% backtracks over all KSs that know how to solve
% a particular goal, except for RequestingKS, which is the
% KS who asked for the goal to be solved in the
% first place. (RequestingKS is included if the 'reflexive' Param
% is present.)
% MemberList can be a list used to reduce the set to at most MemberList
% or can be a specific KS to try, or a variable.
% If an address is specified in MemberList, it can be the same as
% RequestingKS (DLM, 96/10/30).
% Solvable lists can contain complex tests (AC, 97/2/5)
% e.g. {goal1(Y), (g(X) :- X > 1, X < 10), goal2}
% Params is now used to check for 'reflexive' (DLM, 97/03/06).
% Utility is the numeric value the KS has associated with the
% solvable.
choose_ks_for_goal(RequestingKS, Goal, MemberList, Params, SomeKS, Utility) :-
    var(MemberList),
    !,
    ks_ready(SomeKS, ListOfGoals),
    ( icl_GetParamValue(reflexive(true), Params) ->
        true
    | otherwise ->
        SomeKS \== RequestingKS
    ),
    oaa:oaa_goal_matches_solvable(Goal, ListOfGoals, _, Matched),
    Matched = solvable(_, SolveParams, _),
    icl_GetParamValue(utility(Utility), SolveParams).
choose_ks_for_goal(_RequestingKS, Goal, MemberList, _Params, SomeKS, Utility) :-
    (is_list(MemberList) ->
        member(SomeKS, MemberList)
    | SomeKS = MemberList),

```

```

    oaa:icl_true_id(SomeKS, TrueId),
    ks_ready(TrueId, ListOfGoals),
    oaa:oaa_goal_matches_solvable(Goal, ListOfGoals, _, Matched),
    Matched = solvable(_, SolveParams, _),
    icl_GetParamValue(utility(Utility), SolveParams).

% backtracks over all KSs that know how to write a particular goal (or
% read, though that's not currently used), except for RequestingKS,
% which is the KS who asked for the goal to be solved in the first
% place. RequestingKS is never included, because he does the
% appropriate asserts locally, when appropriate.
%
% Perm is 'read' or 'write'.

choose_ks_for_data(RequestingKS, Goal, MemberList, Perm, SomeKS, Utility) :-
    var(MemberList),
    !,
    ks_ready(SomeKS, ListOfGoals),
    SomeKS \== RequestingKS,
    oaa:oaa_data_matches_solvable(Goal, ListOfGoals, Perm, _, Matched),
    Matched = solvable(_, SolveParams, _),
    icl_GetParamValue(utility(Utility), SolveParams).
choose_ks_for_data(_RequestingKS, Goal, MemberList, Perm, SomeKS, Utility) :-
    (is_list(MemberList) ->
        member(SomeKS, MemberList)
    | SomeKS = MemberList),
    ks_ready(SomeKS, ListOfGoals),
    oaa:oaa_data_matches_solvable(Goal, ListOfGoals, Perm, _, Matched),
    Matched = solvable(_, SolveParams, _),
    icl_GetParamValue(utility(Utility), SolveParams).

% ks_ready(*SomeKS, *ListOfGoals).
% Backtracks over all agents that are ready to solve goals.
% If SomeKS is bound (with an agent's local ID), only that agent is
% considered.
ks_ready(SomeKS, ListOfGoals) :-
    % agent must be "ready" to receive messages, or just
    % open if it is an agent compiled with old agentlib.
    (oaa:oaa_solve_local(agent_data(SomeKS, ready, ListOfGoals, _AgentName),
[])) ;
    oaa:oaa_solve_local(agent_data(SomeKS, open, ListOfGoals, _AgentName),
[]),
    oaa_Version(SomeKS, _Language, Version),
    Version < 2.0).
% Facilitator agents look up their own solvables in oaa_solvable/1.
ks_ready(SomeKS, ListOfGoals) :-
    oaa_Id(SomeKS),
    oaa:oaa_solvable(ListOfGoals).

match_ks(all, _KS).
match_ks(KS, KS).

% If params contains a VALID address (symbolic name or id) for one or more
% agents, return the agents' ids.
% If params contains an INVALID address, remove it from the list returned.
% Otherwise, KSAddr should return a variable.
% 97-05-23 (DLM): The address param now should always contain a list,

```

```

% but we'll check just to be safe.

check_address(Params, KSAddr) :-
    memberchk(address(Addr), Params),
    ( is_list(Addr) ->
        AddrList = Addr
      | AddrList = [Addr]),
    find_addresses(AddrList, KSAddr),
    !.
check_address(_Params, _SomeKS).

find_addresses([], []).
find_addresses([Addr | Addrs], [Id | Ids]) :-
    find_address(Addr, Id),
    !,
    find_addresses(Addrs, Ids).
find_addresses([_Addr | Addrs], Ids) :-
    find_addresses(Addrs, Ids).

% Given an agent id (eg. 5) or a symbolic name (eg. 'interface')
% returns the local id for the reference.
%
% TBD: This does not yet handle remote addresses (associated with a different
% facilitator).

find_address(addr(Addr), SomeKS) :-
    com:com_GetInfo(incoming, oaa_addr(Addr)),
    % That's me, the facilitator.
    !,
    oaa_Id(SomeKS).
find_address(addr(Addr, SomeKS), SomeKS) :-
    com:com_GetInfo(incoming, oaa_addr(Addr)),
    % One of my clients.
    !,
    % Make sure it's current:
    oaa:oaa_solve_local(agent_data(SomeKS, _, _ListOfGoals, _AgentName), []).
find_address(name(Name), SomeKS) :-
    !,
    atom(Name),
    oaa:oaa_solve_local(agent_data(SomeKS, _, _ListOfGoals, Name), []).
find_address(SomeKS, SomeKS) :-
    oaa:oaa_solve_local(agent_data(SomeKS, _, _ListOfGoals, _AgentName), []),
    !.

find_level(Params, Level, NewParams) :-
    oaa:remove_element(level_limit(Level), Params, Params2), !,
    (Level > 0 ->
        NewLevel is Level - 1
      | NewLevel is 0),
    NewParams = [level_limit(NewLevel) | Params2].
find_level(Params, 1, Params).

post_to_all_clients(Event) :-
    oaa_Id(FacId),
    oaa:oaa_solve_local(agent_data(ClientId, ready, _Solvable, _AgentName),
    []),

```

```

        ClientId \== FacId,
        oaa_PostEvent(Event, [address(ClientId), from(FacId)] ),
        fail.
post_to_all_clients(_Event).

% This is called when length of KSList is > 0.
%
% goal_count(GoalId,Goal,Params,EvParams,ToBeCalled,Called,
%           Responders,Solvers,Answers,NumAnswers)

dispatch_solve_request(RequestingKS, Goal, Params, EvParams, KSList) :-
    new_goal_id(Id),
    % Note that reply (none) overrides parallel_ok (false). We can't
    % provide parallel_ok (false) if no replies come back from solvers.
    ( memberchk(reply(none), Params) ->
      dispatch_solve_events(KSList, Id, RequestingKS, Goal, Params, EvParams)
    | memberchk(parallel_ok(false), Params) ->
      % Dispatch to one KS; save the rest for later.
      KSList = [FirstKS | Rest],
      assert(goal_count(Id, Goal, Params, EvParams, Rest,
                       [FirstKS], [], [], [], 0)),
      dispatch_solve_event(Id, RequestingKS, Goal, Params, EvParams, FirstKS)
    | otherwise ->
      % Dispatch to all KSs.
      assert(goal_count(Id, Goal, Params, EvParams, [],
                       KSList, [], [], [], 0)),
      dispatch_solve_events(KSList, Id, RequestingKS, Goal, Params, EvParams)
    ).

dispatch_solve_events([], _Id, _RequestingKS, _Goal, _Params, _EvParams).
dispatch_solve_events([SomeKS | Rest], Id, RequestingKS, Goal,
                      Params, EvParams) :-
    dispatch_solve_event(Id, RequestingKS, Goal, Params, EvParams, SomeKS),
    dispatch_solve_events(Rest, Id, RequestingKS, Goal, Params, EvParams).

dispatch_solve_event(Id, RequestingKS, Goal, Params, EvParams, SomeKS) :-
    oaa_Id(SomeKS),
    % That's me, the facilitator.
    !,
    icl_GoalComponents(Goal, _, _, GoalParams),
    append(Params, EvParams, InheritedParams),
    append(GoalParams, InheritedParams, AllParams),
    findall(Goal,
            % InheritedParams here is right, not AllParams:
            oaa:oa_solve_local(Goal, InheritedParams,
                               Solutions),
            ( memberchk(reply(none), AllParams) ->
              true
            | otherwise ->
              oaa_AppDoEvent(

ev_respond_or_post_higher(RequestingKS, SomeKS, Id, Goal, Params, Solutions),
                          [])
    ).
dispatch_solve_event(Id, RequestingKS, Goal, Params, _EvParams, SomeKS) :-
    oaa_TraceMsg('-nRouting goal "ev_solve(-p)" to -p.-n', [Goal, SomeKS]),

```

```

% ask a sub-agent to try and solve goal.
% if solutions are returned, pass them to requestingKS.
% otherwise, ask higher blackboard to try and solve goals.
% note: send ev_solve(id(Id,SomeKS), ...) as a means of insuring
% that each ev_solved() trigger is unique and only matches
% exactly one response. We use _SomeKS in the field indicating
% which agent actually solved the goal because individual
% agents don't necessarily know their internal unique ID #.
oaa_PostEvent( ev_solve(id(Id,SomeKS), Goal, Params),
               [address(SomeKS), from(RequestingKS)]),
( memberchk(reply(none), Params) ->
  true
| otherwise ->
  % If time_limit specified in parameters, setup
  % time_trigger to wakeup if solutions hasn't been returned
  % in specified time.
  ( memberchk(time_limit(NSecs), Params) ->
    add_time_check(NSecs, Id, RequestingKS, Goal,Params)
  | true),
  oaa:oaa_add_trigger_local(
    comm,
    event(ev_solved(id(Id,SomeKS), _SomeKS, Goal, P2, Solutions), _),
    ev_respond_or_post_higher(RequestingKS,SomeKS,Id,Goal,P2,Solutions),
    [recurrence(when), on(receive)])
).

% return_solutions(+RequestingKS, +Responder, +Id, +Goal, +P, +NewSolutions).
% Having just received solutions from a Responder, take the appropriate action.
%
% Even though the Responder has returned copies of the goal and params,
% we don't need them because we have a local copy in goal_count.
%
% @@DLM: Unresolved question about streaming: Should we stream the
% responses with 0 solutions? [My thinking is "yes".]
return_solutions(RequestingKS, Responder, Id, _Goal, _P, NewSolutions) :-
  % ToBeCalled lists solvers not yet called.  PrevCalled lists
  % the called solvers that have yet to respond.
  retract(goal_count(Id, Goal, Params, EvParams,
                    ToBeCalled, PrevCalled, PrevResponders,
                    PrevSolvers, PrevSolutions, PrevNumSol)),
  !,
  % Take Responder out of the called list:
  ( selectchk(Responder, PrevCalled, Called) ->
    true
  | otherwise ->
    format('ERROR: Inappropriate ev_solved event received:~n', []),
    format(' ~w ~w ~w ~w~n', [RequestingKS, Responder, Id, Goal]),
    Called = PrevCalled
  ),
  % and put him into the responder list:
  append(PrevResponders, [Responder], Responders),
  % The solvers are just the responders that succeeded:
  ( NewSolutions = [] ->
    NewSolvers = []
  | otherwise ->
    NewSolvers = [Responder]
  ),

```

```

append(PrevSolvers, NewSolvers, Solvers),
append(PrevSolutions, NewSolutions, Solutions),
length(NewSolutions, NewNumSol),
NumSol is PrevNumSol + NewNumSol,

% This case means that either: (1) we've gotten responses from all
% solvers; and/or (2) we have reached the desired number of solutions.
% By not saving goal_count, we ensure that any additional returned
% solutions are ignored:
( ((ToBeCalled == [], Called == []) ;
  (memberchk(solution_limit(Limit), Params), NumSol >= Limit)) ->
  % This test is a place-holder; streaming not yet official:
  ( memberchk(reply(streaming), Params) ->
    Return = ev_reply_solved([Responder], NewSolvers, Goal, Params,
                             NewSolutions)
  | otherwise ->
    Return = ev_reply_solved(Responders, Solvers, Goal, Params,
                             Solutions)
  ),
  Save = false
% This case happens with parallel_ok(false):
| ToBeCalled = [Next | Rest] ->
  dispatch_solve_event(Id, RequestingKS, Goal, Params, EvParams, Next),
  % This test is a place-holder; streaming not yet official:
  ( memberchk(reply(streaming), Params) ->
    Return = ev_reply_solved([Responder], NewSolvers, Goal, Params,
                             NewSolutions),
    Save = goal_count(Id, Goal, Params, EvParams,
                      Rest, [Next|Called], [], [], [], NumSol)
  | otherwise ->
    Return = false,
    Save = goal_count(Id, Goal, Params, EvParams,
                      Rest, [Next|Called], Responders, Solvers,
                      Solutions, NumSol)
  )
% Still waiting for some called solvers to respond:
| Called = [_ | _] ->
  % This test is a place-holder; streaming not yet official:
  ( memberchk(reply(streaming), Params) ->
    Return = ev_reply_solved([Responder], NewSolvers, Goal, Params,
                             NewSolutions),
    Save = goal_count(Id, Goal, Params, EvParams,
                      ToBeCalled, Called, [], [], [], NumSol)
  | otherwise ->
    Return = false,
    Save = goal_count(Id, Goal, Params, EvParams,
                      ToBeCalled, Called, Responders, Solvers,
                      Solutions, NumSol)
  )
),
( Save == false ->
  true
| otherwise ->
  assert(Save)
),
( Return == false ->
  true

```

```

| otherwise ->
    oaa_TraceMsg('-nRouting answers back to -p:-n  -p-n',
                [RequestingKS,Return]),
    oaa_PostEvent(Return, [address(RequestingKS)])
).
return_solutions(_RequestingKS, _Responder, _Id, _Goal, _P, _NewSolutions).

dispatch_update_request(RequestingKS, Mode, Clause, Params, []) :-
    % No agents able to perform the requested update:
    !,
    ( memberchk(reply(none), Params) ->
      true
    | otherwise ->
      Event = ev_reply_updated(Mode, Clause, Params, [], []),
      oaa_PostEvent(Event, [address(RequestingKS)])
    ).

dispatch_update_request(RequestingKS, Mode, Clause, Params, KSList) :-
    new_goal_id(Id),
    length(KSList, NumKSsForGoal),
    % if more than one KS can solve the goal, remember so that
    % we can collect answers from all of them later
    ( NumKSsForGoal > 1 ->
      assert(update_count(Id, NumKSsForGoal, [], []))
    | otherwise ->
      true
    ),
    member(SomeKS, KSList), % backtrack over all KSs.
    dispatch_update_event(Id, RequestingKS, Mode, Clause, Params, SomeKS),
    fail.

dispatch_update_request(_RequestingKS, _Mode, _Clause, _Params, _KSList).

dispatch_update_event(Id, RequestingKS, Mode, Clause, Params, SomeKS) :-
    oaa_Id(SomeKS),
    % That's me, the facilitator.
    !,
    ( Mode == add ->
      Functor = oaa_add_data_local
    | Mode == replace ->
      Functor = oaa_replace_data_local
    | otherwise ->
      Functor = oaa_remove_data_local
    ),
    append(Params, [from(RequestingKS)], AllParams),
    Goal =.. [Functor, Clause, AllParams],
    ( call(oaa:Goal) ->
      Updaters = [SomeKS]
    | otherwise ->
      Updaters = []
    ),
    ( memberchk(reply(none), Params) ->
      true
    | otherwise ->
      % Params must be returned here (not AllParams):
      return_update(RequestingKS, Mode, SomeKS, Id, Clause, Params, Updaters)
    ).

dispatch_update_event(Id, RequestingKS, Mode, Clause, Params, SomeKS) :-
    oaa_TraceMsg('-nRouting request "ev_update(-p, -p, -p)" to -p.-n',

```

```

        [Mode, Clause, Params, SomeKS]),
append(Params, [from(RequestingKS)], AllParams),
oaa_PostEvent(
    ev_update(id(Id,SomeKS), Mode, Clause, AllParams),
    [address(SomeKS)]),
( memberchk(reply(none), Params) ->
  true
| otherwise ->
  % TBD: Do we want to set a time trigger here?
  oaa:oaa_add_trigger_local(
    comm,
    event(ev_updated(id(Id,SomeKS), _Mode, _Clause, _P2, Updaters), _),
    % Params must be returned here (not AllParams):
    ev_return_update(RequestingKS,Mode,SomeKS,Id,
                      Clause,Params,Updaters),
    [recurrence(when), on(receive)])
).

```

```

% Returns, to requesting KS, the addresses of all agents (including
% facilitator if appropriate), that attempted (NewKSs) and that actually
% satisfied (Updaters) an update request.
%
% NewUpdaters is always either [], or a singleton list.
%
% Possible values for Mode: add, remove, replace.
%
% Note: Params must be returned in ev_reply_updated, so it must be
% unifiable with the params embedded in the requesting event (ev_post_event).

```

```

return_update(RequestingKS, Mode, Responder, Id, Clause, Params,
              NewUpdaters) :-
  retract(update_count(Id, AgentsLeft, PrevKSs, PrevUpdaters)),
  append(PrevUpdaters, NewUpdaters, Updaters),
  append(PrevKSs, [Responder], NewKSs),
  ( AgentsLeft > 1 ->
    NewAgentsLeft is AgentsLeft - 1,
    assert(update_count(Id, NewAgentsLeft, NewKSs, Updaters))
  | otherwise ->
    oaa_TraceMsg('-nRouting updaters back to -p:-n -p-n',
                 [RequestingKS,Updaters]),
    Event = ev_reply_updated(Mode, Clause, Params, NewKSs, Updaters),
    oaa_PostEvent(Event, [address(RequestingKS)])
  ), !.

```

```

return_update(RequestingKS, Mode, Responder, _Id, Clause, Params, Updaters) :-
  oaa_TraceMsg('-nRouting updaters back to -p:-n -p-n',
               [RequestingKS,Updaters]),
  Event = ev_reply_updated(Mode, Clause, Params, [Responder], Updaters),
  oaa_PostEvent(Event, [address(RequestingKS)]).

```

```

% No agents able to install this trigger:
dispatch_trigger_request(RKS, Mode, Type, Condition, Action, Params, []) :-
  !,
  ( memberchk(reply(none), Params) ->
    true
  | otherwise ->
    Event = ev_reply_trigger_updated(Mode, Type, Condition, Action, Params,

```



```

        [], []),
        oaa_PostEvent(Event, [address(RKS)])
    ).
dispatch_trigger_request(RKS, Mode, Type, Condition, Action, Params, KSLList) :-
    new_goal_id(Id),
    length(KSLList, NumKSsForGoal),
    % if more than one KS can solve the goal, remember so that
    % we can collect answers from all of them later
    ( NumKSsForGoal > 1 ->
        assert(update_count(Id, NumKSsForGoal, [], []))
    | otherwise ->
        true
    ),
    member(SomeKS, KSLList), % backtrack over all KSs.
    dispatch_trigger_event(Id, RKS, Mode, Type, Condition, Action, Params,
SomeKS),
    fail.
dispatch_trigger_request(_RKS, _Mode, _Type, _Condition, _Action, _Params,
_KSLList).

dispatch_trigger_event(Id, RKS, Mode, Type, Condition, Action, Params,
SomeKS) :-
    oaa_Id(SomeKS),
    % That's me, the facilitator.
    !,
    ( Mode == add ->
        Functor = oaa_add_trigger_local
    | otherwise ->
        Functor = oaa_remove_trigger_local
    ),
    Goal =.. [Functor, Type, Condition, Action, Params],
    ( call(oaa:Goal) ->
        Updaters = [SomeKS]
    | otherwise ->
        Updaters = []
    ),
    ( memberchk(reply(none), Params) ->
        true
    | otherwise ->
        return_trigger_update(RKS, Mode, SomeKS, Id, Type,
Condition, Action, Params, Updaters)
    ).
dispatch_trigger_event(Id, RKS, Mode, Type, Condition, Action, Params,
SomeKS) :-
    oaa_TraceMsg('-nRouting request-n ev_update_trigger(-p, -p, -p, -p, -p)-nto
-p.-n',
        [Mode, Type, Condition, Action, Params, SomeKS]),
    oaa_PostEvent(
        ev_update_trigger(id(Id, SomeKS), Mode, Type, Condition, Action, Params),
        [address(SomeKS), from(RKS)]),
    ( memberchk(reply(none), Params) ->
        true
    | otherwise ->
        % TBD: Do we want to set a time trigger here?
        oaa:oaa_add_trigger_local(
            comm,

```

```

        event(ev_trigger_updated(id(Id,SomeKS), _Mode, _Type, _Condition,
    _Action, P2, Updaters), _),
        ev_return_trigger_update(RKS,Mode,SomeKS,Id,
                                Type,Condition,Action,P2,Updaters),
        [recurrence(when), on(receive)])
    ).

% Returns, to requesting KS, the addresses of all agents (including
% facilitator if appropriate), that attempted (NewKSs) and that actually
% satisfied (Updaters) a trigger update request.
%
% NewUpdaters is always either [], or a singleton list.
%
% Possible values for Mode: add, remove.

return_trigger_update(RequestingKS, Mode, Responder, Id,
                      Type, Condition, Action, Params, NewUpdaters) :-
    retract(update_count(Id, AgentsLeft, PrevKSs, PrevUpdaters)),
    append(PrevUpdaters, NewUpdaters, Updaters),
    append(PrevKSs, [Responder], NewKSs),
    ( AgentsLeft > 1 ->
      NewAgentsLeft is AgentsLeft - 1,
      assert(update_count(Id, NewAgentsLeft, NewKSs, Updaters))
    | otherwise ->
      Event = ev_reply_trigger_updated(Mode,Type,Condition,Action,
                                       Params, NewKSs, Updaters),
      oaa_PostEvent(Event, [address(RequestingKS)])
    ), !.

return_trigger_update(RequestingKS, Mode, Responder, _Id,
                      Type, Condition, Action, Params, Updaters) :-
    Event = ev_reply_trigger_updated(Mode,Type,Condition,Action,
                                     Params, [Responder], Updaters),
    oaa_PostEvent(Event, [address(RequestingKS)]).

% unify_params(+OrigParams, +Params, -UnifiedParams).
%
% There is now (970219) a requirement that the params returned in
% a ev_solved or ev_solved_by_bb event must be unifiable with the original
% params from the corresponding solve request. In some situations*, the
% Params returned to the facilitator by a solver may not unify with
% the OrigParams, but may contain individual elements with variables
% instantiated by the solver. This pred. can be used to save these
% instantiations.
%
% *Such as, when find_level has been used to create a new params list.
unify_params([], _Params, []).
unify_params([OrigParam | Rest], Params, [OrigParam | UnifiedRest]) :-
    ( memberchk(OrigParam, Params) | true ),
    !,
    unify_params(Rest, Params, UnifiedRest).

*****

% These are extremely simple predicates for maintaining com_connection_info/5,
% which keeps info about the agents to which this agent currently has
% a communications channel.

```

```

add_connected(Id, Connection) :-
    assert(com:com_connection_info(Id, unknown, child,
        [connection(Connection), oaa_id(Id)], connected)).

update_connected(Id, AddInfo) :-
    com_AddInfo(Id, AddInfo).

% remove_connected(+Id).
remove_connected(Id) :-
    retractall(com:com_connection_info(Id, _, _, _, _)).

% if the time_limit(NSec) parameter is sent, install wakeup on server
% to indicate the request has failed if not achieved in the correct time.
add_time_check(NSecs, Id, RequestingKS, Goal, Params) :-
    (time_limit_trigger(Id, _When, RequestingKS, _Goal, _Params) ->
        true % already added for this goal request
    |
        tcp_now(Now),
        tcp_time_plus(Now, NSecs, Soon),
        tcp_schedule_wakeup(Soon, time_limit(Id)),
        assert(time_limit_trigger(Id, Soon, RequestingKS, Goal, Params)),
        oaa_TraceMsg('-nTime limit check added for ~p~n', [Goal])
    ), !.

% if solutions are returned before a time_limit_trigger has expired,
% remove the trigger.
cancel_time_check(Id) :-
    retract(time_limit_trigger(Id, _When, _RequestingKS, Goal, _Params)),
    tcp_cancel_wakeup(When, time_limit(Id)),
    oaa_TraceMsg('-nTime limit check removed because solution returned.~n
~p~n',
        [Goal]), !.
cancel_time_check(_Id).

/* Generates a unique ID for a goal.          */
/* ID's should be unique across blackboards*/
/* which is why we use the KSName prefix */
/* Goal counters are used to make sure the */
/* solution really matches the query.      */

new_goal_id(NewId) :-
    oaa_Name(KSName),
    concat(KSName, '_', Tmp),
    gensym(Tmp, NewId).

% Returns a list containing Num new goal ids.

new_goal_ids(Num, [NewId | RestIds]) :-
    Num > 0,
    !,
    new_goal_id(NewId),
    NewNum is Num - 1,
    new_goal_ids(NewNum, RestIds).
new_goal_ids(_Num, []).

```

```

start :-
    runtime_entry(start).

runtime_entry(start) :-
    initial_solvable(Solvable),
    com_ListenAt(incoming, CInfo),
    format('Listening at ~p~n~n', [CInfo]),
    oaa_RegisterCallback(app_do_event, user:oaa_AppDoEvent),
    oaa_Register(incoming, 'root', Solvable),
    on_exception(_, oaa_AppInit, true),
    oaa_MainLoop(true).

runtime_entry(abort) :- !.
%     format('Closing all connections...~n', []),
%     close_all_connections.

% If the Facilitator is killed (ctrl-c) before disconnecting
% all clients, it will not free the port.
% This code is an attempt to fix this problem, but it doesn't
% help. Why not???
% close_all_connections :-
%     tcp_connected(X,Y),
%     tcp_destroy_listener(Y),
%     tcp_shutdown(X),
%     fail.
% close_all_connections :-
%     tcp_reset, fail.

```

APPENDIX A.III

Source code file named libcom_tcp.pl.



```

%*****
% File      : libcom_tcp.pl
% Primary Authors : Adam Cheyer, David Martin
% Purpose   : TCP instantiation of lowlevel communication primitives for OAA
% Updated   : 01/98
%
% -----
% Unpublished-rights reserved under the copyright laws of the United States.
%
% Unpublished Copyright (c) 1993-98, SRI International.
% "Open Agent Architecture" and "OAA" are Trademarks of SRI International.
% -----
%
%

```

```

%*****
%* RCS Header and internal version
%*****

```

```

:- module(com,
    [com_Connect/2,
      com_Disconnect/1,
      com_ListenAt/2,
      com_SendData/2,
      com_SelectEvent/2,
      com_AddInfo/2,
      com_GetInfo/2]).

% rcs version number
rcsid(libcom_tcp, '$Header:
/tmp_mnt/home/zuma1/martin/OAA/agents/beta/prolog/RCS/com_tcp.pl,v 1.10
1998/05/06 22:35:36 martin Exp $').

:- use_module(library(sets)).
:- use_module(library(tcp)).
:- use_module(library(basics)).
:- use_module(library(lists)).
:- use_module(library(charsio)). % for sprintf and with_output_to_chars
:- use_module(library(ask)). % for ask_oneof
:- use_module(library(environ)). % read environment vars
:- use_module(library(files)). % can_open_file
:- use_module(library(strings)). % for concat

:- dynamic
    com_connection_info/5, % id, comtype, client/server, commInfo, status
    com_already_loaded/1. % filename

```

```

%*****
% name:      com_Connect(+ConnectionId, ?Address)
% purpose:   Given a connection ID and an address, initiates a client connection
% remarks:

```

```

% - if Address is a variable, instantiates the Address by using
%   com_ResolveVariables, which looks in a setup file, command line, and
%   environment variables for the required info.
% - stores the connection info for connection ID in com_connection_info/5.
% - fails if connection can't be made
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
com_Connect(ConnectionId, tcp(Host,Port)) :-
    ground(ConnectionId),
    % if variable address, look it up...
    ((var(Host) ; var(Port)) ->
        com_ResolveVariables([
            [cmd('-oaa_host',Host), cmd('-oaa_port', Port)],
            [env('OAA_HOST', Host), env_int('OAA_PORT', Port)],
            [setup('setup.pl', oaa_host, Host),
             setup('setup.pl',oaa_port, Port)]
        ]
    )
    | true),

    tcp_connect(address(Port, Host), RootConnection),
    assert(com_connection_info(ConnectionId, tcp, client,
        [addr(tcp(Host,Port)),
         oaa_host(Host),oaa_port(Port),connection(RootConnection)],
        connected)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      com_Disconnect(+ConnectionId)
% purpose:  Given a connection ID of type 'client', shuts down the connection.
% remarks:  Succeeds silently if there is not an open connection having the
%           given id.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
com_Disconnect(ConnectionId) :-
    ground(ConnectionId),
    com_connection_info(ConnectionId, tcp, client, _Info, connected),
    com_GetInfo(ConnectionId, connection(Connection)),
    tcp_shutdown(Connection),
    retract(com_connection_info(ConnectionId,tcp,client,_Info,connected)),
    !.
com_Disconnect(_ConnectionId).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      com_ListenAt(+ConnectionId, ?Address)
% purpose:  Given a connection ID and an address, initiate a server connection
% remarks:
% - if Address is a variable, instantiates the Address by using
%   com_ResolveVariables, which looks in a setup file, command line, and
%   environment variables for the required info.
% - stores the connection info for connection ID in com_connection_info/5.
% - fails if connection can't be made
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
com_ListenAt(ConnectionId, tcp(Host,Port)) :-
    ground(ConnectionId),
    % if variable address, look it up...
    ((var(Host) ; var(Port)) ->
        com_ResolveVariables([

```

```

        [cmd('-oaa_host',Host), cmd('-oaa_port', Port)],
        [env('OAA_HOST', Host), env_int('OAA_PORT', Port)],
        [setup('setup.pl',oaa_host, Host),
         setup('setup.pl',oaa_port, Port)]
    ])
| true),

repeat,
(on_exception(E,
    tcp_listen_at_port(Port, Host),
    Exception = E) ->
    ( var(Exception) ->
        assert(com_connection_info(ConnectionId, tcp, server,
            [addr(tcp(Host,Port)),oaa_host(Host),oaa_port(Port)],
            connected)),
            !
        | otherwise ->
            com_ask_about_tcp_exception(Port, Host, Response),
            ( Response == yes ->
                fail
            | otherwise ->
                halt
            )
        )
    )
|
    com_ask_about_tcp_exception(Port, Host, Response),
    ( Response == yes ->
        fail
    | otherwise ->
        halt
    )
).

```

```

com_ask_about_tcp_exception(Port, Host, Response) :-
    repeat,
    with_output_to_chars(
        format('Currently unable to access -w port -w.-n Try again? -w',
            [Host, Port, '[yes, no, help]']),
        Chars),
    name(Prompt, Chars),
    ask_oneof(Prompt, [yes, no, help], Response),
    ( Response == help ->
        com_print_tcp_exception_help,
        fail
    | otherwise ->
        !
    ).

```

```

com_print_tcp_exception_help :-
    write('

```

I've just attempted to listen on the specified port, but was unable to gain control of it. This could be because there's already a Facilitator, or some other program, making use of that port. Or, it could be that a Facilitator using that port was just terminated. In such cases, the port may be inaccessible for a brief period (usually only a few seconds, but sometimes more). It may help to kill any

client agents which may still be connected to the defunct Facilitator.

If you think the specified port may now be accessible, enter "y" and I'll try again. You may request retry any number of times.

If you want me to listen on a different port, enter "n", which will cause me to terminate. Then change your port specification (it's either in a setup file or an environment variable). Then restart me.

').

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      com_SendData(+ConnectionId, +Data)
% purpose:   Sends data to the specified connection ID
% remarks:
% - Checks format for destination connection
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
com_SendData(ConnectionId, Data) :-
    ground(ConnectionId);
    ( com_connection_info(ConnectionId, Type, _ClientServer, InfoList,
        connected),
      (Type = tcp ; Type = unknown), !,
      memberchk(connection(Dest), InfoList)
      ;
      format('-nError: cannot find open connection for -p!~n',
        [ConnectionId]),
      fail
    ),
    ( memberchk(format(F), InfoList) ->
      true
    | memberchk(agent_language(c), InfoList) ->
      F = special_case_c
    | otherwise ->
      F = default
    ),
    !,
    com_send_data_by_format(Dest, F, Data).

% quintus_binary: for inter-quintus communication
com_send_data_by_format(Dest, quintus_binary, Data) :- !,
    tcp_send(Dest, Data).
% prolog: a synonym for quintus_binary
com_send_data_by_format(Dest, prolog, Data) :- !,
    tcp_send(Dest, Data).

% pure_ascii: don't wrap data in term() wrapper
com_send_data_by_format(Dest, pure_ascii, Data) :- !,
    current_output(CurrentOutput),
    flush_output(CurrentOutput),
    tcp_output_stream(Dest, TcpOutput),
    set_output(TcpOutput),
```

```

WriteParams =
    [quoted(true),           % make input acceptable for read
      ignore_ops(false),    % false so list will be printed as '[1,2]'
      % !!! could be a problem with +, other opts.
      numbervars(true),     % print vars as f(A).
      character_escapes(false), % write actual character, not \255
      max_depth(0)],       % no depth limit

write_term(Data, WriteParams),

flush_output(TcpOutput),
set_output(CurrentOutput), !.

% special_case_c: This is the same as default, EXCEPT for the use of
% nl, nl. See comments within the clause for default format.
% Currently we don't understand why it matters.
com_send_data_by_format(Dest, special_case_c, Data) :- !,
    current_output(CurrentOutput),
    flush_output(CurrentOutput),
    tcp_output_stream(Dest, TcpOutput),
    set_output(TcpOutput),

    WriteParams =
        [quoted(true),           % make input acceptable for read
          ignore_ops(false),    % false so list will be printed as '[1,2]'
          % !!! could be a problem with +, other opts.
          numbervars(true),     % print vars as f(A).
          character_escapes(false), % write actual character, not \255
          max_depth(0)],       % no depth limit

    write_term(term(Data), WriteParams),
    write('.'),
    nl, nl,
    flush_output(TcpOutput),
    set_output(CurrentOutput), !.

% DefaultOAA: wrap in term() wrapper for easy parsing
com_send_data_by_format(Dest, _DefaultOAA, Data) :-
    current_output(CurrentOutput),
    flush_output(CurrentOutput),
    tcp_output_stream(Dest, TcpOutput),
    set_output(TcpOutput),

    WriteParams =
        [quoted(true),           % make input acceptable for read
          ignore_ops(false),    % false so list will be printed as '[1,2]'
          % !!! could be a problem with +, other opts.
          numbervars(true),     % print vars as f(A).
          character_escapes(false), % write actual character, not \255
          max_depth(0)],       % no depth limit

    write_term(term(Data), WriteParams),
    write('.'),
    % nl, nl,

% The preceding does not work between two Quintus agents
% (neither does a single nl, nor does it help to use nl(TcpOutput)),
% so we went to the following. However, the following does not work

```

```

% when a QP facilitator sends to the C interface agent.  For now,
% we'll solve this problem by defining the special_case_c format.
% (DLM, 97-04-09)
    put(TcpOutput, 10),
    % This causes the agents to disconnect (at least under UNIX):
    % put(TcpOutput, 13),

    flush_output(TcpOutput),
    set_output(CurrentOutput), !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      com_SelectEvent(+TimeOut, -Event)
% purpose:  Waits and returns an incoming event, or 'timeout' if TimeOut expires
% remarks:
% - TimeOut may be a real number, and represents seconds.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
com_SelectEvent(0, Event) :- !,
    on_exception(E,tcp_select(Event), com_print_err(E)).
com_SelectEvent(Seconds, Event) :-
    on_exception(E,tcp_select(Seconds, Event),com_print_err(E)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      com_print_err
% purpose:  Print error message if problem reading the event
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
com_print_err(E) :-
    format('~n===== READ ERROR !!! =====~n', []),
    format('| Messages in this block are rejected~n', []),
    format('| by the system.~n', []),
    format('-----~n', []),
    print_message(error, E),
    format('=====~n', []), fail.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      com_AddInfo
% purpose:  Adds or changes information about connection
% remarks:
% Info may be status(S), type(T), protocol(P) or any element (or list
% of elements) to be stored in InfoList.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
com_AddInfo(ConnectionId, NewInfo) :-
    retract(com_connection_info(ConnectionId, Protocol, Type,
        InfoList, Status)),
    (NewInfo = status(NewStatus), C = true ; NewStatus = Status),
    (NewInfo = protocol(NewProtocol), C = true ; NewProtocol = Protocol),
    (NewInfo = type(NewType), C = true ; NewType = Type),
    (NewInfo = [_H|_T] ->
        union([InfoList, NewInfo], NewInfoList)
    | (ground(C) ; union([InfoList, [NewInfo]], NewInfoList))
    ),
    assert(com_connection_info(ConnectionId, NewProtocol, NewType,

```

```
NewInfoList, NewStatus)), !.
```

```
*****  
% name:      com_GetInfo  
% purpose:   Looks up information about connection  
% remarks:  
%   Info may be status(S), type(T), protocol(P) or any element stored  
%   in InfoList.  
*****  
com_GetInfo(ConnectionId, Info) :-  
    com_connection_info(ConnectionId, Protocol, Type,  
        InfoList, Status),  
    (Info = status(Status) ;  
    Info = type(Type) ;  
    Info = protocol(Protocol) ;  
    memberchk(Info, InfoList)),  
    !.
```

```
*****  
%  
% name:      com_ResolveVariables  
% purpose:   Tries to instantiate the arguments by looking in the command  
%           line arguments, environment variables, and setup files  
% inputs:  
%   - VarList:  A list of lists: the first sublist that completely resolves  
%               provides the value for com_ResolveVariables.  
% remarks:  
%   sublists may contain elements in the following format:  
%     env(EnvVar, Val)      : looks for "EnvVar" in environment vars  
%     env_int(EnvVar, Val)  : Returns value for EnvVar as an integer  
%     cmd(CmdVar, Val)     : looks for "CmdVar <Val>" on command line  
%     setup(File,SVar, Val) : reads SVar from setup file File  
% example:  
%   resolves host and port by searching first commandline, then environment  
%   variables, finally reads setup file.  
%  
%     com_ResolveVariables([  
%       [cmd('-oaa_host',Host), cmd('-oaa_port', Port)],  
%       [env('OAA_HOST', Host), env_int('OAA_PORT', Port)],  
%       [setup('setup.pl',oaa_host, Host),  
%         setup('setup.pl',oaa_port, Port)]  
%     ])  
%  
*****  
com_ResolveVariables([VarList|_]) :-  
    com_resolve_variables(VarList), !.  
com_ResolveVariables([_VarList|Rest]) :-  
    com_ResolveVariables(Rest).
```

```
com_resolve_variables([]).
```

```
com_resolve_variables([env_int(EnvVar, Val)|Rest]) :- !,  
    environ(EnvVar, EnvAtom),  
    name(EnvAtom, EnvChars),
```

```

        number_chars(Val, EnvChars),
        com_resolve_variables(Rest).

com_resolve_variables([env(EnvVar, Val)|Rest]) :- !,
    environ(EnvVar, Val),
    com_resolve_variables(Rest).

com_resolve_variables([cmd(CmdVar, Val)|Rest]) :- !,
    % get command line arguments
    unix(argv(ListOfArgs)),
    append(_, [CmdVar, Val|_], ListOfArgs),
    com_resolve_variables(Rest).

com_resolve_variables([setup(File,SVar, Val)|Rest]) :- !,
    % read setup file to load all values
    com_read_setup_file(File),
    Pred =.. [SVar, Val],
    on_exception(_, Pred, fail),
    com_resolve_variables(Rest).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      com_read_setup_file
% purpose:   Finds and loads setup file
% remarks:
%   Always succeeds.
%   The search path for 'setup.pl' is as follows:
%       1. Current directory
%       2. Home directory for user
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
com_read_setup_file(File) :-
    com_already_loaded(File), !.
com_read_setup_file(File) :-
    ( absolute_file_name(File, LocalSetupFile),
      can_open_file(LocalSetupFile, read, fail) ->
        SetupFile = LocalSetupFile
    |
      concat('-/',File, HomeName),
      absolute_file_name(HomeName, UserSetupFile),
      can_open_file(UserSetupFile, read, fail) ->
        SetupFile = UserSetupFile
    ),
    (ground(SetupFile) ->
      format('Loading setup file:-n -w-n-n', [SetupFile]),
      ( com_consult(SetupFile, _) ->
        assert(com_already_loaded(File))
      | otherwise ->
        format('~w: A problem was encountered in loading the setup file-n',
          ['WARNING'])
      )
    | true).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

% name:    com_consult(+FilePath, -AbsFileName).
% purpose:
% remarks: We don't use Quintus' builtin consult, because it's too picky
%          about associating predicates with files.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
com_consult(FilePath, AbsFileName) :-
    absolute_file_name(FilePath, AbsFileName),
    can_open_file(AbsFileName, read, fail),
    open(AbsFileName, read, Stream),
    load_clauses(Stream),
    close(Stream).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    load_clauses(+Stream).
% purpose:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load_clauses(Stream) :-
    repeat,
    read_term(Stream, [], Term),
    ( Term = ':-'(_Body) ->
      true
    | Term = end_of_file ->
      true
    | otherwise ->
      load_clause(Term)
    ),
    ( at_end_of_file(Stream) ->
      !
    | otherwise ->
      fail
    ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    load_clause(+Term).
% purpose:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load_clause(Term) :-
    assert( Term ).

```

APPENDIX A.IV

Source code file named liboaa.pl.



```

%*****
% File      : liboaa.pl
% Primary Authors  : Adam Cheyer, David Martin
% Purpose   : Prolog version of library for the Open Agent Architecture
% Updated   : 12/98
%
% -----
% Unpublished-rights reserved under the copyright laws of the United States.
%
%
% Unpublished Copyright (c) 1998, SRI International.
% "Open Agent Architecture" and "OAA" are Trademarks of SRI International.
% -----
%
%
%*****
% Note: internal functions use the naming convention oaa_function_name(),
%       while public predicates use oaa_PublicPredicate().
%*****
% Version 2.0 (change oaa_version assertion)
% - corrects FromKS in do_events by changing event format to include this
%   info.
% - messages are only sent to READY agents. For previous versions, an
%   agent may be either READY or just OPEN.
%*****
% Version 2.1 (change oaa_version assertion)
% - triggers have 2 new arguments, OpMask and Template, and
%   more general semantics. Backwards compatibility is provided.
%*****
% Version 3.0 (change oaa_version assertion)
% - primitives changed to start with oaa_ (and _icl) prefixes
% - Major restructuring and cleanup, including many new capabilities,
%   for first public release (a.k.a. "OAA 2")
%*****

:- module(oaa,
    [icl_GetParamValue/2,
      icl_GetPermValue/2,
      icl_BasicGoal/1,
      icl_GoalComponents/4,
      icl_ConsistentParams/2,
      icl_BuiltIn/1,
      icl_ConvertSolvables/2,
      oaa_LibraryVersion/1,
      oaa_Register/3,
      oaa_RegisterCallback/2,
      oaa_ResolveVariables/1,
      oaa_Ready/1,
      oaa_MainLoop/1,
      oaa_SetTimeout/1,
      oaa_GetEvent/3,
      oaa_ProcessEvent/2,
      oaa_Interpret/2,
      oaa_DelaySolution/1,
      oaa_ReturnDelayedSolutions/2,
      oaa_AddDelayedContextParams/3,

```



```

    oaa_PostEvent/2,
    oaa_CanSolve/2,
    oaa_Version/3,
    oaa_Ping/3,
    oaa_Declare/5,
    oaa_DeclareData/3,
    oaa_Undeclare/3,
    oaa_Redeclare/3,
    oaa_AddData/2,
    oaa_RemoveData/2,
    oaa_ReplaceData/3,
    oaa_CheckTriggers/3,
    oaa_AddTrigger/4,
    oaa_RemoveTrigger/4,
    oaa_Solve/2,
    oaa_InCache/2,
    oaa_AddToCache/2,
    oaa_ClearCache/0,
    oaa_TraceMsg/2,
    oaa_ComTraceMsg/2,
    oaa_Inform/3,
    oaa_Id/1,
    oaa_Name/1
  ]).

```

```

%*****
%* RCS Header and internal version
%*****

```

```

% rcs version number
rcsid('$Header: /home/trestle4/OAA/src/V2/prolog/RCS/oaapl,v 1.127 1998/12/23
23:14:18 martin Exp cheyer $').

```

```

:- op(599,yfx,::).

```

```

%*****
% Include files
%*****

```

```

:- use_module(library(basics)).
:- use_module(library(read_sent)).
:- use_module(library(lists)).
:- use_module(library(sets)).
:- use_module(library(strings)).
:- use_module(library(files)).
:- use_module(library(environ)). % read environment vars
:- use_module(library(ctr)).
:- use_module(library(charsio)). % for sprintf and with_output_to_chars
:- use_module(library(ask)). % for ask_oneof
:- use_module(library(samsort)). % for samsort(Ordered,Raw,Sort)
:- use_module(library(date)). % for now(Time)

```

```

:- use_module(library(tcp), [tcp_now/1, tcp_time_plus/3]).

```

```

% IMPORTANT: COM module. We don't want to hard code the name of the

```

```

% file that contains module 'com'. So, when this file is loaded,
% we first check to see if module 'com' is already present, then
% we check to see if the file containing 'com' has been specified
% on the command line, and if neither of those works, we load the
% default file (./com_tcp).
%
% In the case where the module has already been
% loaded, the following seems like the right thing to do:
% :- use_module(com, _File, all).
% BUT when compiling, this approach results in "undefined" errors from
% qcon. Thus, for now, in oaa.pl, we are explicitly using com: with all
% calls to the com module.

:- ( current_predicate(_, com:_) ->
    use_module(com, _File, all)
  | unix(argv(ListOfArgs)), append(_, ['-com', File | _], ListOfArgs) ->
    use_module(File, all)
  | otherwise ->
    use_module(com_tcp, all)
  ).

%*****
% Global variables
%*****
:- dynamic
    oaa_already_loaded/1, % record if file already loaded
    oaa_solvable/1, % list of agent capabilities
    oaa_trigger/5, % a built-in solvable
    oaa_trace/1, % trace mode: on or off
    oaa_com_trace/1, % com_trace mode: on or off
    oaa_debug/1, % debug mode: on or off
    oaa_cache/2, % cached solutions
    oaa_event_buffer/1, % buffer of waiting events
    oaa_waiting_for/2, % used for recursive blocking solve
    oaa_waiting_event/1, % problem...
    oaa_timeout/1, % tcp timeout value (use oaa_SetTimeout)
    oaa_delay_table/5, % table of delayed solutions
    oaa_delay/2, % the current goal is delayed
    oaa_data_ref/3, % bookkeeping for 'data' solvables
    oaa_current_contexts/2, % Solve parameters to be propagated
    oaa_callback/2, % Record of app-specific callbacks
    % These may appear in setup.pl:
    oaa_host/1, % for root, my host; otherwise,
    % host of my parent
    oaa_port/1. % ... similarly ...

oaa_LibraryVersion(3.0).

% solvables shared by all agents
% Note: all built-in DATA solvables must be declared dynamic to avoid
% QP warnings and exceptions.
oaa_built_in_solvables([
    % @@DLM: If we do away with TriggerId, we could use param
    % unique_values(true).

```

```

    solvable(oaa_trigger(_TriggerId, _Type, _Condition, _Action, _Params),
              [type(data)], [write(true)])
]).

```

```

% We'll always have exactly one oaa_solvable fact. Note that application
% code should NOT include a declaration or clause for oaa_solvable/1.
oaa_solvable([]).

```

```

%*****
% Initialization and connection functions
%*****

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_Register
% purpose:  Once a comm link is established, either as a client to a Facilitator
%           or as a server for other agents, oaa_Register will setup and registration
%           information for this agent.
% inputs:
%   - ConnectionId: the symbolic connection Id (client or server connection)
%   - AgentName: the name of the agent
%   - Solvables: solvable list
% remarks:
%   The following information is stored about the current connection,
%   accessible through com_GetInfo(ConnectionId, Info):
%
%       oaa_name(Name)      : the name of the current agent
%       oaa_id(Id)         : the Id for the agent
%       connection(C)      : system-level communications handle
%                           (e.g., socket number)
%
%   if connecting as client, this is also available:
%       fac_id(Id)        : the Facilitator's Id
%       fac_name(Name)    : the Facilitator's name
%       fac_lang(L)       : the Facilitator's language
%       fac_version(V)    : the version of the Facilitator's agent library
%
%   In addition, the following predicates are written to parent Facilitator,
%   or locally if the ConnectionId is a server connection:
%
%       agent_host(Id, Name, Host)
%
%   Solvables are also written using oaa_Declare()
%
%   It is possible for an agent to create both server and client connections:
%   such an agent was classified in OAA 1.0 as an agent of class "node"
%   (as opposed to a pure client "leaf" or pure server "root").
%
% examples:
%   % connecting to a Facilitator
%       MySolvables = [do(something)],
%       com_Connect(parent, ConnectionInfo),
%       oaa_Register(parent, my_agent_name, MySolvables).
%
%   % connecting as a Facilitator

```

```

%   MySolvables = [],
%   com_ListenAt(incoming, ConnectionInfo),
%   oaa_Register(incoming, root, MySolvables).
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% For client connecting to Facilitator
oaa_Register(ConnectionId, AgentName, Solvables) :-
    % succeeds only if exists an open client connection for ConnectionId
    %   as created by com_Connect()
    com:com_connection_info(ConnectionId, _Protocol, client, _Info,
connected),

    com:com_AddInfo(ConnectionId, oaa_name(AgentName)),

    % FIXED HACK: default now works thanks to update in com_tcp.pl for
    %   the default mode
    % HACK!!! Why doesn't this work right without it?
    % for some reason, when we send the handshaking info in
    %   default mode (instead of quintus_binary), the facilitator's
    %   tcp_select(VerySmallTimeout, Event) doesn't timeout!!!!
    %   So it keeps hanging until some other event (such as disconnect)
    %   arrives.
    com:com_AddInfo(ConnectionId, format(default)),

    % lookupversion number
    oaa_LibraryVersion(Version),

    %%% handshaking with Facilitator -- exchange information...
    % note: for this first communication, no format is defined for the
    %   connection, so it will be sent using default (ascii) format.
    %   Information coming back from Facilitator will update the
    %   format() field for the connection, improving future
    %   communication.
    com:com_SendData(ConnectionId,
        event(ev_connect([oaa_name(AgentName), agent_language(prolog),
        format(quintus_binary), agent_version(Version)]), [])),

    % Get the connection acknowledgement:
    % potential bug: what if selected event is NOT from FacId connection?
    oaa_GetEvent(ConnEvent, _Parms, 0),
    ConnEvent = ev_connected(FacInfoList),
    com:com_AddInfo(ConnectionId, FacInfoList),

    oaa_Id(MyId),

    % write host
    ( environ('HOST', MyHost) ->
        oaa_AddData(agent_host(MyId, AgentName, MyHost), [address(parent)]
| true),

    % Declare solvables (and post to parent facilitator):
    % Note: OK if Solvables = [].
    oaa_Declare(Solvables, [], [], [if_exists(overwrite)], _).

% For Faciliator serving client agents
oaa_Register(ConnectionId, AgentName, Solvables) :-

```

```

% succeeds only if exists an open client connection for ConnectionId
%   as created by com_Connect()
com:com_connection_info(ConnectionId, _Protocol, server, _Info,
connected),

AgentId = 0, % A facilitator's ID is always 0
com:com_AddInfo(ConnectionId, [oaa_id(AgentId), oaa_name(AgentName)]),

% The fac. records its own agent_data in the same way as its clients'.
% Note that we can't call oaa_add_data_local until after the solvables
% have been declared, and we can't declare solvables until we're
% open - so we have to bootstrap this assertion:
oaa_assertz(agent_data(AgentId, open, [], AgentName), AgentId, _),

% Note: OK if Solvables = [].
oaa_Declare(Solvables, [], [], [if_exists(overwrite)], _),

% write host
( environ('HOST', MyHost) ->
. oaa_add_data_local(agent_host(AgentId, AgentName, MyHost), [])
| true).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% name:      oaa_ResolveVariables(+VariableList)
% purpose:  Tries to instantiate the arguments by looking in the command
%           line arguments, environment variables, and setup files
% inputs:
% - VarList: A list of lists: the first sublist that completely resolves
%           provides the value for oaa_ResolveVariables.
% remarks:
% sublists may contain elements in the following format:
%   env(EnvVar, Val)      : looks for "EnvVar" in environment vars
%   env_int(EnvVar, Val)  : Returns value for EnvVar as an integer
%   cmd(CmdVar, Val)     : looks for "CmdVar <Val>" on command line
%   setup(SVar, Val)     : reads SVar from setup file
% example:
% resolves host and port by searching first commandline, then environment
% variables, finally reads setup file.
%
%   oaa_ResolveVariables([
%     [cmd('-oaa_host', Host), cmd('-oaa_port', Port)],
%     [env('OAA_HOST', Host), env_int('OAA_PORT', Port)],
%     [setup(oaa_host, Host), setup(oaa_port, Port)]
%   ])
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_ResolveVariables([VarList|_]) :-
    oaa_resolve_variables(VarList), !.
oaa_ResolveVariables([_VarList|Rest]) :-
    oaa_ResolveVariables(Rest).

oaa_resolve_variables([]).

oaa_resolve_variables([env_int(EnvVar, Val)|Rest]) :- !,

```

```

    environ(EnvVar, EnvAtom),
    name(EnvAtom, EnvChars),
    number_chars(Val, EnvChars),
    oaa_resolve_variables(Rest).

oaa_resolve_variables([env(EnvVar, Val)|Rest]) :- !,
    environ(EnvVar, Val),
    oaa_resolve_variables(Rest).

oaa_resolve_variables([cmd(CmdVar, Val)|Rest]) :- !,
    % get command line arguments
    unix(argv(ListOfArgs)),
    append(_, [CmdVar, Val|_], ListOfArgs),
    oaa_resolve_variables(Rest).

oaa_resolve_variables([setup(SVar, Val)|Rest]) :- !,
    % read setup file to load all values
    oaa_read_setup_file,
    Pred =.. [SVar, Val],
    on_exception(_, Pred, fail),
    oaa_resolve_variables(Rest).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_read_setup_file
% purpose:  Finds and loads setup file
% remarks:
%   Always succeeds.
%   The search path for 'setup.pl' is as follows:
%   1. Current directory
%   2. Home directory for user
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_read_setup_file :-
    oaa_already_loaded(setup), !.
oaa_read_setup_file :-
    ( absolute_file_name('setup.pl', LocalSetupFile),
      can_open_file(LocalSetupFile, read, fail) ->
        SetupFile = LocalSetupFile
    | absolute_file_name('-/setup.pl', UserSetupFile),
      can_open_file(UserSetupFile, read, fail) ->
        SetupFile = UserSetupFile
    ),
    (ground(SetupFile) ->
      format('Loading OAA setup file:-n -w-n', [SetupFile]),
      ( oaa_consult(SetupFile, _) ->
        assert(oaa_already_loaded(setup))
      | otherwise ->
        format('-w: A problem was encountered in loading the setup file-n',
          ['WARNING'])
      )
    | true).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_Ready
% purpose:  Changes the agent's 'open' status to 'ready', indicating that the
%           agent is now ready to receive messages.
% remarks:
%   if requested, prints 'Ready' to standard out.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_Ready(ShouldPrint) :-

```

```

    % replaces 'open' status with 'ready'.
    ((\+ oaa_class(root), oaa_Name(MySymbolicName)) ->
     oaa_PostEvent(ev_ready(MySymbolicName), [])
     | true),

    % if ShouldPrint, print ready
    (on_exception(_,ShouldPrint,fail) ->
     format('Ready.-n', [])
     | true).

```

```

%*****
% Classifying and Manipulating ICL expressions
%*****

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      icl_BuiltIn(+Goal).
% purpose:  Test whether an expression is an ICL built-in goal.
% remarks:
%   - icl_BuiltIn differs significantly from the Quintus Prolog predicate
%     built_in, in that here we do not include basic constructors such
%     as ',' and ';'.
%   - oaa_Interpret/2 must be defined for every goal for which
%     icl_BuiltIn succeeds.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
icl_BuiltIn(( _A = _B)).
icl_BuiltIn(( _A == _B)).
icl_BuiltIn(( _A \== _B)).
icl_BuiltIn(( _A =< _B)).
icl_BuiltIn(( _A >= _B)).
icl_BuiltIn(( _A < _B)).
icl_BuiltIn(( _A > _B)).
icl_BuiltIn(member( _,_)).
icl_BuiltIn(memberchk( _,_)).
icl_BuiltIn(findall( _,_,_)).
icl_BuiltIn(icl_ConsistentParams( _,_)).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      icl_BasicGoal(+Goal).
% purpose:  Test whether an expression is an ICL basic (non-compound) goal;
%           that is, just a functor with 0 or more arguments.
% remarks:
%   - Basic goals include built-in's as well as solvables.
%   - This is a syntactic test; that is, we're not checking whether the
%     Goal is a declared solvable.

```

```

*****
icl_BasicGoal(Goal) :-
    var(Goal), !, fail.
icl_BasicGoal(Goal) :-
    is_list(Goal), !, fail.
icl_BasicGoal(Goal) :-
    icl_compound_goal(Goal), !, fail.
icl_BasicGoal(Goal) :-
    icl_BuiltIn(Goal),
    !.
icl_BasicGoal(Goal) :-
    Goal =.. [Functor | _],
    atom(Functor).

```

```

*****
% name:      icl_compound_goal(+Goal).
% purpose:  Test whether an expression is an ICL compound goal.
*****
icl_compound_goal(_X:_Y).
icl_compound_goal(_X::_Y).
icl_compound_goal((\+ _P)).
icl_compound_goal((_P -> _Q ; _R)).
icl_compound_goal((_P -> _Q)).
icl_compound_goal((_X, _Y)).
icl_compound_goal((_X ; _Y)).

```

```

*****
% name:      icl_GoalComponents(+ICLGoal, -A, -G, -P).
%            icl_GoalComponents(-ICLGoal, +A, +G, +P).
%            icl_GoalComponents(+ICLGoal, +A, +G, +P).
% purpose:  Assemble, disassemble, or match against the top-level components
%           of an ICL goal.
% remarks:
%   - The top-level structure of an ICL goal is Address:Goal::Params,
%     with Address and Params BOTH OPTIONAL. Thus, every ICL goal
%     either explicitly or implicitly includes all three components.
%   - This may be used with any ICL goal, basic or compound.
%   - When P is missing, its value is returned or matched as []. When A is
%     missing, its value is returned or matched as 'unknown'.
*****

```

```

% The first 4 clauses handled all cases where the ICL Goal is bound;
% the remainder handle those where it is a var.

```

```

icl_GoalComponents(A:G::P, Address, Goal, Params) :-
    \+ var(A), \+ var(G), \+ var(P),
    !,
    Address = A, Goal = G, Params = P.
icl_GoalComponents(A:G, Address, Goal, Params) :-
    \+ var(A), \+ var(G),
    !,
    Address = A, Goal = G, Params = [].
icl_GoalComponents(G::P, Address, Goal, Params) :-
    \+ var(G), \+ var(P),
    !,
    Address = unknown, Goal = G, Params = P.

```



```

icl_GoalComponents(G, Address, Goal, Params) :-
    \+ var(G),
    !,
    Address = unknown, Goal = G, Params = [].
icl_GoalComponents(Goal, unknown, Goal, []) :-
    !.
icl_GoalComponents(Address:Goal, Address, Goal, []) :-
    !.
icl_GoalComponents(Goal::Params, unknown, Goal, Params) :-
    !.
icl_GoalComponents(Address:Goal::Params, Address, Goal, Params) :-
    !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Permissions and parameter lists
%
% These procedures are used in processing solvables permissions, and
% parameter lists of all kinds (including those used with solvables,
% those contained in events, and those used in calls to various
% library procedures).
%
% All permissions and many parameters have default values.
%
% Permissions and parameters lists have a standard form, as defined by
% the predicates below. To save bandwidth and promote readability, a
% "perm" or "param" list in standard form OMITs default values. For
% easier processing (e.g., comparing/merging param lists), boolean
% params in standard form always include a single argument 'true' or
% 'false'.
%
% In definitions of solvables and calls to documented library
% procedures, it's OK to include default params in a Params list, if
% desired. For boolean params, when the intended value is 'true', it's
% OK just to specify the functor, for example, instead of
% cache(true), it's OK just to include 'cache'.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% icl_standardize_perms(+Perms, +KeepDefaults, -Standardized).

icl_standardize_perms([], _KeepDefaults, []).
icl_standardize_perms([Perm | Perms], KeepDefaults, [SPerm | SPerms]) :-
    icl_perm_standard_form(Perm, SPerm),
    ( KeepDefaults ; (\+ icl_perm_default(SPerm)) ),
    !,
    icl_standardize_perms(Perms, KeepDefaults, SPerms).
icl_standardize_perms([_Perm | Perms], KeepDefaults, SPerms) :-
    icl_standardize_perms(Perms, KeepDefaults, SPerms).

icl_perm_standard_form(Perm, SPerm) :-
    atom(Perm),
    !,
    SPerm =.. [Perm, true].
icl_perm_standard_form(Perm, Perm).

icl_perm_default(call(true)).

```

```

icl_perm_default(read(false)).
icl_perm_default(write(false)).

% icl_standardize_params(+Params, +KeepDefaults, -Standardized).
%
% Normally there's no need to keep the default value of a param,
% but there are exceptional situations.  If KeepDefaults is true,
% default values are kept.

icl_standardize_params([], _, []).
icl_standardize_params([Param | Rest], KeepDefaults, AllStandardized) :-
    icl_param_standard_form(Param, FullStandardized),
    ( KeepDefaults ->
        Standardized = FullStandardized
    | otherwise ->
        icl_remove_default_params(FullStandardized, Standardized)
    ),
    icl_standardize_params(Rest, KeepDefaults, RestStandardized),
    append(Standardized, RestStandardized, AllStandardized).

% icl_param_standard_form(+Param, -StandardParams).
%
% Maps from an element of a parameter list to a list of elements
% in standardized form.  The parameter list element can be from
% any context (from a call to Solve, AddTrigger, AddData, etc.).

icl_param_standard_form(reply(false), [reply(none)]) :-
    !.
    % broadcast has been retained, as a synonym for reply(none):
icl_param_standard_form(broadcast, [reply(none)]) :-
    !.
icl_param_standard_form(broadcast(true), [reply(none)]) :-
    !.
icl_param_standard_form(broadcast(false), [reply(true)]) :-
    !.
icl_param_standard_form(address(Addr), [address(SAddr)]) :-
    !,
    icl_standardize_address(Addr, SAddr).
icl_param_standard_form(strategy(query), [parallel_ok(true)]) :-
    !.
icl_param_standard_form(strategy(action),
    [parallel_ok(false), solution_limit(1)]) :-
    !.
icl_param_standard_form(strategy(inform),
    [parallel_ok(true), reply(none)]) :-
    !.
icl_param_standard_form(callback(Mod:Proc), [callback(Mod:Proc)]) :-
    !.
icl_param_standard_form(callback(Proc), [callback(user:Proc)]) :-
    !.
icl_param_standard_form(Param, [SParam]) :-
    atom(Param),
    !,
    SParam =.. [Param, true].
icl_param_standard_form(Param, [Param]).

icl_param_default(from(unknown)).

```

```

icl_param_default(priority(5)).
icl_param_default(utility(5)).
icl_param_default(if_exists(append)).
icl_param_default(type(procedure)).
icl_param_default(private(false)).
icl_param_default(single_value(false)).
icl_param_default(unique_values(false)).
icl_param_default(rules_ok(false)).
icl_param_default(bookkeeping(true)).
icl_param_default(persistent(false)).
icl_param_default(at_beginning(false)).
icl_param_default(do_all(false)).
icl_param_default(reflexive(true)).
icl_param_default(parallel_ok(true)).
icl_param_default(reply(true)).
icl_param_default(block(true)).
icl_param_default(cache(false)).
icl_param_default(flush_events(false)).
icl_param_default(recurrence(when)).

icl_remove_default_params([], []).
icl_remove_default_params([Param | Rest], Removed) :-
    icl_param_default(Param),
    !,
    icl_remove_default_params(Rest, Removed).
icl_remove_default_params([Param | Rest], [Param | Removed]) :-
    icl_remove_default_params(Rest, Removed).

% icl_GetParamValue(+Param, +ParamList).
%
% Param must have a functor, but its argument(s) can be either ground
% or variables.  E.g., persistent(X).
%
% To get or test the value of a parameter that has a default, it is
% best to call icl_GetParamValue.  For a parameter that has no default,
% you can use icl_GetParamValue OR memberchk.

icl_GetParamValue(Param, ParamList) :-
    predicate_skeleton(Param, Skel),
    memberchk(Skel, ParamList),
    !,
    Skel = Param.
icl_GetParamValue(Param, _ParamList) :-
    predicate_skeleton(Param, Skel),
    icl_param_default(Skel),
    !,
    Skel = Param.

icl_GetPermValue(Perm, PermList) :-
    predicate_skeleton(Perm, Skel),
    memberchk(Skel, PermList),
    !,
    Skel = Perm.
icl_GetPermValue(Perm, _PermList) :-
    predicate_skeleton(Perm, Skel),
    icl_perm_default(Skel),
    !,

```

Skel = Perm.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      icl_ConsistentParams(+Test, +ParamList)
% purpose:  Often used in solvable declarations to filter on a certain
%           condition.
% definition:
%           Test a param list: if one or more values are given in a parameter
%           list for parameter ParamName, then ParamValue must be defined as
%           one of the values to succeed. If ParamValue is NOT defined, then
%           icl_ConsistentParams succeeds.
% example:
%           A natural language parser agent can only handle English definitions:
%
%           convert(nl, icl,Input,Params,Output) :-
%               icl_ConsistentParams(language(english),Params).
%
%           if "language(english)" is defined in parameter list of a solve request,
%           the nl agent will receive the request.
%           if "language(spanish)" is defined in the parameter list, the nl agent
%           WILL NOT receive the request.
%           if no language parameter is specified, the request WILL be sent
%           if "language(X)" is specified, the request WILL be sent to the nl agent
% remarks:
%           - Test may contain either a single predicate or a list of test predicates,
%           in which case icl_ConsistentParams will execute all consistency tests.
%           - Interesting note: icl_ConsistentParams() checks consistency as a
%           relation between the two arguments, so it doesn't matter which argument
%           specifies the test list and which the parameters to test.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
icl_ConsistentParams(_TestList, []) :- !.
icl_ConsistentParams([], _ParamList) :- !.
icl_ConsistentParams([Test|RTest], [P1|RParams]) :- !,
    ParamList = [P1|RParams],
    predicate_skeleton(Test, TestWithVars),
    (memberchk(TestWithVars, ParamList) ->
        memberchk(Test, ParamList)
    | true),
    icl_ConsistentParams(RTest, ParamList).
% either Test or Params is NOT a list
icl_ConsistentParams(Test, Param) :-
    (Test = [_|_] ->
        NewTest = Test
    | NewTest = [Test]),
    (Param = [_|_] ->
        NewParam = Param
    | NewParam = [Param]),
    icl_ConsistentParams(NewTest, NewParam).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Agent identity and addressing
%
% Every agent (including facilitators) has a symbolic name, a full address,
```

```

% and a local address (or "local ID"). A full address has the form:
%   addr(tcp(Host,Port))           for a facilitator (if TCP is protocol)
%   addr(tcp(Host,Port), LocalID)  for a client agent.
%
% Even though it doesn't appear in the full address, a facilitator also
% has a local ID, for consistency and convenient reference. The
% local ID of a client agent is assigned to it by its facilitator.
% This, and the facilitator's local ID, are passed to the client at
% connection time.
%
% Full addresses are globally unique, and local addresses are unique with
% respect to a facilitator. Symbolic names are NOT unique in any sense.
%
% The local ID happens to be an integer, but developers should not rely
% on this.
%
% When specifying addresses, in address/1 params for calls to
% oaa_AddData, oaa_Solve, etc., either names or addresses may be used.
% In addition, for convenience, reserved terms 'self', 'parent', and
% 'facilitator' may also be used.
%
% More precisely, the address parameter may contain any of the following:
% a full address; a local ID (when the addressee is known to be either
% the facilitator or a peer client); a name, enclosed in the name/1 functor;
% 'self'; 'parent'; or 'facilitator'. ('parent' and 'facilitator' are
% synonymous.)
%
% Address parameters are standardized as follows: A full address for the
% local facilitator or a peer client is changed to the local ID; all
% other full addresses are left as is. Names are left as is. 'self',
% 'parent', and 'facilitator' are changed to the appropriate local ID.
%
%*****

% This can only be used AFTER oaa_SetupCommunication has been called,
% because of the reliance here on com:com_connection_info/5.
icl_standardize_address(Addr, SAddr) :-
    \+ is_list(Addr),
    !,
    icl_standardize_address([Addr], SAddr).
icl_standardize_address([], []).
icl_standardize_address([Addr | Addr], [SAddr | SAddrs]) :-
    icl_standardize_addressee(Addr, SAddr),
    !,
    icl_standardize_address(Addr, SAddrs).
icl_standardize_address([_Addr | Addr], SAddrs) :-
    icl_standardize_address(Addr, SAddrs).

%*****

icl_standardize_addressee(addr(Addr), ParentId) :-
    com:com_GetInfo(parent, addr(Addr)),
    com:com_GetInfo(parent, fac_id(ParentId)),
    !.
icl_standardize_addressee(addr(Addr), addr(Addr)) :-
    !.
icl_standardize_addressee(addr(Addr, LID), LID) :-

```

```

    com:com_GetInfo(parent, addr(Addr)),
    !.
icl_standardize_addressee(addr(Addr, LID), LID) :-
    com:com_GetInfo(incoming, addr(Addr)),
    !.
icl_standardize_addressee(addr(Addr, LID), addr(Addr, LID)) :-
    !.
icl_standardize_addressee(name(Name), name(Name)) :-
    !,
    icl_name(Name).
icl_standardize_addressee(Name, name(Name)) :-
    icl_name(Name),
    !,
    format('~w (~w): addressee name, in address/1 param, should be specified
as:~n name(~w)~n',
        ['WARNING', 'liboaa.pl', Name]).
icl_standardize_addressee(Id, TrueId) :-
    icl_true_id(Id, TrueId),
    !.
icl_standardize_addressee(Whatever, _) :-
    format('~w (~w): Illegal addressee, in address/1 param, discarded:~n ~w~n',
        ['WARNING', 'liboaa.pl', Whatever]),
    fail.

icl_true_id(self, Me) :-
    !,
    oaa_Id(Me).
icl_true_id(parent, Parent) :-
    !,
    com:com_GetInfo(parent, fac_id(Parent)).
icl_true_id(facilitator, Parent) :-
    !,
    com:com_GetInfo(parent, fac_id(Parent)).
icl_true_id(Id, Id) :-
    icl_id(Id).

icl_id(Num) :-
    integer(Num),
    Num >= 0.

icl_name(self) :-
    !, fail.
icl_name(parent) :-
    !, fail.
icl_name(facilitator) :-
    !, fail.
icl_name(Atom) :-
    atom(Atom).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      icl_ConvertSolvables(+ShorthandSolvables, -StandardSolvables).
%           icl_ConvertSolvables(-ShorthandSolvables, +StandardSolvables).
%
% purpose:  Convert between shorthand and standard forms of solvables list.
% remarks:
%         - In the standard form, each element is a term solvable(Goal,

```

```

% Params, Permissions), with Permissions and Params both lists.
% In the Permissions and Params lists, values appear only when they
% are OTHER than the default.
% - In the shorthand form, each element can be solvable/3, as above,
% or solvable(Goal, Params), or solvable(Goal), or just Goal.
% - Note that "shorthand" means "anything goes" - so shorthand
% solvables are a superset of standard solvables.
% - Permissions (defaults in square brackets):
%   call(T_F) [true], read(T_F) [false], write(T_F) [false]
% - Params (defaults in square brackets):
%   type(Data_Procedure) [procedure],
%   callback(Functor) [no default]
%   utility(N) [5]
%   synonym(SynonymHead, RealHead) [none]
%   rules_ok(T_F) [false],
%   single_value(T_F) [false],
%   unique_values(T_F) [false],
%   private(T_F) [false]
%   bookkeeping(T_F) [true]
%   persistent(T_F) [false]
% - Refer to Agent Library Reference Manual for details on Permissions
% and Params.
% - (@@DLM) This might be the place to check the validity of solvables,
% such as using only built-ins in tests. Also, check for dependencies
% between solvables; e.g., when persistent(false) is there,
% bookkeeping(true) must also be there.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
icl_ConvertSolvables(ShorthandSolvables, StandardSolvables) :-
    var(StandardSolvables),
    !,
    icl_standardize_solvables(ShorthandSolvables, StandardSolvables).
icl_ConvertSolvables(ShorthandSolvables, StandardSolvables) :-
    icl_readable_solvables(StandardSolvables, ShorthandSolvables).

% icl_standardize_solvables(+ShorthandSolvables,
%                            -StandardSolvables).
icl_standardize_solvables([], []).
icl_standardize_solvables([Shorthand | RestSH], [Standard | RestStan]) :-
    icl_standardize_solvable(Shorthand, Standard),
    icl_standardize_solvables(RestSH, RestStan).

% icl_standardize_solvable(+Shorthand, -Standard).
icl_standardize_solvable(solvable((Goal :- Test), Params, Perms), Standard) :-
    !,
    append([test(Test)], Params, NewParams),
    icl_standardize_solvable(solvable(Goal, NewParams, Perms), Standard).
icl_standardize_solvable(solvable((Goal :- Test), Params), Standard) :-
    !,
    icl_standardize_solvable(solvable(Goal, [test(Test) | Params], []),
        Standard).
icl_standardize_solvable(solvable((Goal :- Test)), Standard) :-
    !,
    icl_standardize_solvable(solvable(Goal, [test(Test)], []), Standard).
icl_standardize_solvable((Goal :- Test), Standard) :-
    !,
    icl_standardize_solvable(solvable(Goal, [test(Test)], []), Standard).

```

```

icl_standardize_solvable(solvable(Goal, Params, Perms),
                        solvable(Goal, NewParams, NewPerms)) :-
    !,
    icl_standardize_params(Params, false, NewParams),
    icl_standardize_perms(Perms, false, NewPerms).
icl_standardize_solvable(solvable(Goal, Params),
                        solvable(Goal, NewParams, [])) :-
    !,
    icl_standardize_params(Params, false, NewParams).
icl_standardize_solvable(solvable(Goal), solvable(Goal, [], [])) :- !.
icl_standardize_solvable(Goal, solvable(Goal, [], [])) :- !.

% icl_readable_solvable(+StandardSolvables,
%                       -ShorthandSolvables).
% This is provided for use in "pretty-printing" solvables, in trace
% messages, etc.
icl_readable_solvable([], []).
icl_readable_solvable([Standard | RestStan], [Shorthand | RestSh]) :-
    icl_readable_solvable(Standard, Shorthand),
    icl_readable_solvable(RestStan, RestSh).

% icl_readable_solvable(+Standard, -Shorthand).
icl_readable_solvable(solvable(Goal, [], []), Goal) :- !.
icl_readable_solvable(solvable(Goal, Params, []), solvable(Goal, Params)) :- !.
icl_readable_solvable(solvable(Goal, Params, Perms),
                        solvable(Goal, Params, Perms)) :- !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      icl_minimally_instantiate_solvables(+ShorthandSolvables,
%                                               -MinimalSolvables).
% purpose:  Convert from shorthand (or standard form) to minimally instantiated
%           solvables list.
% remarks:  - This is special-purpose. It's used to massage a list of solvables
%           that are to be UNdeclared, to make sure each of them will unify
%           with some existing solvable.  Perms and Params are completely
%           ignored in the unification; only the Goal is relevant.  So each
%           minimally instantiated solvable is simply solvable(Goal, _, _).
%           - Note that "shorthand" means "anything goes" - so shorthand
%           solvables are a superset of standard solvables.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% icl_minimally_instantiate_solvables(+ShorthandSolvables,
%                                     -Solvables).
icl_minimally_instantiate_solvables([], []).
icl_minimally_instantiate_solvables([Shorthand | RestSH],
                                    [Minimal | RestMin]) :-
    icl_minimally_instantiate_solvable(Shorthand, Minimal),
    icl_minimally_instantiate_solvables(RestSH, RestMin).

% icl_minimally_instantiate_solvable(+Shorthand, -Minimal).
icl_minimally_instantiate_solvable(solvable((Goal :- _Test), Params, Perms),
                                    Minimal) :-
    !,
    icl_minimally_instantiate_solvable(solvable(Goal, Params, Perms),
                                        Minimal).
icl_minimally_instantiate_solvable(solvable((Goal :- _Test), Params),
                                    Minimal) :-

```



```

!,
    icl_minimally_instantiate_solvable(solvable(Goal, Params, []), Minimal).
icl_minimally_instantiate_solvable(solvable((Goal :- _Test)), Minimal) :-
!,
    icl_minimally_instantiate_solvable(solvable(Goal, [], []), Minimal).
icl_minimally_instantiate_solvable((Goal :- _Test), Minimal) :-
!,
    icl_minimally_instantiate_solvable(solvable(Goal, [], []), Minimal).
icl_minimally_instantiate_solvable(solvable(Goal, _Params, _Perms),
    solvable(Goal, _, _)) :-
!.
icl_minimally_instantiate_solvable(solvable(Goal, _Params),
    solvable(Goal, _, _)) :-
!.
icl_minimally_instantiate_solvable(solvable(Goal), solvable(Goal, _, _)) :- !.
icl_minimally_instantiate_solvable(Goal, solvable(Goal, _, _)) :- !.

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

% name:    oaa_goal_matches_solvable(+Goal, +Solvables,
%                                     -RealGoal, -MatchedSolvable).
% purpose: Determine whether a call to Goal is handled by the agent with
%          these Solvables.
% arguments:
%   - Goal must be non-compound (basic) to match: no address, no params,
%     no subgoals.
%   - Solvables must be in standard form.
%   - RealGoal is what should actually be called, after taking synonyms
%     into account.
%   - MatchedSolvable is the solvable record corresponding to RealGoal.
% remarks:
%   - A solvable's params may contain a single test, but it can
%     be compound:
%       solvable(g(X), [test((X > 1,X < 10))], [...]).
%     Tests should contain only prolog builtins.
%   - Any solvable can be a synonym of another solvable (including a
%     synonym of a synonym), but eventually there must be a non-synonym
%     solvable. Synonyms must be used with care. If predicate A
%     is synonymed to predicate B, there must be a solvable for clause B,
%     for A to be usable.
%   - When a predicate A is synonymed to predicate B, all other params
%     and all permissions associated with A are ignored.
%   - Uses would_unify (and \+ \+) so that any variables in the goal are
%     not bound by the solvable, thereby unnecessarily constraining query
%     I forget why: I think it was because we had some problems
%     matching solutions coming back. However, this has an unusual
%     side effect: if your solvable is t(6) and your query is t(X),
%     the query arrives at the agent as t(X), not t(6), which might
%     be unexpected. Look into this more someday...
%   - However, when Goal is a synonym, variables in the synonym param DO
%     get unified correctly.

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

oaa_goal_matches_solvable(Goal, Solvables, RealGoal, RealMatched) :-
    oaa_built_in_solvable(BuiltIns),
    append(BuiltIns, Solvables, AllSolvables),
    oaa_goal_in_solvable(Goal, AllSolvables, Matched),
    Matched = solvable(_, Params, _),

```

```

% See if Goal is a synonym predicate
( icl_GetParamValue(synonym(Goal, SynGoal), Params) ->
  oaa_goal_matches_solvable(SynGoal, Solvables, RealGoal, RealMatched)
| otherwise ->
  RealGoal = Goal,
  RealMatched = Matched
),
!.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_goal_in_solvable(+Goal, +Solvables, -MatchedSolvable).
% purpose:   Determine whether a call to Goal is handled by the agent with
%            these Solvables.
% purpose:   Determine whether Goal appears in Solvables, with
%            appropriate Params and Perms for it to be called.
% arguments:
%   - Goal must be non-compound (basic) to match: no address, no params,
%     no subgoals.
%   - Solvables must be in standard form.
% remarks:
%   - Should not be called directly; only by oaa_goal_matches_solvable.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_goal_in_solvable(Goal, [solvable(G1,Params,Perms) | _Rest],
  solvable(G1,Params,Perms)) :-
  would_unify(Goal, G1),
  icl_GetParamValue(synonym(Goal, _RealGoal), Params),
  !.
oaa_goal_in_solvable(Goal, [solvable(G1,Params,Perms) | _Rest],
  solvable(G1,Params,Perms)) :-
  would_unify(Goal, G1),
  icl_GetPermValue(call(true), Perms),
  ( icl_GetParamValue(test(T), Params) ->
    \+ \+ oaa_Interpret((Goal = G1, T), [])
  | otherwise ->
    true
  ),
  !.
oaa_goal_in_solvable(Goal, [_|Rest], Matched) :-
  oaa_goal_in_solvable(Goal, Rest, Matched).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_data_matches_solvable(+Clause, +Solvables, +Perm
%            -RealClause, -MatchedSolvable).
% purpose:   Determine whether Clause can be read or written by the agent with
%            these Solvables, and return the "real" form of the clause that
%            takes synonyms into account.
% arguments:
%   - Clause must be non-compound (basic) to match: no address, no params,
%     no subClauses.
%   - Solvables must be in standard form.
%   - _Perm is 'read' or 'write'.
%   - RealClause is what should actually be used (asserted, retracted,
%     replaced).
%   - MatchedSolvable is the solvable record corresponding to RealClause.
% remarks:

```

```

% "Writing" means making an assertion.
% "Reading" is different than "calling". "Reading" is retrieving the
% definition clauses of a predicate (including the bodies, if any).
% Reading is not currently supported by any library procedures.
% Any solvable can be a synonym of another solvable (including a
% synonym of a synonym), but eventually there must be a non-synonym
% solvable. Synonyms must be used with care. If predicate A
% is synonymed to predicate B, there must be a solvable for clause B,
% for A to be usable.
% When a predicate A is synonymed to predicate B, all other params
% and all permissions associated with A are ignored.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_data_matches_solvable(Cls, Solv, Perm, RealCls, RealMatch) :-
  oaa_built_in_solvable(Cls, BuiltIns),
  append(BuiltIns, Solv, AllSolv),
  oaa_data_in_solvable(Cls, AllSolv, Perm, Match),
  Match = solvable(_, Params, _),
  ( Cls = (Head :- Body) ->
    true
  | otherwise ->
    Head = Cls
  ),
  % See if Cls is a synonym predicate
  ( icl_GetParamValue(synonym(Head, SynHead), Params) ->
    ( Cls = (Head :- Body) ->
      SynCls = (SynHead :- Body)
    | otherwise ->
      SynCls = SynHead
    ),
    oaa_data_matches_solvable(SynCls, Solv, Perm,
      RealCls, RealMatch)
  | otherwise ->
    RealCls = Cls,
    RealMatch = Match
  ),
  !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name: oaa_data_in_solvable(+Clause, +Solvables, +Perm, -MatchedSolvable).
% purpose: Determine whether (the Head of) Clause appears in Solvables, with
% appropriate Params and Perms for it to be read or written.
% arguments:
% - Clause must be non-compound (basic) to match: no address, no params,
% no subClauses.
% - Solvables must be in standard form.
% remarks:
% - Should not be called directly; only by oaa_data_matches_solvable.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_data_in_solvable(Cls, [solvable(G1, Params, Perms) | _Rest], _Perm,
  solvable(G1, Params, Perms) ) :-
  ( Cls = (Head :- _Body) ->
    true
  | otherwise ->
    Head = Cls
  ),
  would_unify(Head, G1),
  icl_GetParamValue(synonym(Head, _RealHead), Params),

```

```

% @@DLM: OK, so it's a synonym, but shouldn't we check
% the permissions and type(data) for the referenced solvable?
!.
oaa_data_in_solvable(Clause, [solvable(G1,Params,Perms) | _Rest], Perm,
                    solvable(G1,Params,Perms) ) :-
    icl_GetParamValue(type(data), Params),
    ( Clause = (Head :- _Body) ->
        icl_GetParamValue( rules_ok(true), Params)
    | otherwise ->
        Head = Clause
    ),
    would_unify(Head, G1),
    ( Perm == write ->
        icl_GetPermValue(write(true), Perms)
    | otherwise ->
        icl_GetPermValue(call(true), Perms)
    ),
    !.
oaa_data_in_solvable(Clause, [_|Rest], Perm, Matched) :-
    oaa_data_in_solvable(Clause, Rest, Perm, Matched).

```

```

%*****
% Retrieving and managing events
%*****

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_MainLoop
% purpose:   The main event loop for the application.
%           Reads an event, executes (interprets) it,
%           checks on_receive triggers for the event,
%           checks any application-dependent triggers,
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

oaa_MainLoop(ShouldPrint) :-
    oaa_Ready(ShouldPrint),

    repeat,
        oaa_GetEvent(Event, Params, 0),
        oaa_ProcessEvent(Event, Params),
    fail.

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_ProcessEvent
% purpose:   Interprets an incoming event
%           - For a timeout, checks task triggers and calls user's idle procedure
%           - Otherwise, oaa_Interprets the event, checks on_receive comm
%           triggers, and then checks task triggers.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_ProcessEvent(timeout, _Params) :- !,
    oaa_CheckTriggers(task, _, _), !,
    oaa_call_callback(app_idle, _, []).
oaa_ProcessEvent(Event, Params) :-

```

```

( oaa_Interpret(Event, Params) -> true | true ),
oaa_CheckTriggers(task, _, _), !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_SetTimeout
% purpose:   Sets the timeout value used by oaa_GetEvent
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_SetTimeout(NSecs) :-
    % Make sure NSecs is valid number
    number(NSecs),
    (NSecs < 0 ->
        Timeout = 0
    |   Timeout = NSecs),

    oaa_TraceMsg('-nSetting event timeout to '~q'.'.~n', [Timeout]),
    on_exception(_, retractall(oaa_timeout(_)), true),
    assert(oaa_timeout(Timeout)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_GetEvent
% purpose:   Return the next event to execute
% remarks:
%   - if a oaa_timeout(Secs) is set to a positive real number by
%     oaa_SetTimeout, wait Secs for an event.
%     If none arrives in this time, return Event = `timeout'
%   - Reads ALL events available on communication stream, sorts the events
%     according to priority, chooses the next event to execute,
%     and then saves the rest for next time oaa_GetEvent is called.
%   - The communication stream is read every time oaa_GetEvent is called, even
%     if there are already saved events (a new one might have a higher
%     priority!)
%   - If saved events exist, return immediately (timeout not considered).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_GetEvent(Event, Params, LowestPriority) :-
    % see if previously saved events to process
    ( retract(oaa_event_buffer(SavedEvents)) ->
        true
    |   otherwise ->
        SavedEvents = []
    ),

    % If at least one event can be found with an appropriate priority
    %   from among the saved events, no timeout needed -- flush tcp
    %   buffer, and read_all available
    (oaa_choose_event(LowestPriority, SavedEvents, _OneEvent, _Remainder) ->
        TimeoutSecs = 0.01
    |   on_exception(_, oaa_timeout(TimeoutSecs), TimeoutSecs=0)
    |   TimeoutSecs=0
    ),

    oaa_read_all_events(TimeoutSecs, MoreEvents, FlushPriority),

    % if one of the new events has a flush in it, see if it

```

```

%   flushes any of the saved events
% note: MoreEvents have already been flushed by FlushPriority
oaa_flush_events(SavedEvents, FlushPriority, RemainingSavedEvents),

% These are the events we've read so far and haven't executed yet...
append(RemainingSavedEvents, MoreEvents, EventList),

(oaa_sort_and_get_event(EventList, LowestPriority, Event, Params) ->
  % we are able to find an appropriate event from list
  % The event will be returned, so fire triggers on it
  oaa_CheckTriggers(comm, event(Event, Params), receive)
|
  % no good event found, return timeout
  Event = timeout,
  Params = []
),
% This cut is essential to avoid faulty behavior (DLM):
!.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:   oaa_sort_and_get_event
% purpose: Sort raw events by priority, choose the highest priority event
%         or FirstIn if equal priority, extract event data and sender,
%         and store the rest of events
% remarks:
%         The chosen event must be of HIGHER priority than LowestPriority, and
%         oaa_sort_and_get_event can fail if no appropriate event is found
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_sort_and_get_event(EventList, LowestPriority, Event, Params) :-
  samsort(oaa_priority_compare, EventList, SortedList),
  oaa_choose_event(LowestPriority, SortedList, RawEvent, Remainder),
  oaa_extract_event(RawEvent, Event, Params),
  (Remainder = [] ;
   assert(oaa_event_buffer(Remainder))),
  !.

oaa_priority_compare(E1, E2) :-
  oaa_extract_event_param(E1, _, priority(P1)),
  oaa_extract_event_param(E2, _, priority(P2)),
  !, P1 >= P2.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:   oaa_choose_event
% purpose: Extracts the first event from a list which has a HIGHER priority
%         than the required lowest. Fails if none found.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_choose_event(LowestPriority, [Event|Remainder], Event, Remainder) :-
  oaa_extract_event_param(Event, _, priority(P)),
  LowestPriority < P,
  !.
oaa_choose_event(LowestPriority, [E|Rest], Event, [E|Rest2]) :-
  oaa_choose_event(LowestPriority, Rest, Event, Rest2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

% name:      oaa_read_all_events
% purpose:   Flush the communication event queue, reading ALL available events and
%            returning a list of them, or empty list if none available.
% remarks:
%   - Events are retrieved in raw (unextracted) form.
%   - We check to make sure the event is Validated (security hook)
%     before returning it
%   - We check to see if the event is flushed by a later event.
%     If so, we notify event sender of the flush and we don't return the
%     event.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_read_all_events(Timeout, Events, FlushPriority) :-
    oaa_select_event(Timeout, E), !,
    (E == timeout ->
        Events = [],
        FlushPriority = 0    % lowest event priority: don't flush events
    |
        % read one event, so read all the rest
        oaa_read_all_events(0.0001, RestEvents, RestFlushPriority),

        % check if read Event is acceptable (security hook)
        (oaa_ValidateEvent(E,OkEvent) ->
            oaa_ComTraceMsg('-n[COM received]:-n  -q-n', [OkEvent]),

            % get event's priority
            oaa_extract_event_param(OkEvent, _, priority(P)),

            % if less than some higher priority flush event, discard event
            %   and perhaps notify sender
            (P < RestFlushPriority ->
                % event will be removed,
                oaa_flush_notification(OkEvent),
                FlushPriority = RestFlushPriority,
                Events = RestEvents
            |
                % keep event: not flushed
                Events = [OkEvent|RestEvents],

                % see if this event adds a flush:
                %   if so record new flush priority
                (oaa_event_param(OkEvent, flush_events(true)) ->
                    FlushPriority = P
                | FlushPriority = RestFlushPriority)
            )

        % Not validated, skip event
        | Events = RestEvents)
    ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_ValidateEvent
% purpose:   Check that an incoming lowlevel event should be processed.
%            This is the place to put security checks on events.
%            The default behavior defined by the library can be made more
%            stringent by individual agents using the callback oaa_AppValidateEvent
% remarks:

```

```

%   oaa_ValidateEvent has the right to modify the incoming event,
%   or refuse it altogether by failing.
*****
oaa_ValidateEvent(E,OkEvent) :-
    % if oaa_AppValidateEvent is defined, use it.
    predicate_property(user:oaa_AppValidateProperty(_,_), _),
    !,
    user:oaa_AppValidateProperty(E, OkEvent).
% currently, no security checks are performed
oaa_ValidateEvent(OkEvent,OkEvent).

*****
% name:      oaa_flush_events
% purpose:  Flushes any events with a lower priority than the FlushPriority
*****
oaa_flush_events([], _FlushPriority, []).
oaa_flush_events([Event|RestEvents], FlushPriority, RemainingEvents) :-
    oaa_flush_events(RestEvents, FlushPriority, RestSaved),

    % get event's priority
    oaa_extract_event_param(Event, _, priority(P)),

    % if lower priority than we are flushing, notify and remove
    (P < FlushPriority ->
        oaa_flush_notification(Event),
        RemainingEvents = RestSaved
    |
        RemainingEvents = [Event|RestSaved]
    ).

*****
% name:      oaa_flush_notification
% purpose:  Given a raw event, grabs its real event and looks up whether
%           a notification should be sent out regarding the event's
%           cancellation due to a flush.
*****
oaa_flush_notification(RawEvent) :-
    oaa_extract_event(RawEvent, Event, _Params),
    (oaa_get_flush_notify(Event, NotifyEvent) ->
        oaa_PostEvent(NotifyEvent, [])
    | true), !.

*****
% name:      oaa_get_flush_notify
% purpose:  Records a list of events which require a return notification
%           if the event is flushed.
% remarks:
%           currently, only the ev_solve() event returns a message;
%           all other events are flushed without notification
*****
% @@Additional entries needed here:
oaa_get_flush_notify(ev_solve(ID, Goal, Params),
    ev_solved(ID, FromMe, Goal, Params, [])) :-

```



```

        (icl_GetParamValue(reply(none), Params) ->
         fail
         | oaa_Id(FromMe)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_select_event
% purpose:  If a positive timeout is defined, wait N seconds for an event
%           to arrive
%           Otherwise block-wait until an event arrives.
% remarks:  IMPORTANT: Connected/1 gets special handling, because we want
%           the connection ID and oaa ID to be assigned immediately.
%           Otherwise, oaa_translate_incoming_event and oaa_unwrap_event
%           won't always work properly for subsequent events from the
%           new connection (or would have to be more complicated).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_select_event(TimeOut, Event) :-
    com:com_SelectEvent(TimeOut, InEvent),
    ( InEvent = connected(_) ->
      oaa_ProcessEvent(InEvent, []),
      oaa_select_event(TimeOut, Event)
    | otherwise ->
      oaa_translate_incoming_event(InEvent, TranslatedEvent),
      oaa_unwrap_event(TranslatedEvent, _Connection, Event)
    ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_unwrap_event(+TranslatedEvent, -Connection, -Event).
% arguments: TranslatedEvent: An event from another agent, which has already
%           been translated for version compatibility, if necessary.
%           Event: An event term in our standard internal format, as required
%           by all other library procedures.
%           Connection: The CONNECTION of the immediate agent
%           from which this message came (note that an agent's CONNECTION
%           can be different than its ID).
% purpose:  Remove an event term from its communications wrapper (if any),
%           and returns it in our standard internal form:
%           'timeout' OR event(Content, Params).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% timeout is the ONLY event that doesn't get embedded in event/2:
oaa_unwrap_event(timeout, unknown, timeout) :-
    !.
oaa_unwrap_event(term(Connection, event(Content, Params)), ConnectionId,
    event(Content, NewParams)) :-
    !,
    ( com:com_GetInfo(ConnectionId, connection(Connection)) ->
      true
    | otherwise ->
      format(
        '-w: incoming event from an unrecognized connection (-w):-n -w-n',
        ['INTERNAL ERROR', Connection, event(Content, Params)]),
      ConnectionId = unknown
    ),
    ( memberchk(from(_), Params) ->
      NewParams = [connection_id(ConnectionId) | Params]
    | Content = ev_connected(InfoList),

```

```

memberchk(fac_id(Id), InfoList) ->
    NewParams = [from(Id), connection_id(ConnectionId) | Params]
| ConnectionId = parent,
    com:com_GetInfo(ConnectionId, fac_id(Id)) ->
    NewParams = [from(Id), connection_id(ConnectionId) | Params]
| com:com_GetInfo(ConnectionId, oaa_id(Id)) ->
    NewParams = [from(Id), connection_id(ConnectionId) | Params]
| otherwise ->
    % With current code, this should never happen. But I can
    % imagine code changes that might need this (DLM 98/02/18):
    NewParams = [from(unknown), connection_id(ConnectionId) | Params]
).

% This handles connected/1, end_of_file/1, wakeup/1:
oaa_unwrap_event(Content, unknown, event(Content, [])).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_translate_incoming_event(+InEvent, -OutEvent).
% purpose: Provides backwards compatibility by calling a hook
%          (user:oaa_event_translation/7) that translates incoming events from agents
of
%          other versions. Also allows for event differences based on language.
%          The idea is to return an event with both format and contents that
%          are appropriate for the agent receiving the event.
% remarks: user:oaa_event_translation/7 can be hard-coded, loaded at runtime,
%          or whatever. If it's not present, we return the same event.
%          Note that the translation hook is somewhat limited. It allows a single
%          event to be translated to another single event, and with essentially
%          no information about context. This inadequate or awkward for some cases.
%          Those cases are handled using extra clauses of user:oaa_AppDoEvent (in
%          translations.pl).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Special cases. There's no need to translate these. And, it could be
% problematical, because we don't yet know the language and version of
% the sender.
oaa_translate_incoming_event(term(Conn, event(Contents, Params)),
    term(Conn, event(Contents, Params))) :-
    ( Contents = ev_connect(_);
      Contents = ev_connected(_) ),
    !.

oaa_translate_incoming_event(term(Connection, InEvent),
    term(Connection, OutEvent)) :-
    current_predicate(oaa_event_translation,
        user:oaa_event_translation(_,'_','_','_','_','_')),
    ( com:com_GetInfo(ConnectionId, connection(Connection)) ->
      true
    | otherwise ->
      true
    ),
    % These assumptions may not always be right, but will
    % nearly always get the desired results.
    % :
    ( ground(ConnectionId),

```

```

        com:com_GetInfo(ConnectionId, agent_version(PriorVersion)) ->
            true
    | otherwise ->
        PriorVersion = 2.1
    ),
    ( ground(ConnectionId),
      com:com_GetInfo(ConnectionId, agent_language(PriorLanguage)) ->
          true
    | otherwise ->
        PriorLanguage = c
    ),
    oaa_LibraryVersion(MyVersion),
    ( MyVersion \== PriorVersion ; PriorLanguage \== prolog ),
    user:oaa_event_translation(PriorVersion, PriorLanguage, MyVersion, prolog,
                               Connection, InEvent, OutEvent),
    !.
% This handles timeout/0, connected/1, end_of_file/1, wakeup/1.
% Also passes through any event for which there is no translation.
oaa_translate_incoming_event(Event, Event) :- !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_extract_event
% purpose:   Extract the content and parameters from an event term.
% remarks:   Always succeeds.
%           The content part of the term is often (loosely) called the Event.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

oaa_extract_event(event(Content, Params), Content, Params) :-
    !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_extract_event_param
% purpose:   Extract the content and a parameter value from an event term.
% remarks:   Always succeeds - unless you ask for a param that has no default
%           value.
%           The content part of the term is often (loosely) called the Event.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

oaa_extract_event_param(event(Content, Params), Content, Param) :- !,
    icl_GetParamValue(Param, Params).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_event_param
% purpose:   Extract a parameter from an event term.
% remarks:   This FAILS if the parameter isn't present (unlike
%           oaa_extract_event_param).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

oaa_event_param(event(_Content, Params), Param) :- !,
    memberchk(Param, Params).

%*****
% Interpreting EVENTS
%*****

```

```

*****
% name:    oaa_Interpret(+ICLEExpression, +Params)
% purpose: Executes an incoming event
% remarks: Implements a simple meta-interpreter for executing complex goals.
%          Agent goals are interpreted by oaa_exec_event().
%
%          The contents of Params will vary depending on context.
%          When oaa_Interpret is called on an incoming event, Params
%          will (usually) include from(Sender). Calls generated internally
%          may contain from(self). Additional params may
%          accumulate through recursive calls to oaa_Interpret.

*****
oaa_Interpret(Goal, _) :- var(Goal), !, fail. % How could this happen?
oaa_Interpret(true, _) :- !.
oaa_Interpret(false, _) :- !, fail.
oaa_Interpret(false, _) :- !, fail.
oaa_Interpret((\+ P), Params) :- !, \+ oaa_Interpret(P, Params).
oaa_Interpret((P -> Q ; _R), Params) :-
    oaa_Interpret(P, Params), !, oaa_Interpret(Q, Params).
oaa_Interpret((_P -> _Q ; R), Params) :- !, oaa_Interpret(R, Params).
oaa_Interpret((P -> Q), Params) :- !, oaa_Interpret((P -> Q ; fail), Params).
oaa_Interpret((X, Y), Params) :- !,
    oaa_Interpret(X, Params), oaa_Interpret(Y, Params).
oaa_Interpret((X ; Y), Params) :- !,
    (oaa_Interpret(X, Params) ; oaa_Interpret(Y, Params)).
oaa_Interpret(findall(Var, Goal, All), Params) :- !,
    findall(Var, oaa_Interpret(Goal, Params), All).
oaa_Interpret(P, _Params) :- icl_BuiltIn(P), !, call(P).
oaa_Interpret(X, Params) :- oaa_exec_event(X, Params).

*****
% name:    oaa_exec_event
% purpose: Defines execution of events built into all agents
% remarks: Goals that can't be handled by oaa_exec_event are passed to the
%          user-declared app_do_event callback, if present.
*****
% turn on trace
oaa_exec_event(ev_trace_on, _) :-
    abolish(oaa_trace/1),
    assert(oaa_trace(on)),
    format('~nTrace on.~n', []), !.

% turn off trace
oaa_exec_event(ev_trace_off, _) :-
    abolish(oaa_trace/1),
    assert(oaa_trace(off)),
    format('~nTrace off.~n', []), !.

% tcp level trace
oaa_exec_event(ev_com_trace_on, _) :-
    abolish(oaa_com_trace/1),
    assert(oaa_com_trace(on)),
    format('~nCOMMUNICATION PROTOCOL trace on.~n', []), !.

% tcp level trace

```

```

oaa_exec_event(ev_com_trace_off, _) :-
    abolish(oaa_com_trace/1),
    assert(oaa_com_trace(off)),
    format('~nCOMMUNICATION PROTOCOL trace off.~n', []), !.

% turn on debug
oaa_exec_event(ev_debug_on, _) :-
    abolish(oaa_debug/1),
    assert(oaa_debug(on)),
    format('~nDebug on.~n', []), !.

% turn off debug
oaa_exec_event(ev_debug_off, _) :-
    abolish(oaa_debug/1),
    assert(oaa_debug(off)),
    format('~nDebug off.~n', []), !.

% Set the timeout value
oaa_exec_event(ev_set_timeout(N), _) :-
    abolish(timeout/1),
    assert(timeout(N)),
    format('~nTimeout set to ~q.~n', [N]), !.

% Notification that some other agent has disconnected. Currently, this applies
% only to peer client agents, and the arg. will always be a local ID.
oaa_exec_event(ev_agent_disconnected(LID), _) :-
    oaa_remove_data_owned_by(LID).

% quit to UNIX
oaa_exec_event(ev_halt, _) :-
    format('~nDisconnecting...~n', []),
    com:com_Disconnect(parent),
    ( oaa_call_callback(app_done, _, []) ; true ),
    halt.

oaa_exec_event(ev_update(ID, Mode, Clause, Params), EvParams) :-
    oaa_Id(AgentId),
    append(Params, EvParams, AllParams),
    ( Mode = add ->
        Functor = oaa_add_data_local
    | Mode = remove ->
        Functor = oaa_remove_data_local
    | Mode = replace ->
        Functor = oaa_replace_data_local
    ),
    Call =.. [Functor, Clause, AllParams],
    ( call(Call) ->
        Updaters = [AgentId]
    | otherwise ->
        Updaters = []
    ),
    (icl_GetParamValue(reply(none), AllParams) -> true |
        oaa_PostEvent(ev_updated(ID, Mode, Clause, Params, Updaters),
            []))
    ).

```

```

% add or remove a local trigger
oaa_exec_event(ev_update_trigger(ID, Mode, Type,
                                Condition, Action, TrigParams),
              Params) :-
    oaa_Id(AgentId),
    append(TrigParams, Params, NewParams),
    ( Mode == add ->
      Functor = oaa_add_trigger_local
    | Mode == remove ->
      Functor = oaa_remove_trigger_local
    ),
    Call =.. [Functor, Type, Condition, Action, NewParams],
    ( call(Call) ->
      Updaters = [AgentId]
    | otherwise ->
      Updaters = []
    ),
    ( icl_GetParamValue(reply(none), Params) ->
      true
    | otherwise ->
      oaa_PostEvent(ev_trigger_updated(ID, Mode, Type, Condition,
                                       Action, TrigParams, Updaters),
                    []
      ),
    ( Mode = add ->
      oaa_Inform(trigger, 'trigger_added(-q,-q,-q,-q)-n',
                 [Type, Condition, Action, NewParams])
    | true
    ).

% When asked to solve a goal, see if you know how to solve
% it, then find all solutions.  Send the solutions to the
% caller.
%
% The various params lists must be used with care.  Searching different
% lists may be appropriate for different params, depending on their
% meanings.  Another consideration is that Solve params and Goal params,
% as returned to the requesting agent, must unify with the original
% lists that came from the requesting agent.

oaa_exec_event(ev_solve(ID, FullGoal, SolveParams), Params) :-
    oaa_class(leaf),
    icl_GoalComponents(FullGoal, _, _, GoalParams),

    % More "local" params take precedence, so they go to the
    % beginning of the list:
    append([SolveParams, Params], InheritedParams),
    append([GoalParams, InheritedParams], AllParams),
    % Assert context:
    findall(context(C), member(context(C), AllParams), Contexts),
    asserta( oaa_current_contexts(ID, Contexts) ),

    oaa_TraceMsg('-n-nAttempting to solve:-n Goal:-q-n Params:-q-n',
                [FullGoal, InheritedParams]),
    findall(FullGoal,
            oaa_solve_local(FullGoal, InheritedParams),
            Solutions),

```

```

    oaa_TraceMsg('-nSolutions found for -q:-n    -q-n',
                [FullGoal, Solutions]),

% If user has requested to delay the solution (oaaDelaySolution)
% save current UserId, Goal and Params in delay table, to be
% sent back in an ev_solved() msg later (oaaReturnDelayedSolutions).

(retract(oaa_delay(ID, UserId)) ->
  assert(oaa_delay_table(ID, UserId, FullGoal, SolveParams, AllParams))
  |
  (icl_GetParamValue(reply(none), AllParams) -> true |
    (oaa_Id(FromKS) ; FromKS = unknown), !,
    oaa_PostEvent(ev_solved(ID, FromKS, FullGoal, SolveParams,
                          Solutions), []))
  )
),

% Retract context:
retractall( oaa_current_contexts(ID, _) ).

% This is for subgoals (of goals passed in solve events) that have
% Params.  Subgoals with no params will fall through to the next clause.
oaa_exec_event(Goal::GoalParams, Params) :-
  oaa_solve_local(Goal::GoalParams, Params).

% call user events.  Must not have a cut, to return all solutions.
oaa_exec_event(Event, Params) :-
  oaa_turn_on_debug,
  ( oaa_solvable(Solvables) -> true | otherwise -> Solvables = []),
  ( (oaa_goal_matches_solvable(Event, Solvables, Goal, Matched),
    Matched = solvable(_, SolvParams, _),
    (icl_GetParamValue(callback(CB), SolvParams) ;
    oaa_callback(app_do_event, CB)))
  ;
  (oaa_callback(app_do_event, CB),
  Goal = Event)
),
!,
( CB = Module:Functor ->
  true
| otherwise ->
  Module = user,
  Functor = CB
),
Call =.. [Functor, Goal, Params],
on_exception(E,
  Module:Call,
  ( oaa_TraceMsg('WARNING (agent.pl): Exception raised thru callback
handler (-w):-n    -q-n',
                [Functor, E]),
  fail )),
  oaa_turn_off_debug.

% What to do about test(TEST)?
% if test(TEST) is listed in arguments, solve

```

```

%   it locally.
passes_tests(Params) :-
    oaa_class(leaf),
    icl_GetParamValue(test(Test), Params),
    !,
    oaa_Solve(Test, [level_limit(0)]).
% With compound goals, we also want to allow tests on the facilitator.
% @@DLM: Is this the best way?
passes_tests(Params) :-
    (oaa_class(root);oaa_class(node)),
    icl_GetParamValue(test(Test), Params),
    !,
    oaa_solve_local(Test, []).
passes_tests(_Params) :-
    true.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_DelaySolution
% purpose:   Requests that the current AppDoEvent not return solutions to the
%           current goal until a later time.
% inputs:
%   - Id: an Id which will be used to later match solutions to request
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_DelaySolution(Id) :-
    oaa_current_contexts(GoalId, _Contexts), !,
    assert(oaa_delay(GoalId, Id)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_ReturnDelayedSolutions
% purpose:   Returns the list of solutions for a delayed request
% inputs:
%   - Id: an Id referring to a previously saved oaa_DelaySolution
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_ReturnDelayedSolutions(Id, SolutionList) :-
    (retract(oaa_delay_table(GoalId, Id, Goal, SolveParams, AllParams)) ->
        (icl_GetParamValue(reply(none), AllParams) -> true |
            (oaa_Id(FromKS) ; FromKS = unknown), !,
            % make sure all Solutions unify with original goal
            findall(Goal, member(Goal, SolutionList), Solutions),
            oaa_PostEvent(ev_solved(GoalId, FromKS, Goal, SolveParams,
                Solutions), []))
        )
    | true).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_AddDelayedContextParams
% purpose:   When a goal is delayed using oaa_DelaySolution(), incoming context
%           parameters from the original request can not be automatically
%           concatenated to outgoing oaa_Solve requests -- since an agent can
%           manage multiple delayed goals at the same time, liboaa doesn't
%           know the correct context for the outgoing oaa_Solve without explicit
%           direction from the programmer. Hence, an agent programmer who
%           wants to call oaa_Solve during a delayed goal is expected to
%           use this function to add the saved contexts for the delayed goal to

```



```

%      his/her outgoing oaa_Solve parameters.
% inputs:
%   - Id: an Id which will be used to later match solutions to request
%   - Params: Parameters for solve goal
%   - NewParams: Params augmented by saved contexts.
% example:
%   oaa_AppDoEvent(goal(_X),_Params) :- oaa_DelayEvent(a_goal).
%   oaa_AppDoEvent(temp_event(Y),_Params) :-
%       oaa_AddDelayedContextParams(a_goal, [], P),
%       oaa_Solve(sub_goal(Y), P).
%   oaa_AppDoEvent(final_event(S),_Params) :-
%       oaa_ReturnDelayedSolutions(a_goal, [goal(S)]).
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_AddDelayedContextParams(Id, Params, NewParams) :-
    retract(oaa_delay_table(_GoalId, Id, _Goal, _SolveParams, AllParams)),
    findall(context(C), member(context(C), AllParams), Contexts),
    append(Contexts, Params, NewParams).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%*****
% Agent-Facilitator communication
%*****

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_PostEvent
% purpose:   Sends a low-level event to another agent
% remarks:
%   Should NOT be used before there's a connection established for
%   the destination (such as when a client sends ev_connect to its
%   facilitator). In such unusual cases, use com_SendData directly.
%   For application developers, this just means don't call
%   oaa_PostEvent until after you've called oaa_Register.
% Parameters may include:
%   - priority(P):
%   - address(A): specify address of specific server or client agent
%     A must be an agent ID, not a name. If caller is a client agent,
%     the only meaningful address is that of the client's facilitator.
%   - from(KS): where the event originally originated
% IMPORTANT: there may be a different address INSIDE the event;
% these should not be confused!
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_PostEvent(Contents, Params) :-

    % see if any params of interest
    (memberchk(priority(_P), Params);
     memberchk(from(_Agent), Params) ->
        SendEvent = event(Contents, Params)
    |
        SendEvent = event(Contents, []))
    ),

    % find destination: if none, dest = server
    (memberchk(address(Dest), Params) ->
        true

```

```

|
  Dest = parent
),

icl_true_id(Dest, DestId),
  oaa_translate_outgoing_event(SendEvent, DestId, TransEvent),

oaa_ComTraceMsg('-n[COM send to -q]:-n  -q-n', [Dest, TransEvent]),

oaa_convert_id_to_comm_id(DestId, CommId),
% send event to destination
com:com_SendData(CommId, TransEvent),

% Use SendEvent here, because triggers always contain event/2
% to unify with.
  oaa_CheckTriggers(comm, SendEvent, send).

oaa_convert_id_to_comm_id(Id, CId) :-
  com:com_GetInfo(CId, fac_id(Id)), !.
oaa_convert_id_to_comm_id(Id, CId) :-
  com:com_GetInfo(CId, oaa_id(Id)), !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_translate_outgoing_event(+Event, +DestId, -NewEvent).
% purpose: Provides backwards compatibility by calling a hook
%           (user:oaa_event_translation/7) that translates outgoing events to agents of
%           other versions. Also allows for event differences based on language.
% remarks: user:oaa_event_translation/7 can be hard-coded, loaded at runtime,
%           or whatever. If it's not present, we return the same event.
%           See also comments for oaa_translate_incoming_event.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Special cases. There's no need to translate these. And, it could be
% problematical, because we don't yet know the language and version of
% the receiver. See comments for oaa_unwrap_event.
oaa_translate_outgoing_event(event(Contents, Params), _DestId,
  event(Contents, Params)) :-
  ( Contents = ev_connect(_) ;
    Contents = ev_connected(_) ),
  !.
oaa_translate_outgoing_event(event(Content, Params), DestId, TransEvent) :-
  current_predicate(oaa_event_translation,
    user:oaa_event_translation(_,_,_,_,_,_)),
  % These assumptions may not always be right, but will
  % nearly always get the desired results:
  com:com_GetInfo(Connection, oaa_id(DestId)),
  ( com:com_GetInfo(Connection, agent_version(DestVersion)) ->
    true
  | otherwise ->
    DestVersion = 2.1
  ),
  ( com:com_GetInfo(Connection, agent_language(DestLanguage)) ->
    true
  | otherwise ->
    DestLanguage = c

```

```

),
  oaa_LibraryVersion(MyVersion),
  user:oaa_event_translation(MyVersion, prolog, DestVersion, DestLanguage,
                             Connection, event(Content, Params), TransEvent),
!.
oaa_translate_outgoing_event(Event, _, Event).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_Version
% purpose:   Lookup the language and library version number for an agent
% remarks:   The default version (if unspecified) is 1.0
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

oaa_Version(AgentId, Language, Version) :-
  icl_true_id(AgentId, TrueId),
  % Asking for my version:
  oaa_Id(TrueId),
  Language = prolog,
  oaa_LibraryVersion(Version),
  !.
oaa_Version(AgentId, Language, Version) :-
  icl_true_id(AgentId, TrueId),
  ( com:com_GetInfo(CommId, oaa_id(TrueId)) ;
    com:com_GetInfo(CommId, fac_id(TrueId)) ),
  ( com:com_GetInfo(CommId, agent_language(Language)) ->
    true
  | otherwise ->
    Language = unknown
  ),
  ( com:com_GetInfo(CommId, agent_version(Version)) ->
    true
  | otherwise ->
    Version = 1.0
  ),
  !.
oaa_Version(AgentId, Language, Version) :-
  (oaa_class(leaf) ; oaa_class(node)),
  icl_true_id(AgentId, TrueId),
  % The use of caching here could be dangerous - unless we install a
  % mechanism for automatic updating of the cache.
  oaa_Solve(agent_version(TrueId, Language, Version),
            [address(parent)]),
  !.
oaa_Version(_, prolog, 1.0).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_CanSolve
% purpose:   Asks the Facilitator for a list of agents which could solve a Goal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_CanSolve(Goal, KSLList) :-
  oaa_Solve(can_solve(Goal, KSLList), [address(parent)]).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_Ping

```

```

% purpose: Tests whether a given agent is currently responding to requests.
% inputs:
%   AgentAddr: address of agent to test
%   TimeLimit: Time limit (in seconds) for how long to wait for a response
% outputs:
%   TotalResponseTime for round trip (in seconds)
% remarks: Fails if a ping is not returned in TimeLimit amount of time
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_Ping(AgentAddr, TimeLimit, TotalResponseTime) :-
    ground(AgentAddr),
    number(TimeLimit),
    TimeLimit >= 0,
    tcp_now(Before),
    oaa_Solve(true, [address(AgentAddr), time_limit(TimeLimit)]),
    tcp_now(After),
    tcp_time_plus(Before, TotalResponseTimeMs, After),
    TotalResponseTime is TotalResponseTimeMs / 1000.

```

```

%*****
% Declaring Solvables
%*****

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_Declare(+Solvables, +CommonPermissions, +CommonParams, +Params,
%                  -DeclaredSolvables)
% purpose: Declare solvables for a client or facilitator, and inform the
%          parent if appropriate.
% arguments:
%   Solvables: A single solvable or a list of solvables, in shorthand or
%              standard form.
%   CommonPermissions: Permissions to be distributed to each solvable in
%                      Solvables. This is purely for programming convenience. See
%                      comments for icl_ConvertSolvables for possible values, and
%                      solvables documentation for their meanings.
%   CommonParams: Params to be distributed to each solvable in Solvables.
%                 This is purely for programming convenience. See comments for
%                 icl_ConvertSolvables for possible values, and solvables
%                 documentation for their meanings.
%   Params:
%     address(X): Where the solvable will exist. X may be either 'self'
%               or 'parent' (or the appropriate local ids). Default: 'self'.
%     if_exists(OverwriteOrAppend): What to do when declaring solvables
%                                   for self, and some already exist. Default: append.
%   DeclaredSolvables: Returns a list, in standard form, of all solvables
%                       successfully declared.
% remarks:
%   - Any agent can declare solvables for itself. In addition, a client can
%     ask its facilitator to declare solvables. Client-requested facilitator
%     solvables will automatically acquire permission write(true), and params
%     type(data), rules_ok(false), private(false), and bookkeeping(true).
%   - If called by a leaf or node agent, assumes agent is already registered
%     with a parent facilitator.
%   - Predicates can only be declared once. Changing an existing
%     predicate definition should be done with oaa_Redeclare. However,

```

```

%      a request to declare a predicate, which is already declared in
%      precisely the same way, succeeds transparently.
%      - @@Future params may include 'num_context_args(N)'.
%      - @@Future solvable params may include 'shared'.
%      - synonym predicates can have their own triggers, but share the clause
%      database with their master table.
%      - views and filters, as provided by the OAA V1 DB agent, are not
%      supported as separate params, but the same functionality is available
%      using other params.
%      - @@Do we want client agents to request declarations on other client
%      agents?
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_Declare(Solvable, InitialCommonPerms, InitialCommonParams,
            InitialParams, DeclaredSolvables) :-
    ( is_list(Solvable) ->
      SolvableList = Solvable
    | otherwise ->
      SolvableList = [Solvable]
    ),
    icl_ConvertSolvables(SolvableList, Solvables),
    icl_standardize_perms(InitialCommonPerms, false, CommonPerms),
    icl_standardize_params(InitialCommonParams, false, CommonParams),
    icl_standardize_params(InitialParams, false, Params),
    oaa_distribute_perms(Solvables, CommonPerms, Solvables1),
    oaa_distribute_params(Solvables1, CommonParams, NewSolvables),
    oaa_declare_aux(add, NewSolvables, Params, DeclaredSolvables).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_DeclareData(+Solvables, +Params, -DeclaredSolvables)
% purpose:   Declare data solvables for an agent.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_DeclareData(Solv, Params, DeclaredSolvs) :-
    \+ is_list(Solv),
    !,
    oaa_DeclareData([Solv], Params, DeclaredSolvs).
oaa_DeclareData(Solv, Params, DeclaredSolvs) :-
    % It's only necessary to specify the non-default perms and params.
    CommonPerms = [write(true)],
    CommonParams = [type(data)],
    oaa_Declare(Solv, CommonPerms, CommonParams, Params, DeclaredSolvs).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_Undeclare(+Solvables, +Params, -UndeclaredSolvables)
% purpose:   Remove solvables from a client or facilitator, and inform the
%           parent if appropriate.
% arguments:
%   Solvables: A single solvable or a list of solvables, in shorthand or
%             standard form. If a solvable is in standard form, however, ONLY
%             the goal is considered in selecting the solvables to be removed
%             (permissions and parameters are ignored).
%   Params:
%     address(X): Where the solvable exists. X may be either 'self'
%               or 'parent' (or the appropriate local ids). Default: 'self'.
%   DeclaredSolvables: Returns a list, in standard form, of all solvables
%                     successfully removed.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

% remarks:
%   - If called by a leaf or node agent, assumes agent is already registered
%     with a parent facilitator.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_Undeclare(Solvable, InitialParams, UndeclaredSolvables) :-
    ( is_list(Solvable) ->
      SolvableList = Solvable
    | otherwise ->
      SolvableList = [Solvable]
    ),
    icl_minimally_instantiate_solvables(SolvableList, Solvables),
    icl_standardize_params(InitialParams, false, Params),
    oaa_declare_aux(remove, Solvables, Params, UndeclaredSolvables).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_Redeclare(+Solvable, +NewSolvable, +Params)
% purpose:   Replace a solvable on a client or facilitator, and inform the
%           parent if appropriate.
% arguments:
%   Solvable: A single solvable, in shorthand or standard form.  If in
%             standard form, however, ONLY the goal is considered in selecting
%             the solvable to be replaced (permissions and parameters are ignored).
%   NewSolvable: A single solvable, in shorthand or standard form.
%   Params:
%     address(X): Where the solvable exists.  X may be either 'self'
%               or 'parent' (or the appropriate local ids).  Default: 'self'.
% remarks:
%   - If called by a leaf or node agent, assumes agent is already registered
%     with a parent facilitator.
%   - FAILS if the operation cannot be completed.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_Redeclare(InitialSolvable, InitialNewSolvable, InitialParams) :-
    icl_minimally_instantiate_solvables([InitialSolvable], [Solvable]),
    icl_ConvertSolvables([InitialNewSolvable], [NewSolvable]),
    icl_standardize_params(InitialParams, false, Params),
    oaa_declare_aux(replace, Solvable, [with(NewSolvable) | Params],
                    RedeclaredSolvables),
    RedeclaredSolvables \== [].

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_declare_aux(+Mode, +Solvables, +Params, -DeclaredSolvables)
% purpose:   Common code for oaa_Declare, oaa_Undeclare, oaa_Redeclare.
% Mode:     add, remove, or replace.
% Solvables: for Mode = add, a list of Solvables in standard form.
%            for Mode = remove, a list of Solvables in "minimally instantiated"
%            form.
%            for Mode = replace, a list containing a single Solvable, in
%            "minimally instantiated" form.
% Params:   whatever is appropriate for oaa_Declare, _Undeclare, _Redeclare.
%           Must already be in standard form.
% DeclaredSolvables: A list of all solvables successfully added (or removed
%                   or replaced), in standard form.
% remarks:
%   A number of params and perms are required when requesting that a
%   parent declare solvables (see comments for oaa_Declare).  We could ensure

```

```

% their presence here, but it's not essential, because the facilitator will
% enforce this.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Here, a client is asking the facilitator to add, remove, or replace
% solvables.
oaa_declare_aux(Mode, Solvables, Params, DeclaredSolvables) :-
    com:com_GetInfo(parent, fac_id(ParentId)),
    memberchk(address([ParentId]), Params),
    !,
    % Send the request to the Facilitator
    oaa_PostEvent(ev_post_declare(Mode, Solvables, Params), []),
    oaa_poll_until_event(
        ev_reply_declared(Mode, Solvables, Params, DeclaredSolvables)).

% Leaf, node or root adding, removing or replacing its own solvables:
oaa_declare_aux(Mode, Solvables, Params, DeclaredSolvables) :-
    oaa_Id(Me),
    ( memberchk(address(Addr), Params) ->
        Addr = [Me]
    | true),
    !,

    oaa_declare_local(Mode, Solvables, Params, DeclaredSolvables),

    % If I'm a facilitator, I must also "register" my Solvables with myself.
    % (If I'm a node, this will also register them with my parent.)
    ( (\+ oaa_class(leaf), DeclaredSolvables \== []) ->
        oaa_Name(MyName),
        user:oaa_AppDoEvent(
            ev_register_solvables(Mode, DeclaredSolvables, MyName, Params),
            [from(Me)])
    | true
    ),

    % If I'm a leaf, post public solvables to parent facilitator:
    select_elements(DeclaredSolvables, oaa_public_solvable, PublicSolvables),
    ( (oaa_class(leaf), PublicSolvables \== []) ->
        com:com_GetInfo(parent, oaa_name(MyNameC)),
        oaa_PostEvent(
            ev_register_solvables(Mode, PublicSolvables, MyNameC, Params),
            [])
    | true ).

% Solvable must be in standard form.
oaa_public_solvable(solvable(_Solvable, Params, _Perms)) :-
    icl_GetParamValue(private(false), Params).

% Solvable must be in standard form.
oaa_data_solvable(solvable(_Solvable, Params, _Perms)) :-
    icl_GetParamValue(type(data), Params).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_declare_local(+Mode, +Solvables, +Params, -DeclaredSolvables)
% purpose:   Declare solvables for an agent.
% Mode:     add, remove, or replace.
% Solvables: The form they're in depends on the mode.  See oaa_declare_aux.

```

```

% DeclaredSolvables: Returns those members of Solvables for which
%   the operation was successful (more specifically, those that should
%   be passed up to the parent in ev_register_solvables). Always returned
%   in STANDARD FORM.
% Also see:  comments for oaa_Declare, oaa_Undeclare, oaa_Redeclare.
% remarks:
%   - This performs the local processing needed by calls to oaa_Declare,
%     and by ev_declare events.
%   - Solvables and Params must already be in standard form.
%
%   @@DLM: Could do more careful testing to be sure the solvables are
%   all valid for the requested operation.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_declare_local(Mode, Solvable, Params, DeclaredSolvables) :-
    \+ is_list(Solvable),
    !,
    oaa_declare_local(Mode, [Solvable], Params, DeclaredSolvables).
oaa_declare_local(add, InitialSolvables, Params, DeclaredSolvables) :-
    ( icl_GetParamValue(if_exists(overwrite), Params) ->
      CurrentSolvables = []
    | oaa_solvables(CurrentSolvables) ->
      true
    | CurrentSolvables = []
    ),
    % This will eliminate those that unify with an already declared solvable.
    % @@DLM: Should do more, though: warnings.
    solvables_to_be_added(InitialSolvables, CurrentSolvables,
                          DeclaredSolvables),

    % Make sure Quintus has the correct properties for each DB solvable.
    select_elements(DeclaredSolvables, oaa_data_solvable, DBSolvables),
    oaa_declare_for_prolog(DBSolvables),

    append(CurrentSolvables, DeclaredSolvables, AllSolvables),
    retractall(oaa_solvables(_)),
    assert(oaa_solvables(AllSolvables)).

oaa_declare_local(remove, Solvables, _Params, RemovedSolvables) :-
    % See which ones are really declared:
    ( oaa_solvables(Current) -> true | Current = [] ),
    solvables_to_be_removed(Solvables, Current, RemovedSolvables),
    % Retract all clauses from data solvables:
    select_elements(RemovedSolvables, oaa_data_solvable, DBSolvables),
    oaa_remove_solvables_data(DBSolvables),
    % Assert the new solvables list:
    retractall(oaa_solvables(_)),
    subtract(Current, RemovedSolvables, New),
    assert(oaa_solvables(New)).

oaa_declare_local(replace, [Solvable], Params, [Solvable]) :-
    memberchk(with(NewSolvable), Params),
    % Make sure Solvable is really declared:
    ( oaa_solvables(Current) -> true | otherwise -> Current = []),
    memberchk(Solvable, Current),
    !,
    % If a data solvable, maybe retract all its clauses:
    ( oaa_data_solvable(Solvable) ->

```



```

        oaa_remove_solvable_data([Solvable])
    | true
    ),
    % Assert the new solvables list:
    retractall(oaa_solvable(_)),
    replace_element(Solvable, Current, NewSolvable, New),
    assert(oaa_solvable(New)).
oaa_declare_local(replace, [Solvable], _Params, []) :-
    Solvable = solvable(Goal, _, _),
    format('-w: Ignoring attempt to replace a non-existent solvable:-n -w-n',
        ['WARNING', Goal]).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name: oaa_distribute_params(+Solvables, +CommonParams, -NewSolvables).
%       oaa_distribute_perms(+Solvables, +CommonPerms, -NewSolvables).
% purpose: Add CommonParams (CommonPerms) to the Params (Permissions) list of
%         each solvable in Solvables.
% Solvables: a solvables list, in standard form.
% remarks: @@Should warn when a solvables has a param that conflicts with
%         CommonParams. Also, should have an arg that says which version of
%         of the conflicting param to keep.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

oaa_distribute_params([], _CommonParams, []).
oaa_distribute_params([Solvable | Solvables], CommonParams,
    [NewSolvable | NewSolvables]) :-
    Solvable = solvable(Goal, Params, Perms),
    union(Params, CommonParams, NewParams),
    NewSolvable = solvable(Goal, NewParams, Perms),
    oaa_distribute_params(Solvables, CommonParams, NewSolvables).

```

```

oaa_distribute_perms([], _CommonPerms, []).
oaa_distribute_perms([Solvable | Solvables], CommonPerms,
    [NewSolvable | NewSolvables]) :-
    Solvable = solvable(Goal, Params, Perms),
    union(Perms, CommonPerms, NewPerms),
    NewSolvable = solvable(Goal, Params, NewPerms),
    oaa_distribute_perms(Solvables, CommonPerms, NewSolvables).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name: solvables_to_be_added(+ProposedSolvs, +CurrentSolvs, -SolvsToBeAdded).
% purpose: Checks a list of solvables, to make sure they can legally be
%         declared.
% ProposedSolvs: Must be in STANDARD FORM.
% CurrentSolvs: This agent's current solvables.
% SolvsToBeAdded: A subset of ProposedSolvs.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

solvables_to_be_added([], _Current, []).
solvables_to_be_added([Solvable | Solvables], Current, OKSolvables) :-
    Solvable = solvable(Goal, _, _),
    memberchk(solvable(Goal, _, _), Current),
    !,
    format('-w: Ignoring attempt to declare an already existing solvable:-n
-w-n',
        ['WARNING', Goal]),
    solvables_to_be_added(Solvables, Current, OKSolvables).

```

```

solvable_to_be_added([Solvable | Solvables], Current,
                    [Solvable | OKSolvables]) :-
    solvable_to_be_added(Solvables, Current, OKSolvables).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name: solvable_to_be_removed(+ProposedSolvs, +CurrentSolvs,
%                               -SolvsToBeRemoved).
% purpose: Checks a list of solvables, to make sure they can legally be
%          UNdeclared.
% ProposedSolvs: Must be in MINIMALLY INSTANTIATED FORM.
% CurrentSolvs: This agent's current solvables.
% SolvsToBeRemoved: A subset of ProposedSolvs, but returned in standard form,
%                  fully instantiated.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
solvable_to_be_removed([], _Current, []).
solvable_to_be_removed([Solvable | Solvables], Current,
                    [Solvable | OKSolvables]) :-
    memberchk(Solvable, Current),
    !,
    solvable_to_be_removed(Solvables, Current, OKSolvables).
solvable_to_be_removed([Solvable | Solvables], Current, OKSolvables) :-
    Solvable = solvable(Goal, _, _),
    format('-w: Ignoring attempt to remove a non-existent solvable:-n -w-n',
           ['WARNING', Goal]),
    solvable_to_be_removed(Solvables, Current, OKSolvables).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Updating Data Solvables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name: oaa_AddData(+Clause, +Params).
% purpose: Add a new clause for a DATA solvable (locally and/or remotely)
% Params:
%   address(X): a list including 'self', 'parent', and/or the
%               addresses of other client agents. The default (no address)
%               behavior is the same as with oaa_Solve.
%   reflexive(T_F): Save as with oaa_Solve. Default: true.
%   at_beginning(T_F): if true, uses asserta instead of assertz.
%   Default: false.
%   single_value(T_F): if true, ALL clauses for this predicate are removed
%   before adding the new clause.
%   Default: false.
%   unique_values(T_F): if true, at most one copy of each value is stored.
%   Default: false.
%   owner(LocalId): if bookkeeping(true) for this solvable, record
%   LocalId as the owner.
%   Default: the agent from which the request originated.
%   get_address(X): Returns a list of addresses (ids) of agents that
%   were sent the request.
%   get_satisfiers(X): Returns a list of addresses (ids) of agents that
%   successfully completed the request.

```

```

% reply({true,none}): When data is being added on
%   a remote agent or agents, this tells whether reply message(s) are
%   desired.
%   block(Mode) : true: Block until the reply arrives.
%                 : false: Don't block. In
%                   this case, the reply events (ev_reply_updated)
%                   can be handled by the user's app_do_event callback
%   Default: true. Note that reply(none) overrides
%                 block(true).
% remarks:
%   - Clause is normally a fact (no body), but with Prolog agents, and
%   with rules_ok(true), it's possible for it to have a body.
%   - Triggers will be examined with the on(add) operation mask
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_AddData(Clause, Params) :-
    oaa_update(add, Clause, Params).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_RemoveData(+Clause, +Params).
% purpose: Remove a clause from a DATA solvable (locally and/or remotely)
% Params:
%   address(X): a list including 'self', 'parent', and/or the
%               addresses of other client agents. The default (no address)
%               behavior is the same as with oaa_Solve and oaa_AddData.
%   reflexive(T_F): Save as with oaa_Solve. Default: true.
%   do_all(T_F): If true, removes all predicate values that match the Clause
%                Default: false (removes only the first)
%   get_address(X): Returns a list of addresses (ids) of agents that
%                  were sent the request.
%   get_satisfiers(X): Returns a list of addresses (ids) of agents that
%                     successfully completed the request.
%   owner(LocalId): if bookkeeping(true) for this solvable, remove only
%                   data owned by LocalId.
%                   Default: ignore owner in removing data.
%   reply({true,none}): When data is being removed on
%                       a remote agent or agents, this tells whether reply message(s) are
%                       desired.
%   block(Mode) : true: Block until the reply arrives.
%                 : false: Don't block. In
%                   this case, the reply events (ev_reply_updated)
%                   can be handled by the user's app_do_event callback
%   Default: true. Note that reply(none) overrides
%                 block(true).
% remarks:
%   - Clause is normally a fact (no body), but with Prolog agents, and
%   with rules_ok(true), it's possible for it to have a body.
%   - Triggers will be examined with the 'on_Retract' operation mask.
%   - Not for backtracking.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_RemoveData(Clause, Params) :-
    oaa_update(remove, Clause, Params).

%-----
% name:      oaa_ReplaceData(+Clause1, +Clause2, +Params).
% purpose: Change a predicate value to a new one

```

```

% Clause1: Must be a clause of a writable data solvable.
% Clause2: Must be a clause of a writable data solvable.
% Params:
%   address(X): a list including 'self', 'parent', and/or the
%   addresses of other client agents. The default (no address)
%   behavior is the same as with oaa_Solve and oaa_AddData.
%   reflexive(T_F): Save as with oaa_Solve. Default: true.
%   do_all(T_F): If, true, changes all predicate values that match the
%   Clause1 specification
%   default is 'false': changes only the first
%   at_beginning(T_F): If true, uses asserta instead of assertz
%   default is 'false'
%   owner(LocalId): if bookkeeping(true) for this solvable, record
%   LocalId as the owner of each new data item. Note: It is not possible
%   to specify the owner of the data to be replaced, just that of the
%   NEW data.
%   Default: the agent from which the request originated.
%   get_address(X): Returns a list of addresses (ids) of agents that
%   were sent the request.
%   get_satisfiers(X): Returns a list of addresses (ids) of agents that
%   successfully completed the request.
%   reply({true,none}): When data is being replaced on
%   a remote agent or agents, this tells whether reply message(s) are
%   desired.
%   block(Mode)   : true: Block until the reply arrives.
%                  : false: Don't block. In
%                  this case, the reply events (ev_reply_updated)
%                  can be handled by the user's app_do_event callback
%                  Default: true. Note that reply(none) overrides
%                  block(true).
% remarks:
%   - Clause1 and/or Clause2 may be synonym predicates.
%   - Clause1 and Clause2 are not required to have the same functor.
%   - Clause1 and Clause2 may share variables.
%   - Triggers will be examined with the 'remove' operation mask with Clause1,
%   and the 'add' operation mask with Clause2.
%   - db_replace triggers on the Pred2 argument, not on the Pred1 arg
%   - at_beginning param only used if do_all is false
%-----
oaa_ReplaceData(Clausel, Clause2, Params) :-
    oaa_update(replace, Clausel, [with(Clausel) | Params]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_update(+Mode, +Clause, +Params).
% purpose:   Common code for oaa_AddData, oaaRemoveData, and oaa_ReplaceData.
% Mode:      add, remove, or replace.
% Clause, Params: May include whatever is appropriate for oaa_AddData,
%                oaaRemoveData, or oaa_ReplaceData.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_update(Mode, Clause, InitialParams) :-
    icl_standardize_params(InitialParams, false, Params),
    % Is there a specified address?
    ( memberchk(address(Addr), Params) ->
      true
    | otherwise ->
      Addr = []

```

```

),
% Decide whether or not to update locally:
oaa_Id(Me),
( memberchk(Me, Addr) ->
    delete(Addr, Me, NewAddr),
    replace_element(address(Addr), Params, address(NewAddr), Params1),
    Self = true
| otherwise ->
    NewAddr = Addr,
    Params1 = Params
),
( Addr = [], icl_GetParamValue(reflexive(true), Params1) ->
    % do NOT use remove_element here:
    delete(Params1, reflexive(true), Params2),
    ( oaa_solvables(Solvables) -> true | otherwise -> Solvables = [] ),
    ( oaa_data_matches_solvables(Clause, Solvables, write, _, _) ->
        Self = true
    | otherwise ->
        true
    )
| otherwise ->
    Params2 = Params1
),
% Update locally if appropriate:
( Self == true ->
    Requestees1 = [Me],
    ( Mode == add ->
        Functor = oaa_add_data_local
    | Mode == replace ->
        Functor = oaa_replace_data_local
    | Mode == remove ->
        Functor = oaa_remove_data_local
    ),
    LocalCall =.. [Functor, Clause, Params2],
    ( call(LocalCall) ->
        Updaters1 = [Me]
    | Updaters1 = [] )
| otherwise ->
    Requestees1 = [],
    Updaters1 = []
),
% Update remotely if appropriate:
( oaa_class(leaf), (Addr == [] ; NewAddr \== []) ->
    % Send the ev_post_update event to the Facilitator
    oaa_PostEvent(ev_post_update(Mode, Clause, Params2), []),
    % In the return event, Requestee2s lists all agents to whom
    % the update request was sent; Updaters2 lists those who succeeded.
    ( (icl_GetParamValue(reply(asynchronous), Params) ;
        icl_GetParamValue(reply(none), Params)) ->
        Requestees2 = [],
        Updaters2 = []
    | otherwise ->
        oaa_poll_until_event(
            ev_reply_updated(Mode, Clause, Params2, Requestees2, Updaters2))
    )
),

```

```

    )
  | otherwise ->
    Requestees2 = [],
    Updaters2 = []
  ),
append(Updaters1, Updaters2, Updaters),
% Return Updaters if requested:
( memberchk(get_satisfiers(Updaters), Params) -> true | true ),
append(Requestees1, Requestees2, Requestees),
% Return Requestees if requested:
( memberchk(get_address(Requestees), Params) -> true | true ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_add_data_local(+Clause, +Params)
% purpose:   Assert a clause for an agent's solvable.
% arguments: See comments for oaa_AddData.
% remarks:
%   This performs the local processing needed for calls to oaa_AddData, and
%   ev_update(add, ...) requests.
%   Application code should not call oaa_add_data_local directly, but rather
%   oaa_AddData with address(self).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_add_data_local(Clause1, Params) :-
  ( oaa_solvable(Solvable) -> true | otherwise -> Solvable = []),
  oaa_data_matches_solvable(Clause1, Solvable, write, Clause, Matched),
  Matched = solvable(Pred, DeclParams, _Perms),
  ( Clause = (Head :- Body) ->
    true
  | otherwise ->
    Head = Clause,
    Body = true
  ),

append(Params, DeclParams, AllParams),
% If there's no callback, leave Callback a var:
( memberchk(callback(Callback), AllParams) -> true | true ),

% if single value, erase all old values
(icl_GetParamValue(single_value(true), AllParams) ->
  ( \+ icl_GetParamValue(bookkeeping(false), DeclParams) ->
    oaa_retractall((Pred :- _), _OldOwner, Callback)
  | otherwise ->
    retract_all((Pred :- _))
  )
| true),

% if unique_values(true), make sure fact not already in database
( clause(Head, Body), icl_GetParamValue(unique_values(true), AllParams) ->
  true
| otherwise ->
  ( \+ icl_GetParamValue(bookkeeping(false), DeclParams) ->
    oaa_data_owner(Params, Owner),
    ( icl_GetParamValue(at_beginning(true), AllParams) ->
      oaa_asserta(Clause, Owner, Callback)
    |
      oaa_assertz(Clause, Owner, Callback)
    )
  )
)

```

```

    )
  | otherwise ->
    ( icl_GetParamValue(at_beginning(true), AllParams) ->
      asserta(Clause)
    |
      assertz(Clause)
    )
  )
),
oaa_CheckTriggers(data, Head, add),
!.

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_remove_data_local(+Clause, +Params)
% purpose:   Retract a clause (or all clauses) from an agent's solvable.
% arguments: See comments for oaaRemoveData.
% remarks:

```

```

% This performs the local processing needed for calls to oaaRemoveData, and
% ev_update(remove, ...) requests.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

oaa_remove_data_local(Clause1, Params) :-
  ( oaa_solvable(Solvable) -> true | otherwise -> Solvable = []),
  oaa_data_matches_solvable(Clause1, Solvable, write, Clause, Matched),
  Matched = solvable(_Pred, DeclParams, _Perms),
  ( Clause = (Head :- Body) ->
    true
  | otherwise ->
    Head = Clause,
    Body = true
  ),
  append(Params, DeclParams, AllParams),
  ( memberchk(callback(Callback), AllParams) -> true | true ),

  ( \+ icl_GetParamValue(bookkeeping(false), DeclParams) ->
    ( icl_GetParamValue(owner(Owner), Params) -> true | true ),
    ( icl_GetParamValue(do_all(true), Params) ->
      oaa_retractall(Clause, Owner, Callback)
    | otherwise ->
      oaa_retract(Clause, Owner, Callback)
    )
  | otherwise ->
    ( icl_GetParamValue(do_all(true), Params) ->
      retract_all(Clause)
    | otherwise ->
      retract(Clause)
    )
  ),
  oaa_CheckTriggers(data, Head, remove),
  !.

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_replace_data_local(+Clause1, +Params)
% purpose:   Replace one or more clauses from an agent's solvable.
% arguments: See comments for oaa_ReplaceData.

```

```

% remarks:
%   This performs the local processing needed for calls to oaa_ReplaceData, and
%   ev_update(replace, ...) requests.
%   Clause1 is the thing to be replaced. The thing to replace it with must
%   be present in Params, as with(Clause2).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_replace_data_local(ClauselIn, Params) :-
    memberchk(with(Clause2In), Params),
    ( oaa_solvables(Solvables) -> true | otherwise -> Solvables = []),
    oaa_data_matches_solvables(ClauselIn, Solvables, write, Clause1, Matched),
    oaa_data_matches_solvables(Clause2In, Solvables, write, Clause2, _Matched2),
    Matched = solvable(_Pred, DeclParams, _Perms),
    ( Clause1 = (Head :- Body) ->
        true
    | otherwise ->
        Head = Clause1,
        Body = true
    ),

    append(Params, DeclParams, AllParams),
    ( memberchk(callback(Callback), AllParams) -> true | true ),

% do replace of either one or all occurrences
( \+ icl_GetParamValue(bookkeeping(false), DeclParams) ->
    oaa_data_owner(Params, Owner),
    ( icl_GetParamValue(do_all(true), Params) ->
        oaa_replace_all(Clausel1, Clause2, Owner, Callback)
    | otherwise ->
        oaa_retract(Clausel1, _OldOwner, Callback),
        (icl_GetParamValue(at_beginning(true), AllParams) ->
            oaa_asserta(Clausel2, Owner, Callback)
        | oaa_assertz(Clausel2, Owner, Callback)
        )
    )
| otherwise ->
    ( icl_GetParamValue(do_all(true), Params) ->
        replace_all(Clausel1, Clause2)
    | otherwise ->
        retract(Clausel1),
        (icl_GetParamValue(at_beginning(true), AllParams) ->
            asserta(Clausel2)
        | assertz(Clausel2)
        )
    )
),
oaa_CheckTriggers(data, Clause1, remove),
oaa_CheckTriggers(data, Clause2, add),
!.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      retract_all
% purpose:   Remove all clauses matching Clause1
% remarks:   Always succeeds. Needed because retractall((func(X) :- Y)) doesn't
%           work.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
retract_all(Clausel1) :-
    retract(Clausel1),

```



```
fail.
retract_all(_Clause1).
```

```
*****
% name: replace_all
% purpose: Replace all clauses matching Clause1 by Clause2
% remarks: Always succeeds
*****
replace_all(Clause1, Clause2) :-
    retract(Clause1),
    assert(Clause2),
    fail.
replace_all(_Clause1, _Clause2).
```

```
*****
% name: oaa_data_owner(+Params, -Owner)
% purpose: Determine data ownership from the available params
*****
oaa_data_owner(Params, Owner) :-
    ( memberchk(owner(Owner), Params) ->
      true
    | memberchk(from(Owner), Params) ->
      true
    | oaa_Id(Owner) ->
      true
    | otherwise ->
      Owner = unknown
    ).
```

```
*****
% name: oaa_Id(MyId)
% purpose: Return the Id of the current agent
*****
% if connected to a Facilitator, use this Id
oaa_Id(MyId) :-
    com:com_GetInfo(parent, oaa_id(MyId)), !.
% For root, get any id
oaa_Id(MyId) :-
    com:com_GetInfo(ConnectionId, type(server)),
    com:com_GetInfo(ConnectionId, oaa_id(MyId)), !.
```

```
*****
% name: oaa_Name(MyName)
% purpose: Return the name of the current agent
*****
% if connected to a Facilitator, use this Id
oaa_Name(MyName) :-
    com:com_GetInfo(parent, oaa_name(MyName)), !.
% For root, get any id
oaa_Name(MyName) :-
    com:com_GetInfo(ConnectionId, type(server)),
    com:com_GetInfo(ConnectionId, oaa_name(MyName)), !.
```

```
*****
% name: oaa_class(MyClass)
```

```

% purpose: Return the class (leaf, node, root) of the current agent
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% if connected to a Facilitator, use this Id
oaa_class(leaf) :-
    com:com_GetInfo(_, type(client)),
    \+ com:com_GetInfo(_, type(server)), !.
oaa_class(node) :-
    com:com_GetInfo(_, type(client)),
    com:com_GetInfo(_, type(server)), !.
oaa_class(root) :-
    com:com_GetInfo(_, type(server)),
    \+ com:com_GetInfo(_, type(client)), !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name: oaa_asserta(Clause, Owner, SpecifiedCallback)
%       oaa_assertz(Clause, Owner, SpecifiedCallback)
%       oaa_retract(Clause, Owner, SpecifiedCallback)
%       oaa_retractall(Clause, Owner, SpecifiedCallback)
%       oaa_replace_all(Clause1, Clause2, Owner, SpecifiedCallback)
% purpose: Perform data updates with bookkeeping info (in oaa_data_ref/3)
% remarks: These should only be used with data solvables having param
%          bookkeeping(true).
%          There are still a couple limitations related to data callbacks.
%          First, callbacks don't work when bookkeeping(false).
%          Second, oaa_replace_all assumes the same callback is appropriate
%          for both the old and new facts.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_asserta(Clause, Owner, Callback) :-
    asserta(Clause, Ref),
    now(Time),
    assert(oaa_data_ref(Ref, Owner, Time)),
    oaa_call_callback(app_on_data_change, Callback, [add(Clause)]).

oaa_assertz(Clause, Owner, Callback) :-
    assertz(Clause, Ref),
    now(Time),
    assert(oaa_data_ref(Ref, Owner, Time)),
    oaa_call_callback(app_on_data_change, Callback, [add(Clause)]).

oaa_retract(Clause, Owner, Callback) :-
    ( Clause = (Head :- Body) ->
      true
    | otherwise ->
      Head = Clause,
      Body = true
    ),
    clause(Head, Body, Ref),
    ( retract(oaa_data_ref(Ref, Owner, _)) ->
      erase(Ref),
      oaa_call_callback(app_on_data_change, Callback, [remove(Clause)])
    ).

oaa_retractall(Clause, Owner, Callback) :-
    ( Clause = (Head :- Body) ->
      true
    | otherwise ->

```

```

        Head = Clause,
        Body = true
    ),
    clause(Head, Body, Ref),
    ( retract(oaa_data_ref(Ref, Owner, _)) ->
      erase(Ref),
      oaa_call_callback(app_on_data_change, Callback, [remove(Check)])
    ),
    fail.
oaa_retractall(_Clause, _Owner, _Callback).

```

```

oaa_replace_all(Check1, Check2, Owner, Callback) :-
    oaa_retract(Check1, _OldOwner, Callback),
    oaa_assertz(Check2, Owner, Callback),
    % This would be redundant:
    % oaa_call_callback(app_on_data_change, Callback, [replace(Check1,
    Check2)]),
    fail.
oaa_replace_all(_Check1, _Check2, _Owner, _Callback).

```

```

%*****
% Trigger Handling
%*****

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_CheckTriggers
% purpose:  Given a trigger type, a mask and an Op (e.g. [send, receive],
%           [add, remove], etc), see if any triggers fire.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_CheckTriggers(Type, Condition, Op) :-
    % for each matching trigger
    oaa_solve_local(
        oaa_trigger(TriggerId, Type, Condition, Action, Params),
        []),

    ( (Type == task, \+ var(Condition)) ->
      % We don't want this to succeed more than once, so use ->
      ( oaa_Interpret(Condition, [from(self)]) -> true )
    | otherwise ->
      true
    ),

    % see if on(Op) has been specified
    (memberchk(on(OpSpecified), Params) ->
      OpMask = OpSpecified
    | OpMask = _),

    % see if Op is OK
    ( (ground(OpMask), OpMask = [_|_]) ->
      memberchk(Op, OpMask)
    | otherwise ->
      Op = OpMask
    ),

    % test additional conditions

```

```

(memberchk(test(Test), Params) ->
  % We don't want this to succeed more than once, so use ->
  ( oaa_Interpret(Test, [from(self)]) -> true )
| Test = 'true'),

% check recurrence: remove trigger?
(remove_element(recurrence(R), Params, NewParams) ->
  (R = whenever ->
    true % don't remove trigger if 'whenever'
| integer(R), R > 1 ->
  R2 is R - 1,
  % decrement recurrence count
  oaa_remove_data_local(
    oaa_trigger(TriggerId, Type, Condition, Action, Params),
    []),
  oaa_add_data_local(
    oaa_trigger(TriggerId, Type, Condition, Action,
      [recurrence(R2)|NewParams]),
    []))
| oaa_remove_local_trigger_by_id(TriggerId)
)

|
  R = when,
  oaa_remove_local_trigger_by_id(TriggerId)
),

oaa_TraceMsg(
  '-n-q trigger fired (-q): -q AND -q,-n Action: -q-n',
  [Type, Op, Cond, Test, Action]),

(Type \== comm ->
  oaa_Inform(trigger,
    'trigger_fired(~q,-q,-q,-q)-n',
    [Type, Cond, Action, Params])
| true),

% FIRE!!!!
oaa_fire_trigger(Action),

% loop back for more triggers
fail.
oaa_CheckTriggers(_Type, _Cond, _Op).

oaa_fire_trigger(oaa_Solve(Goal, Params)) :-
  !,
  ( memberchk(block(_), Params) ->
    NewParams = Params
  | otherwise ->
    append([block(false)], Params, NewParams)
  ),
  oaa_Solve(Goal, NewParams).
oaa_fire_trigger(oaa_Solve(Goal)) :-
  !,
  oaa_Solve(Goal, [block(false)]).
oaa_fire_trigger(oaa_Interpret(Goal, Params)) :-
  !,

```

```

    ( memberchk(from(_), Params) ->
      NewParams = Params
    | otherwise ->
      oaa_Id(Me),
      append([from(Me)], Params, NewParams)
    ),
    oaa_Interpret(Goal, NewParams).
oaa_fire_trigger(oaa_Interpret(Goal)) :-
    !,
    oaa_Id(Me),
    oaa_Interpret(Goal, [from(Me)]).
oaa_fire_trigger(Goal) :-
    oaa_Id(Me),
    oaa_Interpret(Goal, [from(Me)]).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

% name:      oaa_AddTrigger
% purpose:   Adds a trigger according to parameters
% Type      = comm, data, task, time
% Condition= comm:event to match,  data:data to match, task:solvable to call
%           time:@@
% Action    = Can be any of these:
%             oaa_Solve(Goal, Params)
%             oaa_Interpret(Goal, Params)
%             Goal [passed to oaa_Interpret with default params]
% Params    =
%   address(X): a list including 'self', 'parent', and/or the
%   addresses of other client agents. Default: see below.
%   test(T): additional tests before trigger will fire [@@needs work?]
%   on(OP) : operation check: on(add), on(remove), on(receive), etc.
%   recurrence(R): when, whenever, or integer (# of times to execute)
%   reply({true,none}): When a trigger is being added on
%   a remote agent or agents, this tells whether reply message(s) are
%   desired.
%   block(Mode) : true: Block until the reply arrives.
%                 : false: Don't block. In
%                   this case, the reply events
%                   can be handled by the user's app_do_event callback
%                 Default: true. Note that reply(none) overrides
%                 block(true).
%   get_address(X): Returns a list of addresses (ids) of agents that
%   were sent the request.
%   get_satisfiers(X): Returns a list of addresses (ids) of agents that
%   successfully completed the request.
%

```

```

% Default destination for triggers:
%   Data triggers: all agents with solvables matching the Condition
%   field.
%   All other types: the local agent

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

oaa_AddTrigger(Type, Condition, Action, InitialParams) :-
    oaa_update_trigger(add, Type, Condition, Action, InitialParams).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

% name:      oaa_RemoveTrigger

```

```

% purpose: Removes a trigger from a local or remote agent
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_RemoveTrigger(Type,Condition,Action,Params) :-
    oaa_update_trigger(remove, Type, Condition, Action, Params).

oaa_update_trigger(Mode, Type, InCondition, Action, InParams) :-
    ( (Type == comm, \+ InCondition = event(_,_)) ->
        Condition = event(InCondition, _)
    | otherwise ->
        Condition = InCondition
    ),
    icl_standardize_params(InParams, false, Params),
    % Is there a specified address?
    ( memberchk(address(Addr), Params) ->
        true
    | otherwise ->
        Addr = []
    ),

    % Decide whether or not to update locally:
    oaa_Id(Me),
    ( Addr \== [], memberchk(Me, Addr) ->
        delete(Addr, Me, NewAddr),
        replace_element(address(Addr), Params, address(NewAddr), Params1),
        Self = true
    | Addr = [], Type == data, icl_GetParamValue(reflexive(true), Params) ->
        % Do NOT use remove_element here:
        delete(Params, reflexive(true), Params1),
        NewAddr = Addr,
        Self = true
    | Addr = [], Type \== data ->
        NewAddr = Addr,
        Params1 = Params,
        Self = true
    | otherwise ->
        NewAddr = Addr,
        Params1 = Params
    ),

    % Update locally if appropriate:
    ( Self == true ->
        Requestees1 = [Me],
        ( Type == add ->
            Functor = oaa_add_trigger_local
        | otherwise ->
            Functor = oaa_remove_trigger_local
        ),
        LocalCall =.. [Functor, Type, Condition, Action, Params1],
        ( call(LocalCall) ->
            Updaters1 = [Me]
        | Updaters1 = []
        )
    | otherwise ->
        Requestees1 = [],
        Updaters1 = []
    ),

    % Update remotely if appropriate:

```

```

( oaa_class(leaf), ((Addr == [], Type = data) ; NewAddr \== []) ->
  % Send the request event to the Facilitator
  oaa_PostEvent(
    ev_post_trigger_update(Mode,Type,Condition,Action,Params1), [],
    ( (icl_GetParamValue(reply(asynchronous), Params) ;
      icl_GetParamValue(reply(none), Params)) ->
      Requestees2 = [],
      Updaters2 = []
    | otherwise ->
      % In the return event, Requestees lists all agents to whom
      % the update request was sent; Updaters2 lists those who succeeded.
      oaa_poll_until_event(
        ev_reply_trigger_updated(Mode, Type, Condition, Action, Params1,
          Requestees2, Updaters2))
      )
    | otherwise ->
      Requestees2 = [],
      Updaters2 = []
  ),
  append(Updaters1, Updaters2, Updaters),
  % Return Updaters if requested:
  ( memberchk(get_satisfiers(Updaters), Params) -> true | true ),
  append(Requestees1, Requestees2, Requestees),
  % Return Requestees if requested:
  ( memberchk(get_address(Requestees), Params) -> true | true ).

oaa_add_trigger_local(Type, Condition, Action, Params) :-
  gensym(trg, TriggerId),
  oaa_add_data_local(
    oaa_trigger(TriggerId, Type, Condition, Action, Params),
    []).

oaa_remove_trigger_local(Type, Condition, Action, Params) :-
  oaa_remove_data_local(
    oaa_trigger(_TriggerId, Type, Condition, Action, Params),
    []).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_remove_local_trigger_by_id
% purpose:  Removes a local trigger given its unique identifier
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_remove_local_trigger_by_id(TriggerId) :-
  oaa_remove_data_local(oaa_trigger(TriggerId, _,_,_,_), []).
!.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Requesting Services
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_Solve
% purpose:  Sends work or information requests to distributed agents, brokered
%           by the Facilitator agent
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

%
% The default behavior (paramlist = []) is to act like the Prolog primitive
% call(Goal), blocking until Goal is finished, and unifying and backtracking
% over solutions for Goal.
%
% This behavior may be modified by a parameter list, which may contain:
%
%   cache(T_F)           : cache all solutions locally, and if good solutions
%                         already exist in the cache, use the local values
%                         instead of making a distributed request.
%                         Default: false.
%   level_limit(N)      : highest number of hierarchical levels to climb for
%                         solutions.
%   address(AgentId)    : send request to specific agent, given its name or Addr
%                         If AgentID is 'self', solves the goal locally
%   reply(Mode)         : true: Reply desired.
%                         : none: No reply desired.
%                         Default: true, except when the call to oaa_solve
%                         is a trigger action, in which case it is
%                         none. 'none' is used here instead of false,
%                         because we anticipate some additional values.
%   block(Mode)         : true: Block until the reply arrives.
%                         : false: Don't block. In
%                         this case, the reply events (ev_reply_solved)
%                         can be handled by the user's app_do_event callback
%                         Default: true, except when the call to oaa_solve
%                         is a trigger action, in which case it is
%                         false. Note that reply(none) overrides
%                         block(true).
%   solution_limit(N)   : limits the maximum number of solutions found to N
%   time_limit(N)       : Waits a maximum of N seconds before returning
%                         (failure if no solution found in time).
%   context(C)          : Passes a context value through any subsequent
%                         solves.
%   parallel_ok(T_F)    : if T_F is 'true' (default), multiple agents
%                         that can solve the Goal will attempt to work on it
%                         in parallel. If 'false', one agent will be selected
%                         at a time to solve the goal, until the maximum
%                         number of requested solutions (see solution_limit) is
%                         found.
%   reflexive(T_F)      : If T_F is `true', the Facilitator will consider the
%                         originating agent when choosing agents to solve a
%                         request. Default: true.
%   priority(P)         : P ranges from 1 (low priority) to 10 (high priority)
%                         with a default of 5.
%   flush_events(T_F)   : Will flush (dispose of) all events of lower priority
%                         currently queued at the destination agent. These
%                         events are lost, and will not be executed.
%                         This parameter should be used with caution!!!
%                         Default: false.
%   get_address(X)      : Returns a list of addresses (ids) of agents that
%                         were asked to solve the goal, or one of its subgoals
%   get_satisfiers(X)   : Returns a list of addresses (ids) of agents that

```



```

%           succeeded in solving the goal, or one of its
%           subgoals.
%
%   strategy(S)      : Shorthand for certain combinations of the above
%                     parameters.  S is one of
%                     query = [parallel_ok(true)]
%                     action = [parallel_ok(false), solution_limit(1)]
%                     inform = [parallel_ok(true), reply(none)]
%
% Remarks: Note that certain combinations of parameters are inconsistent,
% and are handled as follows:
%   reply(none) overrides block(true)
%   reply(none) overrides parallel_ok(false)
%
% All of the above parameters may be used in the "global" parameter
% list (the second argument to oaa_Solve), when Goal is non-compound.
% Most can be used in the global list with compound goals also.
% Some of these parameters can also be used in the NESTED parameter
% lists of compound goals.  Uses of these parameters with compound
% goals are documented elsewhere.  When that documentation exists,
% this will go there:
% With many compound goals, however, the get_satisfier/1 parameter isn't
% really meaningful.  Thus, with compound goals, it is often best to use
% this parameter in a nested parameter list.

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

oaa_Solve(Goal, InitialParams) :-
    % Trace message
    oaa_TraceMsg('-n-nStarting oaa_Solve request:-n  -q [-q]...~n',
                [Goal,Params]),

    icl_standardize_params(InitialParams, false, Params),
    % Check for inappropriate params
    ( icl_GetParamValue(cache(true), Params), icl_compound_goal(Goal) ->
      format('-w: -w (-w)-n Goal: ~w~n',
            ['WARNING', 'Ignoring 'cache' parameter',
             'cannot be used with compound goal', Goal]),
      Compound = true
    | otherwise ->
      Compound = false
    ),

    % Add context to params
    ( oaa_current_contexts(_, Contexts) ->
      append(Contexts, Params, NewParams)
    | otherwise ->
      NewParams = Params
    ),

    % check cache
    (icl_GetParamValue(cache(true), NewParams), \+ Compound,
     on_exception(_, oaa_InCache(Goal, Solutions), fail) ->
      oaa_TraceMsg('-n-nSolutions found in cache:-n  -q.-n',
                  [Solutions])
    |
      % Should I solve this only locally?
      (oaa_Id(Me),

```

```

    memberchk(address(Me), Params) ->
        findall(Goal, oaa_solve_local(Goal, NewParams), Solutions)

|
% send request to Facilitator
oaa_cont_solve(Goal, NewParams, Solutions),

% print appropriate trace message
(icl_GetParamValue(reply(none), NewParams) ->
    oaa_TraceMsg('-n-nMessage broadcast.-n', []))
|
    oaa_TraceMsg('-n-nSolutions returned:-n    -q.-n',
        [Solutions])
),

% cache returned solutions if necessary
((icl_GetParamValue(cache(true), NewParams), Solutions \== []) ->
    oaa_AddToCache(Goal, Solutions),
    oaa_TraceMsg('Solutions cached.-n', []))
| true)
)

),!,

% backtrack over all solutions
member(Goal, Solutions).

oaa_solve_local(FullGoal, Params) :-
% Validate the goal:
icl_GoalComponents(FullGoal, _, Goal1, GoalParams),
( oaa_solvable(Solvables) -> true | otherwise -> Solvables = []),
( icl_compound_goal(Goal1) ;
    icl_BuiltIn(Goal1) ;
    oaa_goal_matches_solvable(Goal1, Solvables, Goal, Matched) ),
!,

% More "local" params take precedence, so they go to the
% beginning of the list:
append([GoalParams, Params], AllParams),

% We don't want tests to be performed repeatedly with compound goals,
% so we remove them after testing.
( passes_tests(AllParams) ->
    delete(AllParams, test(_), NewParams),
    ( ( \+ var(Matched), Matched = solvable(_, SolvParams, _),
        icl_GetParamValue(type(data), SolvParams) ) ->
        ( memberchk(solution_limit(N), AllParams) ->
            call_n(N, Goal)
            | otherwise ->
                call(Goal)
        )
    )
| otherwise ->
    ( memberchk(solution_limit(N), AllParams) ->
        call_n(N, oaa_Interpret(Goal, NewParams))
    | otherwise ->
        oaa_Interpret(Goal, NewParams)
    )
)

```

```

)
| otherwise ->
  oaa_TraceMsg('-nDoesn''t pass test in: -q-n', [AllParams]),
  fail
).

oaa_solve_local(FullGoal, _Params) :-
  format('-nError: do not know how to solve: -q-n', [FullGoal]), fail.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_cont_solve
% purpose:   Post request for solutions, and if appropriate, poll until
%           results are returned.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_cont_solve(Goal, GlobalParams, Solutions) :-
  % Send the ev_post_solve event to the Facilitator
  oaa_PostEvent(ev_post_solve(Goal, GlobalParams), []),

  % Compound goals may also contain relevant params
  icl_GoalComponents(Goal, _, _, Params),
  append(Params, GlobalParams, AllParams),

  % If delayed reply or no reply OK, succeed immediately
  ( ( icl_GetParamValue(reply(false), AllParams) ;
      icl_GetParamValue(reply(none), AllParams) ;
      icl_GetParamValue(block(false), AllParams) ) ->
    Solutions = [Goal],
    Requestees = [],
    Solvers = []
  |
    % otherwise wait for solutions to return

    icl_GetParamValue(priority(P), AllParams),
    oaa_poll_until_event(ev_reply_solved(Requestees, Solvers, Goal,
SolvedParams, Solutions),
                        P),

    % The facilitator is responsible for making SolvedParams
    % unifiable with GlobalParams. This msg is to keep facilitator
    % writers honest.
    ( GlobalParams = SolvedParams ->
      true
    | otherwise ->
      format('~w: ~w ~w-n ~w: ~w-n',
        ['WARNING:', 'Params in solved event don''t unify',
        'with original params', 'SolvedParams', SolvedParams])
    )
  ),
  % Return Solvers if requested:
  ( memberchk(get_satisfiers(Solvers), GlobalParams) -> true | true ),
  % Return Requestees if requested:
  ( memberchk(get_address(Requestees), GlobalParams) -> true | true ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_Solve/1
% purpose:   Convenience function: oaa_Solve with default parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
oaa_Solve(Goal) :- oaa_Solve(Goal, []).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_InCache
% purpose: Retrieve solutions from the cache if the goal we are
%           asking for is properly contained in the cache (check subsumption)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_InCache(Goal, Solutions):-
    oaa_cache(SomeGoal, _),
    subsumes_chk(SomeGoal, Goal),
    !,
    findall(Solution, oaa_cache(Goal, Solution), Solutions).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_AddToCache
% purpose: Add each solution to goal one at a time
%           so we can retrieve solutions later using findall
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_AddToCache(Goal, Solutions) :-
    member(Solution, Solutions),
    \+ oaa_cache(Goal, Solution),
    assert(oaa_cache(Goal, Solution)),
    fail.
oaa_AddToCache(_Goal, _Solutions).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_ClearCache
% purpose: Clear the cache
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_ClearCache :-
    retractall(oaa_cache(_, _)).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_poll_until_event
% purpose: Block until requested event arrives in oaa_GetEvent
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_poll_until_event(Event) :-
    icl_param_default(priority(P)),
    oaa_poll_until_event(Event, P).

oaa_poll_until_event(Event, Priority) :-
    oaa_poll_until_all_events([Event], Priority).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_poll_until_all_events
% purpose: Block until all requested events arrive
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% no more events: we're done!
```

```

oaa_poll_until_all_events([], _Priority) :- !.

%% @@Adam - you were apparently working on this; I corrected a syntax
%% error or two, but otherwise left it alone. - Dave
oaa_poll_until_all_events(EventList, Priority) :-
    % If we have a waiting_event, grab it
    % see problem description in (oaa_is_waiting_for)
    (oaa_grab_waiting_event(EventList, Event) ;
     oaa_GetEvent(Event, Params, 0)),

    % if timeout returned, check triggers and call user:oaa_AppIdle
    % then fail (continue with next clause)
    (Event = timeout ->
     oaa_CheckTriggers(task, _, _),
     oaa_call_callback(app_idle, _, []),
     fail
    |
     oaa_cont_poll_until_all_events(EventList, Event, Params, Priority)
    ), !.

% if oaa_GetEvent fails (e.g. timeout), just continue waiting
oaa_poll_until_all_events(EventList, Priority) :-
    oaa_poll_until_all_events(EventList, Priority).

oaa_cont_poll_until_all_events(EventList, Event, _Params, Priority) :-
    remove_element(Event, EventList, NewEventList), !,
    oaa_poll_until_all_events(NewEventList, Priority).
oaa_cont_poll_until_all_events(EventList, Event, Params, Priority) :-
    % if the new event is a ev_reply_solved() message for which we
    % are waiting at a higher recursive level, save this for
    % a later time, until we pop back out to the correct level.
    (oaa_is_waiting_for(Event) ->
     assert(oaa_waiting_event(Event))
    |
     % record what events we are waiting for on this processing level
     gensym(wait, WaitId),
     assert(oaa_waiting_for(WaitId, EventList)),

     (oaa_ProcessEvent(Event, Params) | true), !,

     % level over, remove waiting statement
     retract(oaa_waiting_for(WaitId, EventList))
    ),
    oaa_poll_until_all_events(EventList, Priority).

%*****
% Callbacks
%*****

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_RegisterCallback
% purpose:  Declare what procedures should be used for callbacks.  These
%           are application-defined procedures called by library code.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

oaa_RegisterCallback(CallbackID, CallbackProc) :-

```

```

( CallbackProc = Module:Proc ->
  true
| otherwise ->
  Module = user,
  Proc = CallbackProc
),
retractall( oaa_callback(CallbackID, _) ),
assert( oaa_callback(CallbackID, Module:Proc) ).

oaa_call_callback(CallbackID, SpecifiedCB, Args) :-
( ground(SpecifiedCB) ->
  SpecifiedCB = Module:Functor
| otherwise ->
  oaa_callback(CallbackID, Module:Functor)
),
!,
Call =.. [Functor | Args],
on_exception(E,
  Module:Call,
  ( oaa_TraceMsg('WARNING (oaa.pl): Exception raised thru callback
handler (-w):-n -q-n',
                [Module:Functor, E]),
    fail )
).
oaa_call_callback(_CallbackID, _SpecifiedCB, _Args).

%*****
% Debugging
%*****

%*****
% name:      oaa_TraceMsg
% purpose:  If trace mode is on, display message and arguments
%*****
oaa_TraceMsg(FormatString, Args) :-
  (oaa_trace(on) ->
    format(FormatString, Args)
%    oaa_Inform(trace_info, FormatString, Args)
    ;
    true).

%*****
% name:      oaa_ComTraceMsg
% purpose:  If com trace mode is on, display message and arguments
%*****
oaa_ComTraceMsg(FormatString, Args) :-
  (oaa_com_trace(on) ->
    format(FormatString, Args)
%    oaa_Inform(trace_info, FormatString, Args)
    ;
    true).

%*****
% name:      oaa_turn_on_debug
% purpose:  start debugging if debug mode is on
% remarks:

```

```

% Use predicate_property and call so as to avoid errors in
% building and running a Quintus runtime system.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_turn_on_debug :-
    (oaa_debug(on) ->
        ( predicate_property(user:trace, built_in) ->
            call(user:trace)
          | true )
    | true).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name: oaa_turn_off_debug
% purpose: stop debugging if debug mode is on
% remarks:
% Use predicate_property and call so as to avoid errors in
% building and running a Quintus runtime system.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_turn_off_debug :-
    (oaa_debug(on) ->
        ( predicate_property(user:nodebug, built_in) ->
            call(user:nodebug)
          | true )
    | true).

%*****
% User Interface
%*****

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name: oaa_inform
% purpose: sends a typed message to interested agents
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_inform(TypeInfo, FormatString, Args) :-
    oaa_TraceMsg(FormatString, Args),
    (oaa_class(leaf) ->
        sprintf(Result, FormatString, Args),
        oaa_Solve(inform(TypeInfo, Result), [strategy(inform)])
    |
        true
    ), !.

%*****
% Connection primitives
%*****

%%% BUG/HACK!!!!
% tcp_send/1 is not currently defined (new version of quintus)
% so these predicates should fail. This means we can't have
% multilevel facilitators.
% However, if we fix it by the tcp_send/2 version (commented out),
% killing the agent doesn't shut down both connections and the
% facilitator server doesn't register the agent as disconnected.

```

```

% This must be fixed, but I don't have time now...

% Ask the root agent for the address of facilitator FacName.
% Either FacId or FacName may be bound.
% IMPORTANT: This assumes the root agent is the only connection when
% this is called.
% @@Not happy with the use of a Connection number in the address param here.
% Can an address be a connection number as well as an id or name??? [No.]

% get_address(FacId, FacName, Port, Host):-
%     tcp_connected(RootConnection),
%     oaa_Solve(agent_location(FacId, FacName, Port, Host),
%             [address(RootConnection)]).

%% succeed if FacName has not been registered with the root agent.
%%     otherwise, ask user to enter a different name for FacName

% check_name_duplication(MyName, NewMyName) :-
%     tcp_send(ev_check_agent_name(MyName)),
%     oaa_select_event(0, X),
%     oaa_extract_event(X, Result, _), %% 'UNIQUE'
%     (Result == 'UNIQUE' -> NewMyName = MyName
%     ;
%     format('Name is duplicated~n', []),
%     format('The following are registered ~n ~q ~n', [Result]),
%     format('Input agent name again:', []),
%     read(NewMyName)).

% report_address_to_root(MyName, NewAddress):-
%     tcp_send(register_port_number(MyName, NewAddress)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% routines to fix bug:
%   blocking solve1
%     incoming event generates blocking solve2
%     solution to solve1 thrown away!!!
%     solutions to solve2
%   stuck waiting for solve1 forever
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_is_waiting_for
% purpose: Check to see if the current event is something we are waiting
%           for on a higher recursive level
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_is_waiting_for(Event) :-
    oaa_waiting_for(_Id, EventList),
    memberchk(Event, EventList).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_grab_waiting_event
% purpose: If one of the delayed events is in the EventList that we are
%           waiting for, return this event and remove from delayed list
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_grab_waiting_event(EventList, Event) :-

```



```

    oaa_waiting_event(Event),
    memberchk(Event, EventList),
    !,
    retract(oaa_waiting_event(Event)).

%*****
% OAA Utilities
%*****

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_remove_solvable_data(Solvable).
% purpose:   For each data solvable, remove all clauses belonging to it.
% remarks:   - Solvables must be in standard form, and should include only
%            data solvables.
%            - Permissions are ignored.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_remove_solvable_data([]).
oaa_remove_solvable_data([Solvable | Solvables]) :-
    Solvable = solvable(Goal, Params, _Perms),
    icl_GetParamValue(type(data), Params),
    \+ memberchk(synonym(_, _), Params),
    !,
    % This should have already been done, but to be safe:
    (clause(Goal, _, _) -> true | true),
    predicate_skeleton(Goal, Skeleton),
    ( oaa_remove_data_local(Skeleton, [do_all(true)]) ->
      true
    | otherwise ->
      format('~w: Problem in removing all data for solvable: ~w~n',
        ['! ERROR', Goal])
    ),
    oaa_remove_solvable_data(Solvables).
oaa_remove_solvable_data([_Solvable | Solvables]) :-
    oaa_remove_solvable_data(Solvables).

oaa_remove_data_owned_by(Id) :-
    ( oaa_solvable(Solvables) -> true | otherwise -> Solvables = []),
    oaa_built_in_solvable(BuiltIns),
    append(BuiltIns, Solvables, AllSolvables),
    oaa_remove_data_owned_by(AllSolvables, Id).

oaa_remove_data_owned_by([], _Id).
oaa_remove_data_owned_by([Solvable | Solvables], Id) :-
    Solvable = solvable(Goal, Params, _Perms),
    icl_GetParamValue(type(data), Params),
    \+ icl_GetParamValue(persistent(true), Params),
    \+ icl_GetParamValue(synonym(_, _), Params),
    !,
    % This should have already been done, but to be safe:
    (clause(Goal, _, _) -> true | true),
    predicate_skeleton(Goal, Skeleton),
    ( oaa_remove_data_local(Skeleton, [owner(Id), do_all(true)]) ->
      true
    | otherwise ->
      format('~w: Problem in removing data owned by ~w for solvable:~n ~w~n',
        ['! ERROR', Id, Goal])
    )

```

```

),
oaa_remove_data_owned_by(Solvables, Id).
oaa_remove_data_owned_by([_Solvable | Solvables], Id) :-
oaa_remove_data_owned_by(Solvables, Id).

```

```

%*****
% General Utilities
%*****

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_consult(+FilePath, -AbsFileName).
% purpose:
% remarks: We don't use Quintus' builtin consult, because it's too picky
%          about associating predicates with files.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_consult(FilePath, AbsFileName) :-
absolute_file_name(FilePath, AbsFileName),
can_open_file(AbsFileName, read, fail),
open(AbsFileName, read, Stream),
load_clauses(Stream),
close(Stream).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      load_clauses(+Stream).
% purpose:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load_clauses(Stream) :-
repeat,
read_term(Stream, [], Term),
( Term = ':-'(_Body) ->
true
| Term = end_of_file ->
true
| otherwise ->
load_clause(Term)
),
( at_end_of_file(Stream) ->
!
| otherwise ->
fail
).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      load_clause(+Term).
% purpose:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load_clause(Term) :-
assert( Term ).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_declare_for_prolog(Solvables).
% purpose: For each solvable, make sure it's known to Prolog as a dynamic
%          predicate. This will prevent exceptions and warnings from

```

```

%      calls and retracts before there have been any asserts.
% remarks: Solvables must be in standard form, and should include only
%      data solvables.
%      This is probably Quintus-specific.
%      We are assuming that none of these predicates are known to
%      Prolog as compiled predicates.  Would be better to check for this.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_declare_for_prolog([]).
oaa_declare_for_prolog([solvable(Pred, _, _) | Rest]) :-
    copy_term(Pred, PredCopy),
    ( clause(PredCopy, _Body) -> true | true ),
    oaa_declare_for_prolog(Rest).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      predicate_skeleton(+Goal, +Skeleton).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
predicate_skeleton(Goal, Skeleton) :-
    functor(Goal, Functor, Arity),
    functor(Skeleton, Functor, Arity).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      sprintf
% purpose:   C-like command formats a string + args into an atom
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sprintf(AtomResult, FormatStr, Args) :-
    with_output_to_chars(format(FormatStr, Args), Chars),
    name(AtomResult, Chars).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      memberchk_nobind
% purpose:   like memberchk, but doesn't bind variables in Elt when doing test.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
memberchk_nobind(Elt, [H|_]) :-
    would_unify(Elt, H), !.
memberchk_nobind(Elt, [_|T]) :-
    memberchk_nobind(Elt, T).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      would_unify
% purpose:   succeeds if X and Y WOULD unify, but doesn't actually do the
%            unification (no variables are bound by test)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
would_unify(X,Y) :- \+ \+ X = Y.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      remove_element
% purpose:   Removes the element X from a list
% remarks:   Fails if X is not an element in the list
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
remove_element(X, [X|Rest], Rest) :- !.
remove_element(X, [Y|Rest], [Y|Rest2]) :- remove_element(X, Rest, Rest2).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      replace_element(Elt, List, New, NewList)
% purpose:  Replaces the element Elt, if present in List, with the element New
% remarks:  If there are multiple occurrences of Elt, only replaces the first
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
replace_element(Elt, [Elt|Rest], New, [New|Rest]) :- !.
replace_element(Elt, [_|Rest], New, [_|Rest2]) :-
    replace_element(Elt, Rest, New, Rest2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      select_elements(List, Selector, NewList)
% purpose:  Selects all List elements for which Selector(element) succeeds.
% remarks:  If there are multiple occurrences of Elt, only replaces the first
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
select_elements([], _Selector, []).
select_elements([Element | Elements], Selector, [Element | Selected]) :-
    Test =.. [Selector, Element],
    call( Test ),
    !,
    select_elements(Elements, Selector, Selected).
select_elements([_Element | Elements], Selector, Selected) :-
    select_elements(Elements, Selector, Selected).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      call_n(+N, +Goal)
% purpose:  Call Goal with a limit on the number of solutions generated.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

call_n(1, Goal) :-
    call(Goal),
    !.
call_n(N, Goal) :-
    % Remember the counter's value in case anyone else is using it.
    ctr_is(12, CtrOrig),
    call_n_aux(N, Goal, CtrOrig).

call_n_aux(N, Goal, CtrOrig) :-
    N > 1,
    ctr_set(12, 1),
    call(Goal),
    ctr_inc(12, 1, M),
    ( M =< N ->
        true
    | otherwise ->
        ctr_set(12, CtrOrig),
        !,
        fail
    ).
% This clause is for when the Goal fails before M > N:
call_n_aux(_N, _Goal, CtrOrig) :-
    ctr_set(12, CtrOrig),
    !,
    fail.

```

```

% findall with a limit on the number of solutions generated.
findNSolutions(0, _Var, _Predicate, []).
findNSolutions(1, Var, Predicate, [Var]) :-
    call(Predicate), !.
findNSolutions(1, _Var, _Predicate, []).
findNSolutions(N, Var, Predicate, Solutions) :-
    N > 1,
    % Save the counter's value in case anyone else is using it.
    ctr_is(12, CtrOrig),
    ctr_set(12, 1),
    findall(Var,
        (Predicate, ctr_inc(12, 1, M),
         (M >= N -> ! | otherwise -> true)),
        Solutions),
    ctr_set(12, CtrOrig).

% =====
% No longer used: replaced or obsolete
% =====

% initialize all data flags
% oaa_init_flags :-
%     % set appropriate prolog flags
%     prolog_flag(fileerrors,_,on),
%     prolog_flag(syntax_errors,_,error),
%     % Let's use retractall so as to avoid unknown exceptions when tracing:
%     retractall(oaa_cache(_, _)),
%     retractall(oaa_already_loaded(_)),
%     assert(oaa_trace(off)),
%     assert(oaa_debug(off)),
%     assert(oaa_com_trace(off)),
%     tcp_trace(_,off).

```

APPENDIX A.V

Source code file named translations.pl.



```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   File       : translations.pl
%   Primary Authors   : David Martin, Adam Cheyer
%   Purpose      : Provides translations for backward compatibility with OAA 1.0
%
%   -----
%   Unpublished-rights reserved under the copyright laws of the United States.
%
%   -----
%   Unpublished Copyright (c) 1998, SRI International.
%   "Open Agent Architecture" and "OAA" are Trademarks of SRI International.
%   -----
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

% This file is loaded by facilitator code, and thus no
% module imports are needed here.

```

```

% Currently, we support a 3.0 facilitator with a mix of 3.0 and/or pre-3.0
% clients.
% A pre-3.0 facilitator with a 3.0 client is NOT supported, and probably
% never will be.

```

```

:- multifile oaa_AppDoEvent/2.

```

```

% At present we only support the case where the facilitator is 3.0, and
% the client is pre-3.0.

```

```

% Here we can ignore the languages.
oaa_event_translation(2.0, L1, 3.0, L2, Connection, Event1, Event2) :-
  oaa_event_translation(2.1, L1, 3.0, L2, Connection, Event1, Event2).
oaa_event_translation(2.1, _L1, 3.0, _L2, _Connection, Event1, Event2) :-
  ( Event1 = event(From, Contents1, Priority) ->
    Params2 = [from(From), priority(Priority)]
  | Event1 = event(From, Contents1) ->
    Params2 = [from(From)]
  | Event1 = Contents1 ->
    Params2 = []
  ),
  ( ev_trans_21_30(Contents1, Contents2) ->
    true
  | otherwise ->
    Contents2 = Contents1
  ),
  Event2 = event(Contents2, Params2).

```

```

% Here we can ignore the languages.
oaa_event_translation(3.0, L1, 2.0, L2, Connection, Event1, Event2) :-
  oaa_event_translation(3.0, L1, 2.1, L2, Connection, Event1, Event2).
oaa_event_translation(3.0, _L1, 2.1, _L2, _Connection, Event1, Event2) :-
  Event1 = event(Contents1, Params1),
  ( ev_trans_30_21(Contents1, Params1, Contents2) ->
    true
  | otherwise ->
    Contents1 = Contents2
  ),
  ( memberchk(from(KS), Params1) ->

```

```

    Event2 = event(KS, Contents2)
| otherwise ->
    Event2 = Contents2
),
!.
% Anything not specified explicitly stays the same:
oaa_event_translation(3.0, _L1, 2.1, _L2, _Connection, E1, E1).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The following could go to or from the facilitator.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ev_trans_21_30(trace_on, ev_trace_on).
ev_trans_21_30(trace_off, ev_trace_off).
ev_trans_21_30(tcp_trace_on, ev_com_trace_on).
ev_trans_21_30(tcp_trace_off, ev_com_trace_off).
ev_trans_21_30(debug_on, ev_debug_on).
ev_trans_21_30(debug_off, ev_debug_off).
ev_trans_21_30(set_timeout(N), ev_set_timeout(N)).
ev_trans_21_30(halt, ev_halt).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The following are sent only from (pre-3.0) client to facilitator.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ev_trans_21_30(post_event(Event), ev_post_event(NewEvent)) :-
    ev_trans_21_30(Event, NewEvent).
ev_trans_21_30(post_event(To, Event), ev_post_event(To, NewEvent)) :-
    ev_trans_21_30(Event, NewEvent).

ev_trans_21_30(post_query(Goal, Params),
    ev_post_solve(Goal, [reflexive(false) | NewParams])) :-
    params_trans_21_30(Params, NewParams).

% This is the message from a facilitator to its parent facilitator;
% will probably evolve:
% ev_trans_21_30(register_solvable_goals(AGL), register_solvable_goals(AGL)).
% NO, we don't want to translate this. The old form is still handled
% by the new facilitator:
% ev_trans_21_30(register_solvable_goals(GoalList, KSName),
%     ev_register_solvables(add, GoalList, KSName,
%         [if_exists(overwrite)]))].

ev_trans_21_30(solved(GoalId, FromKS, Goal, SolveParams, Solutions),
    ev_solved(GoalId, FromKS, Goal, SolveParams, Solutions)).

/* post_trigger/4: retained for backwards compatibility */
ev_trans_21_30(post_trigger(test, Type, Cond, Action), NewEvent) :-
    ev_trans_21_30(post_trigger(test, Type, unused, unused, Cond, Action),
        NewEvent).

/* post_trigger/4: retained for backwards compatibility */
ev_trans_21_30(post_trigger(data, Type, Cond, Action), NewEvent) :-
    ev_trans_21_30(post_trigger(data, Type,
        [on_write, on_write_replace, on_replace],
        Cond, true, Action), NewEvent).

```



```

/* post_trigger/4: retained for backwards compatibility */
ev_trans_21_30(post_trigger(event, Type, Cond, Action), NewEvent) :-
    ev_trans_21_30(post_trigger(event, Type, [on_receive], Cond, true, Action),
        NewEvent).

ev_trans_21_30(post_trigger(Kind,Recur,OpMask,Template,Test,Action),
    ev_post_trigger_update(add,Mode,Condition,NewAction,Params)) :-
    ( Kind == test -> Mode = task
    | Kind == event -> Mode = comm
    | Kind == alarm -> Mode = time
    | otherwise -> Mode = Kind ),
    ( Recur == whenever ->
        Recurrence = [recurrence(whenever)]
    | otherwise ->
        Recurrence = [recurrence(when)]
    ),
    template_trans_21_30(Kind, Template, Condition),
    ( var(Test) -> TestParam = [] | otherwise -> TestParam = [test(Test)] ),
    ( Mode == data, ev_trans_21_30(Action, NewAction) -> true
    | otherwise -> NewAction = Action ),
    opmask_trans_21_30(OpMask, OpParam),
    ( Mode == data ->
        oaa_Id(FacId),
        Addr = [address(FacId)]
    | otherwise ->
        Addr = []
    ),
    append([Addr, [reply(none), reflexive(false)],
        Recurrence, TestParam, OpParam], Params).
ev_trans_21_30(post_trigger(KS, Kind,Recur,OpMask,Template,Test,Action),
    ev_post_trigger_update(add,Type,Condition,NewAction,Params)) :-
    ( Kind == test -> Type = task
    | Kind == event -> Type = comm
    | Kind == alarm -> Type = time
    | otherwise -> Type = Kind),
    ( Recur == whenever ->
        Recurrence = recurrence(whenever)
    | otherwise ->
        Recurrence = recurrence(when)
    ),
    template_trans_21_30(Kind, Template, Condition),
    ( var(Test) -> TestParam = [] | otherwise -> TestParam = [test(Test)] ),
    oaa_Id(FacId),
    ( KS == FacId, ev_trans_21_30(Action, NewAction) -> true
    | otherwise -> NewAction = Action ),
    opmask_trans_21_30(OpMask, OpParam),
    append([[address(KS), reply(none), reflexive(false)],
        Recurrence, TestParam, OpParam],
        Params).

params_trans_21_30([], []).
params_trans_21_30([Param | Params], [NewParam | NewParams]) :-
    ( param_trans_21_30(Param, NewParam) ->
        true
    | otherwise ->
        NewParam = Param
    ),

```

```

params_trans_21_30(Params, NewParams).

param_trans_21_30(cache, cache(true)).
param_trans_21_30(solution_limit(N), solution_limit(N)).
param_trans_21_30(reflexive, reflexive(true)).
param_trans_21_30(address(A), address(NewA)) :-
    ( is_list(A) -> NewA = A | otherwise -> NewA = [A] ).
param_trans_21_30(broadcast, reply(none)).
param_trans_21_30(asynchronous, reply(asynchronous)).
% @@DLM: is this handled?:
param_trans_21_30(test(T), test(T)).
param_trans_21_30(level_limit(N), level_limit(N)).
param_trans_21_30(time_limit(N), time_limit(N)).
% @@DLM: NOT HANDLED!:
param_trans_21_30(and_parallel, and_parallel).
param_trans_21_30(or_parallel, or_parallel).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The following could go to or from the facilitator.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ev_trans_30_21(ev_trace_on, _EvParams, trace_on).
ev_trans_30_21(ev_trace_off, _EvParams, trace_off).
ev_trans_30_21(ev_com_trace_on, _EvParams, tcp_trace_on).
ev_trans_30_21(ev_com_trace_off, _EvParams, tcp_trace_off).
ev_trans_30_21(ev_debug_on, _EvParams, debug_on).
ev_trans_30_21(ev_debug_off, _EvParams, debug_off).
ev_trans_30_21(ev_set_timeout(N), _EvParams, set_timeout(N)).
ev_trans_30_21(ev_halt, _EvParams, halt).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The following are sent only from facilitator to client.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ev_trans_30_21(
    ev_solve(ID, Goal, NewParams),
    _EventParams,
    solve(ID, Goal, Params)) :-
    params_trans_30_21(Params, NewParams).

ev_trans_30_21(ev_reply_solved(_, Solved, Goal, SolveParams, Solutions),
    _EventParams,
    solved(FromKS, Goal, SolveParams, Solutions)) :-
    ( Solved = [FromKS] ->
        true
    | otherwise ->
        FromKS = Solved
    ).

% OBSOLETE: forget these:
% ev_trans_30_21(add_trigger(data, Type, Cond, Action),
% ev_trans_30_21(add_trigger(event, Type, Cond, Action)
% ev_trans_30_21(add_trigger(test, Type, Cond, Action)
% @@DLM: Don't think this is needed:
% ev_trans_30_21(inform_ui(TypeInfo, Result), ))

ev_trans_30_21(

```

```

ev_update_trigger(_ID, add, Type, Condition, Action, TrigParams),
_EventParams,
add_trigger(Kind, Recur, OpMask, Template, Test, Action) :-
( Type = task -> Kind == test
| Type = comm-> Kind == event
| Type = time-> Kind == alarm
| otherwise -> Type = Kind ),
( memberchk(recurrence(whenever), TrigParams) ->
Recur = whenever
| otherwise ->
Recur = when
),
Template = Condition,
( memberchk(test(Test), TrigParams) -> true | otherwise -> Test = _ ),
( memberchk(on(OpParam), TrigParams) ->
true
| otherwise ->
OpParam = _
),
opmask_trans_30_21(OpParam, OpMask),
( memberchk(test(Test), TrigParams) -> true | true ).

params_trans_30_21([], []).
params_trans_30_21([Param | Params], [NewParam | NewParams]) :-
( param_trans_30_21(Param, NewParam) ->
true
| otherwise ->
NewParam = Param
),
params_trans_30_21(Params, NewParams).

param_trans_30_21(cache(true), cache).
param_trans_30_21(solution_limit(N), solution_limit(N)).
param_trans_30_21(reflexive(true), reflexive).
% @@DLM: double-check this:
param_trans_30_21(address(A), address(A)).
param_trans_30_21(reply(none), broadcast).
param_trans_30_21(reply(asynchronous), asynchronous).
% @@DLM: is this handled?:
param_trans_30_21(test(T), test(T)).
param_trans_30_21(level_limit(N), level_limit(N)).
param_trans_30_21(time_limit(N), time_limit(N)).
% @@DLM: NOT HANDLED!:
param_trans_30_21(and_parallel, and_parallel).
param_trans_30_21(or_parallel, or_parallel).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The following are sent only from a pre-3.0 facilitator to a client.
% Backwards compatibility not currently supported.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ev_trans_21_30(solved(FromKS, Goal, SolveParams, Solutions),
% ev_reply_solved([FromKS], Solvers, Goal, SolveParams, Solutions)) :-
% ( Solutions == [] ->
% Solvers = []
% | otherwise ->

```

```

%   Solvers = [FromKS]
%   ),
%   ( memberchk(get_address(FromKS), SolveParams) ->
%   true
%   | otherwise ->
%   FromKS = unknown
%   ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Auxiliary procedures.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Returns either a Singleton list or an empty list.
opmask_trans_21_30(OpMask, []) :-
    var(OpMask),
    !.
opmask_trans_21_30(OpMask, OpParam) :-
    \+ is_list(OpMask),
    !,
    opmask_trans_21_30([OpMask], OpParam).
opmask_trans_21_30([], []).
opmask_trans_21_30([Elt | Rest], [EltTrans | RestTrans]) :-
    opmask_elt_trans_21_30(Elt, EltTrans),
    !,
    opmask_trans_21_30(Rest, RestTrans).
opmask_trans_21_30([_Elt | Rest], RestTrans) :-
    !,
    opmask_trans_21_30(Rest, RestTrans).
opmask_elt_trans_21_30(on_send, on(send)).
opmask_elt_trans_21_30(on_receive, on(receive)).
opmask_elt_trans_21_30(on_write, on(add)).
opmask_elt_trans_21_30(on_retract, on(remove)).
opmask_elt_trans_21_30(on_replace, on(replace)).
% This one probably doesn't have a precise translation:
opmask_elt_trans_21_30(on_write_replace, on(replace)).

opmask_trans_30_21(OpMask, OpMask) :-
    var(OpMask),
    !.
opmask_trans_30_21(OpMask, OpParam) :-
    \+ is_list(OpMask),
    !,
    opmask_trans_30_21([OpMask], OpParam).
opmask_trans_30_21([], []).
opmask_trans_30_21([Elt | Rest], [EltTrans | RestTrans]) :-
    opmask_elt_trans_30_21(Elt, EltTrans),
    !,
    opmask_trans_30_21(Rest, RestTrans).
opmask_trans_30_21([_Elt | Rest], RestTrans) :-
    !,
    opmask_trans_30_21(Rest, RestTrans).
opmask_elt_trans_30_21(on_send, on_send).
opmask_elt_trans_30_21(on_receive, on_receive).
opmask_elt_trans_30_21(on_add, on_write).
opmask_elt_trans_30_21(on_remove, on_retract).
opmask_elt_trans_30_21(on_replace, on_replace).
% This one probably doesn't have a precise translation:

```

```

opmask_elt_trans_30_21(on(replace), on_write_replace).

template_trans_21_30(data,
                    data(ksdata, [AgentId,Status,Solvables,Name]),
                    agent_data(AgentId,Status,Solvables,Name)) :-
    !.
template_trans_21_30(data, Template, Template) :-
    !.
template_trans_21_30(event, Template, Condition) :-
    !,
    ev_trans_21_30(Template, Condition).
template_trans_21_30(_, Template, Template).

*****
% Event handlers for selected pre-3.0 events.
%
% In these cases, this approach is easier than providing an event
% translation.
*****

oaa_AppDoEvent(register_solvable_goals(GoalList), Params) :-
    memberchk( connection_id(Connection), Params),
    % This hack inherited from b.pl:
    oaa_AppDoEvent(register_solvable_goals(GoalList, Connection),
                  Params).

oaa_AppDoEvent(register_solvable_goals(GoalList, Name), Params) :-
    memberchk( connection_id(Connection), Params),
    update_connected(Connection, [oaa_name(Name)]),
    icl_ConvertSolvables(GoalList, Solvables),
    oaa_AppDoEvent(ev_register_solvables(add,Solvables,Name,[if_exists(overwri
te)]),
                  Params).

oaa_AppDoEvent(can_solve(Goal), EvParams) :-
    memberchk(from(KS), EvParams),
    findall(SomeKS, choose_ks_for_goal(KS, Goal, _, [], SomeKS, _), AgentList),
    oaa_PostEvent(return_can_solve(Goal, AgentList), [address(KS)]).

*****
% BB events
*****

oaa_AppDoEvent(write_bb(ksdata, [Id,Status,Solvables,Name]),
              EvParams) :-
    !,
    ( var(Solvables) ->
      % (Surely this never happens.)
      oaa:oaa_add_data_local(agent_data(Id,Status,Solvables,Name), [from(Id)])
    | otherwise ->
      icl_ConvertSolvables(Solvables, FormalSolvables),
      oaa_AppDoEvent(ev_register_solvables(add,FormalSolvables,Name,
                                          [if_exists(overwrite)]),
                    [from(Id) | EvParams])
    ).
oaa_AppDoEvent(write_bb(oaa_version, V), EvParams) :-

```

```

!,
memberchk(from(Id), EvParams),
% oaa:oa_add_data_local(data(oaa_Version, V), [from(Id)]),
com_GetInfo(ConnectionId, oaa_id(Id)),
com_AddInfo(ConnectionId, agent_version(V)).
oaa_AppDoEvent(write_bb(language, Language), EvParams) :-
!,
memberchk(from(Id), EvParams),
com_GetInfo(ConnectionId, oaa_id(Id)),
com_AddInfo(ConnectionId, agent_language(Language)).
oaa_AppDoEvent(write_bb(kshost, Host), EvParams) :-
!,
memberchk(from(Id), EvParams),
oaa:oa_solve_local(agent_data(Id, _, _, Name), []),
oaa:oa_add_data_local(agent_host(Id, Name, Host),
[from(Id) | EvParams]).
oaa_AppDoEvent(write_bb(Item, Data), EvParams) :-
!,
memberchk(from(Id), EvParams),
oaa:oa_add_data_local(data(Item, Data), [from(Id)]).

oaa_AppDoEvent(write_once_bb(Item, Data), EvParams) :-
(Item = ksdata ; Item = oaa_version ; Item = language ; Item = kshost),
!,
oaa_AppDoEvent(write_bb(Item, Data), [single_value(true) | EvParams]).
oaa_AppDoEvent(write_once_bb(Item, Data), EvParams) :-
!,
memberchk(from(Id), EvParams),
oaa:oa_add_data_local(data(Item, Data), [from(Id), single_value(true)]).

oaa_AppDoEvent(write_replace_bb(Item, Data), EvParams) :-
(Item = ksdata ; Item = oaa_version ; Item = language ; Item = kshost),
!,
oaa_AppDoEvent(write_bb(Item, Data), [unique_values(true) | EvParams]).
oaa_AppDoEvent(write_replace_bb(Item, Data), EvParams) :-
!,
memberchk(from(Id), EvParams),
oaa:oa_add_data_local(data(Item, Data), [from(Id), unique_values(true)]).

oaa_AppDoEvent(replace_bb(ksdata, [A,open,C,Name], [A,ready,C,Name]),
EvParams) :-
!,
oaa_AppDoEvent(ev_ready(Name), EvParams).
oaa_AppDoEvent(replace_bb(ksdata, [Id,Status,Solvables,Name],
[NewId,NewStatus,NewSolvables,NewName]),
EvParams) :-
!,
( var(NewSolvables) ->
oaa:oa_replace_data_local(agent_data(Id,Status,Solvables,Name),
[from(Id), with(agent_data(NewId,NewStatus,NewSolvables,NewName))])
| otherwise ->
' icl_ConvertSolvables(NewSolvables, FormalSolvables),
oaa_AppDoEvent(ev_register_solvables(add,FormalSolvables,NewName,
[if_exists(overwrite)]),
[from(NewId) | EvParams])
).
oaa_AppDoEvent(replace_bb(Item,OldData,NewData), EvParams) :-

```

```

!,
memberchk(from(Id), EvParams),
oaa:oaa_replace_data_local(data(Item,OldData),
                           [from(Id),with(data(Item,NewData))]).

% @@DLM: May need more special-purpose clauses starting here:
oaa_AppDoEvent(retract_bb(Item,Data), EvParams) :-
!,
memberchk(from(Id), EvParams),
oaa:oaa_remove_data_local(data(Item,Data), [from(Id)]).

oaa_AppDoEvent(read_bb(ksdata,[AgentId,Status,Solvables,Name]), EvParams) :-
!,
memberchk(from(Id), EvParams),
findall(read_bb(ksdata,[AgentId,Status,Solvables,Name]),
        oaa:oaa_solve_local(agent_data(AgentId,Status,Solvables,Name), []),
        Solutions),
oaa_simplify_ksdata(Solutions, Simplified),
oaa_PostEvent(return_read_bb(Simplified), [address(Id)]).
oaa_AppDoEvent(read_bb(KS,kshost,Host), EvParams) :-
!,
memberchk(from(Id), EvParams),
findall(read_bb(KS, kshost, Host),
        oaa:oaa_solve_local(agent_host(KS,_,Host), []),
        Solutions),
oaa_PostEvent(return_read_bb(Solutions), [address(Id)]).
oaa_AppDoEvent(read_bb(oaa_version,V), EvParams) :-
!,
memberchk(from(Id), EvParams),
% Not sure if this works (but this clause is probably never called):
findall(read_bb(oaa_version, V),
        ( com_GetInfo(ConnectionId, oaa_id(_)),
          com_GetInfo(ConnectionId, agent_version(V)) ),
        Solutions),
oaa_PostEvent(return_read_bb(Solutions), [address(Id)]).

oaa_AppDoEvent(read_bb(KS,oaa_version,V), EvParams) :-
!,
memberchk(from(Id), EvParams),
findall(read_bb(KS, oaa_version, V),
        ( com_GetInfo(ConnectionId, oaa_id(KS)),
          com_GetInfo(ConnectionId, agent_version(V)) ),
        Solutions),
oaa_PostEvent(return_read_bb(Solutions), [address(Id)]).
oaa_AppDoEvent(read_bb(Item,Data), EvParams) :-
!,
memberchk(from(Id), EvParams),
findall(read_bb(Item,Data),
        oaa:oaa_solve_local(data(Item,Data), []),
        Solutions),
oaa_PostEvent(return_read_bb(Solutions), [address(Id)]).
% @@The owner parameter isn't implemented yet for solve!
oaa_AppDoEvent(read_bb(_KS, Item,Data), EvParams) :-
!,
memberchk(from(Id), EvParams),
findall(read_bb(Item,Data),
        oaa:oaa_solve_local(data(Item,Data), []),

```

```
Solutions),
oaa_PostEvent(return_read_bb(Solutions), [address(Id)]).

oaa_simplify_ksdata([], []).
oaa_simplify_ksdata([KSData | Rest], [Simplified | RestSimp]) :-
    KSData = read_bb(ksdata, [A, B, Solvables, D]),
    icl_ConvertSolvables(SimplifiedSolvables, Solvables),
    Simplified = read_bb(ksdata, [A, B, SimplifiedSolvables, D]),
    oaa_simplify_ksdata(Rest, RestSimp).
```


IN THE CLAIMS:

Sub
A1

1 1. A computer-implemented method for communication and cooperative task
 2 completion among a plurality of distributed electronic agents, comprising the
 3 acts of:
 4 registering a description of each active client agent's functional capabilities, using an
 5 expandable, platform-independent, inter-agent language;
 6 receiving a request for service as a base goal in the inter-agent language, in the form
 7 of an arbitrarily complex goal expression; and
 8 dynamically interpreting the goal expression, said act of interpreting further
 9 comprising:
 10 generating one or more sub-goals using the inter-agent language; and
 11 dispatching each of the sub-goals to a selected client agent for performance,
 12 based on a match between the sub-goal being dispatched and the
 13 registered functional capabilities of the selected client agent.

1 2. A computer-implemented method as recited in claim 1, further including the
 2 following acts of:
 3 receiving a new request for service as a base goal using the inter-agent language, in
 4 the form of another arbitrarily complex goal expression, from at least one of
 5 the selected client agents in response to the sub-goal dispatched to said agent;
 6 and
 7 recursively applying the last step of claim 1 in order to perform the new request for
 8 service.

1 3. A computer implemented method as recited in claim 2 wherein the act
 2 of registering a specific agent further includes:
 3 invoking the specific agent in order to activate the specific agent;
 4 instantiating an instance of the specific agent; and
 5 transmitting the new agent profile from the specific agent to the facilitator
 6 agent in response to the instantiation of the specific agent.

Sub
B1

1 4. A computer implemented method as recited in claim 1 further
 2 including the act of deactivating a specific client agent no longer available to provide
 3 services by deleting the registration of the specific client agent.

1 5. A computer implemented method as recited in claim 1 further
 2 comprising the act of providing an agent registry data structure.

092519-01059
665010-86152260

655070-8675260

1 6. A computer implemented method as recited in claim 5 wherein the
2 agent registry data structure includes at least one symbolic name for each active agent.

1 7. A computer implemented method as recited in claim 5 wherein the
2 agent registry data structure includes at least one data declaration for each active
3 agent.

1 8. A computer implemented method as recited in claim 5 wherein the
2 agent registry data structure includes at least one trigger declaration for one active
3 agent.

1 9. A computer implemented method as recited in claim 5 wherein the
2 agent registry data structure includes at least one task declaration, and process
3 characteristics for each active agent.

1 10. A computer implemented method as recited in claim 5 wherein the
2 agent registry data structure includes at least one process characteristic for each active
3 agent.

1 11. A computer implemented method as recited in claim 1 further
2 comprising the act of establishing communication between the plurality of distributed
3 agents.

1 12. A computer implemented method as recited in claim 1 further
2 comprising the acts of:

3 receiving a request for service in a second language differing from the inter-
4 agent language;

5 selecting a registered agent capable of converting the second language into the
6 inter-agent language; and

7 forwarding the request for service in a second language to the registered agent
8 capable of converting the second language into the inter-agent language, implicitly
9 requesting that such a conversion be performed and the results returned.

1 13. A computer implemented method as recited in claim 12 wherein the
2 request includes a natural language query, and the registered agent capable of
3 converting the second language into the inter-agent language service is a natural
4 language agent.

1 14. A computer implemented method as recited in claim 13 wherein the
2 natural language query was generated by a user interface agent.

1 15. A computer implemented method as recited in claim 1, wherein the
2 base goal requires setting a trigger having conditional functionality and consequential
3 functionality.

1 16. A computer implemented method as recited in claim 15 wherein the
2 trigger is an outgoing communications trigger, the computer implemented method
3 further including the acts of:

4 monitoring all outgoing communication events in order to determine whether a
5 specific outgoing communication event has occurred; and

6 in response to the occurrence of the specific outgoing communication event,
7 performing the particular action defined by the trigger.

1 17. A computer implemented method as recited in claim 15 wherein the
2 trigger is an incoming communications trigger, the computer implemented method
3 further including the acts of:

4 monitoring all incoming communication events in order to determine whether
5 a specific incoming communication event has occurred; and

6 in response to the occurrence of a specific incoming communication event
7 satisfying the trigger conditional functionality, performing the particular
8 consequential functionality defined by the trigger.

1 18. A computer implemented method as recited in claim 15 wherein the
2 trigger is a data trigger, the computer implemented method further including the acts
3 of:

4 monitoring a state of a data repository; and

5 in response to a particular state event satisfying the trigger conditional
6 functionality, performing the particular consequential functionality defined by the
7 trigger.

1 19. A computer implemented method as recited in claim 15 wherein the
2 trigger is a time trigger, the computer implemented method further including the acts
3 of:

4 monitoring for the occurrence of a particular time condition; and

5 in response to the occurrence of a particular time condition satisfying the
6 trigger conditional functionality, performing the particular consequential functionality
7 defined by the trigger.

1 20. A computer implemented method as recited in claim 15 wherein the
2 trigger is installed and executed within the facilitator agent.

665010 86152260

W

665010-86T52260

1 21. A computer implemented method as recited in claim 15 wherein the
2 trigger is installed and executed within a first service-providing agent.

1 22. A computer implemented method as recited in claim 15 wherein the
2 conditional functionality of the trigger is installed on a facilitator agent.

1 23. A computer implemented method as recited in claim 22 wherein the
2 consequential functionality is installed on a specific service-providing agent other
3 than a facilitator agent.

1 24. A computer implemented method as recited in claim 15 wherein the
2 conditional functionality of the trigger is installed on a specific service-providing
3 agent other than a facilitator agent.

1 25. A computer implemented method as recited in claim 15 wherein the
2 consequential functionality of the trigger is installed on a facilitator agent.

1 26. A computer implemented method as recited in claim 1 wherein the
2 base goal is a compound goal having sub-goals separated by operators.

1 27. A computer implemented method as recited in claim 26 wherein the
2 type of available operators includes a conjunction operator, a disjunction operator,
3 and a conditional execution operator.

1 28. A computer implemented method as recited in claim 27 wherein the type
2 of available operators further includes a parallel disjunction operator that indicates that
3 disjunct goals are to be performed by different agents.

65070-86T52260

65070 8675260

1 29. A computer program stored on a computer readable medium, the
2 computer program executable to facilitate cooperative task completion within a
3 distributed computing environment, the distributed computing environment including
4 a plurality of autonomous electronic agents, the distributed computing environment
5 supporting an Interagent Communication Language, the computer program
6 comprising computer executable instructions for:

7 providing an agent registry that declares capabilities of service-providing
8 electronic agents currently active within the distributed computing environment;

9 interpreting a service request in order to determine a base goal that may be a
10 compound, arbitrarily complex base goal, the service request adhering to an
11 Interagent Communication Language (ICL), the act of interpreting including the sub-
12 acts of:

13 determining any task completion advice provided by the base goal, and
14 determining any task completion constraints provided by the base goal;
15 constructing a base goal satisfaction plan including the sub-acts of:
16 determining whether the requested service is available,
17 determining sub-goals required in completing the base goal,
18 selecting service-providing electronic agents from the agent registry
19 suitable for performing the determined sub-goals, and
20 ordering a delegation of sub-goal requests to best complete the
21 requested service; and
22 implementing the base goal satisfaction plan.

1 30. A computer program as recited in claim 29 wherein the computer
2 executable instruction for providing an agent registry includes the following computer
3 executable instructions for registering a specific service-providing electronic agent
4 into the agent registry:

5 establishing a bi-directional communications link between the specific agent
6 and a facilitator agent controlling the agent registry;

7 providing a new agent profile to the facilitator agent, the new agent profile
8 defining publicly available capabilities of the specific agent; and

9 registering the specific agent together with the new agent profile within the
10 agent registry, thereby making available to the facilitator agent the capabilities of the
11 specific agent.

655070 8675260

1 31. A computer program as recited in claim 30 wherein the computer
2 executable instruction for registering a specific agent further includes:
3 invoking the specific agent in order to activate the specific agent;
4 instantiating an instance of the specific agent; and
5 transmitting the new agent profile from the specific agent to the facilitator
6 agent in response to the instantiation of the specific agent.

1 32. A computer program as recited in claim 29 wherein the computer
2 executable instruction for providing an agent registry includes a computer executable
3 instruction for removing a specific service-providing electronic agent from the
4 registry upon determining that the specific agent is no longer available to provide
5 services.

1 33. A computer program as recited in claim 29 wherein the provided agent
2 registry includes a symbolic name, a unique address, data declarations, trigger
3 declarations, task declarations, and process characteristics for each active agent.

1 34. A computer program as recited in claim 29 further including computer
2 executable instructions for receiving the service request via a communications link
3 established with a client.

1 35. A computer program as recited in claim 29 wherein the computer
2 executable instruction for providing a service request includes instructions for:
3 receiving a non-ICL format service request;
4 selecting an active agent capable of converting the non-ICL formal service
5 request into an ICL format service request;
6 forwarding the non-ICL format service request to the active agent capable of
7 converting the non-ICL format service request, together with a request that such
8 conversion be performed; and
9 receiving an ICL format service request corresponding to the non-ICL format
10 service request.

1 36. A computer program as recited in claim 35 wherein the non-ICL
2 format service request includes a natural language query, and the active agent capable
3 of converting the non-ICL formal service request into an ICL format service request is
4 a natural language agent.

1 37. A computer program as recited in claim 36 wherein the natural
2 language query is generated by a user interface agent.

665070 BERTS26B

1 38. A computer program as recited in claim 29, the computer program
2 further including computer executable instructions for implementing a base goal that
3 requires setting a trigger having conditional and consequential functionality.

1 39. A computer program as recited in claim 38 wherein the trigger is an
2 outgoing communications trigger, the computer program further including computer
3 executable instructions for:

- 4 monitoring all outgoing communication events in order to determine whether a
- 5 specific outgoing communication event has occurred; and
- 6 in response to the occurrence of the specific outgoing communication event,
- 7 performing the particular action defined by the trigger.

1 40. A computer program as recited in claim 38 wherein the trigger is an
2 incoming communications trigger, the computer program further including computer
3 executable instructions for:

- 4 monitoring all incoming communication events in order to determine whether
- 5 a specific incoming communication event has occurred; and
- 6 in response to the occurrence of the specific incoming communication event,
- 7 performing the particular action defined by the trigger.

1 41. A computer program as recited in claim 38 wherein the trigger is a data
2 trigger, the computer program further including computer executable instructions for:

- 3 monitoring a state of a data repository; and
- 4 in response to a particular state event, performing the particular action defined
- 5 by the trigger.

1 42. A computer program as recited in claim 38 wherein the trigger is a
2 time trigger, the computer program further including computer executable instructions
3 for:

- 4 monitoring for the occurrence of a particular time condition; and
- 5 in response to the occurrence of the particular time condition, performing the
- 6 particular action defined by the trigger.

1 43. A computer program as recited in claim 38 further including computer
2 executable instructions for installing and executing the trigger within the facilitator
3 agent.

1 44. A computer program as recited in claim 38 further including computer
2 executable instructions for installing and executing the trigger within a first service-
3 providing agent.

65010 86752260

Sub
or

1 45. A computer program as recited in claim 29 further including computer
2 executable instructions for interpreting compound goals having sub-goals separated
3 by operators.

1 46. A computer program as recited in claim 45 wherein the type of
2 available operators includes a conjunction operator, a disjunction operator, and a
3 conditional execution operator.

1 47. A computer program as recited in claim 46 wherein the type of
2 available operators further includes a parallel disjunction operator that indicates that
3 disjunct goals are to be performed by different agents.

1 48. An Interagent Communication Language (ICL) providing a basis for
2 facilitated cooperative task completion within a distributed computing environment
3 having a facilitator agent and a plurality of autonomous service-providing electronic
4 agents, the ICL enabling agents to perform queries of other agents, exchange
5 information with other agents, set triggers within other agents, an ICL syntax
6 supporting compound goal expressions such that goals within a single request
7 provided according to the ICL syntax may be coupled by a conjunctive operator, a
8 disjunctive operator, a conditional execution operator, and a parallel disjunctive
9 operator parallel disjunctive operator that indicates that disjunct goals are to be
10 performed by different agents.

1 49. An ICL as recited in claim 48, wherein the ICL is computer platform
2 independent.

1 50. An ICL as recited in claim 48 wherein the ICL is independent of
2 computer programming languages which the plurality of agents are programmed in.

1 51. An ICL as recited in claim 48 wherein the ICL syntax supports explicit
2 task completion constraints within goal expressions.

1 52. An ICL as recited in claim 51 wherein possible types of task
2 completion constraints include use of specific agent constraints and response time
3 constraints.

1 53. An ICL as recited in claim 51 wherein the ICL syntax supports explicit
2 task completion advisory suggestions within goal expressions.

1 54. An ICL as recited in claim 48 wherein the ICL syntax supports explicit
2 task completion advisory suggestions within goal expressions.

65070-136752260

1 55. An ICL as recited in claim 48 wherein each autonomous service-
2 providing electronic agent defines and publishes a set of capability declarations or
3 solvables, expressed in ICL, that describes services provided by such electronic agent.

1 56. An ICL as recited in claim 55 wherein an electronic agent's solvables
2 define an interface for the electronic agent.

1 57. An ICL as recited in claim 56 wherein the facilitator agent maintains
2 an agent registry making available a plurality of electronic agent interfaces.

1 58. An ICL as recited in claim 57 wherein the possible types of solvables
2 includes procedure solvables, a procedure solvable operable to implement a procedure
3 such as a test or an action.

1 59. An ICL as recited in claim 58 wherein the possible types of solvables
2 further includes data solvables, a data solvable operable to provide access to a
3 collection of data.

1 60. An ICL as recited in claim 58 wherein the possible types of solvables
2 includes data solvables, a data solvable operable to provide access to a collection of
3 data.

1 61. A facilitator agent arranged to coordinate cooperative task completion
2 within a distributed computing environment having a plurality of autonomous service-
3 providing electronic agents, the facilitator agent comprising:

4 an agent registry that declares capabilities of service-providing electronic
5 agents currently active within the distributed computing environment; and

6 a facilitating engine operable to parse a service request in order to interpret a
7 compound goal set forth therein, the compound goal including both local and global
8 constraints and control parameters, the service request formed according to an
9 Interagent Communication Language (ICL), the facilitating engine further operable to
10 construct a goal satisfaction plan specifying the coordination of a suitable delegation
11 of sub-goal requests to complete the requested service satisfying both the local and
12 global constraints and control parameters.

1 62. A facilitator agent as recited in claim 61, wherein the facilitating
2 engine is capable of modifying the goal satisfaction plan during execution, the
3 modifying initiated by events such as new agent declarations within the agent registry,
4 decisions made by remote agents, and information provided to the facilitating engine
5 by remote agents.

65070-3675250

1 63. A facilitator agent as recited in claim 61 wherein the agent registry
2 includes a symbolic name, a unique address, data declarations, trigger declarations,
3 task declarations, and process characteristics for each active agent.

1 64. A facilitator agent as recited in claim 61 wherein the facilitating engine
2 is operable to install a trigger mechanism requesting that a certain action be taken
3 when a certain set of conditions are met.

1 65. A facilitator agent as recited in claim 64 wherein the trigger
2 mechanism is a communication trigger that monitors communication events and
3 performs the certain action when a certain communication event occurs.

1 66. A facilitator agent as recited in claim 64 wherein the trigger
2 mechanism is a data trigger that monitors a state of a data repository and performs the
3 certain action when a certain data state is obtained.

1 67. A facilitator agent as recited in claim 66 wherein the data repository is
2 local to the facilitator agent.

1 68. A facilitator agent as recited in claim 66 wherein the data repository is
2 remote from the facilitator agent.

1 69. A facilitator agent as recited in claim 64 wherein the trigger
2 mechanism is a task trigger having a set of conditions.

1 70. A facilitator agent as recited in claim 61, the facilitator agent further
2 including a global database accessible to at least one of the service-providing
3 electronic agents.

1 71. A software-based, flexible computer architecture for communication
2 and cooperation among distributed electronic agents, the architecture contemplating a
3 distributed computing system comprising:

4 a plurality of service-providing electronic agents; and
5 a facilitator agent in bi-directional communications with the plurality of
6 service-providing electronic agents, the facilitator agent including:

7 an agent registry that declares capabilities of service-providing
8 electronic agents currently active within the distributed computing
9 environment;

10 a facilitating engine operable to parse a service request in order
11 to interpret an arbitrarily complex goal set forth therein, the facilitating
12 engine further operable to construct a goal satisfaction plan including

665070 B6T52260

13 the coordination of a suitable delegation of sub-goal requests to best
14 complete the requested service.

1 72. A computer architecture as recited in claim 71, wherein the basis for
2 the computer architect is an Interagent Communication Language (ICL) enabling
3 agents to perform queries of other agents, exchange information with other agents,
4 and set triggers within other agents, the ICL further defined by an ICL syntax
5 supporting compound goal expressions such that goals within a single request
6 provided according to the ICL syntax may be coupled by a conjunctive operator, a
7 disjunctive operator, a conditional execution operator, and a parallel disjunctive
8 operator parallel disjunctive operator that indicates that disjunct goals are to be
9 performed by different agents.

1 73. A computer architecture as recited in claim 72, wherein the ICL is
2 computer platform independent.

1 74. A computer architecture as recited in claim 73 wherein the ICL is
2 independent of computer programming languages in which the plurality of agents are
3 programmed.

1 75. A computer architecture as recited in claim 73 wherein the ICL syntax
2 supports explicit task completion constraints within goal expressions.

1 76. A computer architecture as recited in claim 75 wherein possible types
2 of task completion constraints include use of specific agent constraints and response
3 time constraints.

1 77. A computer architecture as recited in claim 75 wherein the ICL syntax
2 supports explicit task completion advisory suggestions within goal expressions.

1 78. A computer architecture as recited in claim 73 wherein the ICL syntax
2 supports explicit task completion advisory suggestions within goal expressions.

1 79. A computer architecture as recited in claim 73 wherein each
2 autonomous service-providing electronic agent defines and publishes a set of
3 capability declarations or solvables, expressed in ICL, that describes services
4 provided by such electronic agent.

1 80. A computer architecture as recited in claim 79 wherein an electronic
2 agent's solvables define an interface for the electronic agent.

1 81. A computer architecture as recited in claim 80 wherein the possible
2 types of solvables includes procedure solvables, a procedure solvable operable to
3 implement a procedure such as a test or an action.

1 82. A computer architecture as recited in claim 81 wherein the possible
2 types of solvables further includes data solvables, a data solvable operable to provide
3 access to a collection of data.

1 83. A computer architecture as recited in claim 82 wherein the possible
2 types of solvables includes a data solvable operable to provide access
3 to modify a collection of data.

1 84. A computer architecture as recited in claim 71 wherein the planning
2 component of the facilitating engine are distributed across at least two
3 computer processes.

1 85. A computer architecture as recited in claim 71 wherein the execution
2 component of the facilitating engine is distributed across at least two
3 computer processes.

1 86. A data wave carrier providing a transport mechanism for information
2 communication in a distributed computing environment having at least one facilitator
3 agent and at least one active client agent, the data wave carrier comprising a signal
4 representation of an inter-agent language description of an active client agent's
5 functional capabilities.

1 87. A data wave carrier as recited in claim 85, the data wave carrier further
2 comprising a signal representation of a request for service in the inter-agent language
3 from a first agent to a second agent.

1 88. A data wave carrier as recited in claim 85, the data wave carrier further
2 comprising a signal representation of a goal dispatched to an agent for performance
3 from a facilitator agent.

1 89. A data wave carrier as recited in claim 88 wherein a later state of the
2 data wave carrier comprises a signal representation of a response to the dispatched
3 goal including results and/or a status report from the agent for performance to the
4 facilitator agent.

del 83 7

655070 BERTS250

del 83 7

Software-Based Architecture for Communication and Cooperation Among
Distributed Electronic Agents

ABSTRACT

5 A highly flexible, software-based architecture is disclosed for constructing
distributed systems. The architecture supports cooperative task completion by
flexible, dynamic configurations of autonomous electronic agents. Communication
and cooperation between agents are brokered by one or more facilitators, which are
responsible for matching requests, from users and agents, with descriptions of the
10 capabilities of other agents. It is not generally required that a user or agent know the
identities, locations, or number of other agents involved in satisfying a request, and
relatively minimal effort is involved in incorporating new agents and "wrapping"
legacy applications. Extreme flexibility is achieved through an architecture organized
around the declaration of capabilities by service-providing agents, the construction of
15 arbitrarily complex goals by users and service-requesting agents, and the role of
facilitators in delegating and coordinating the satisfaction of these goals, subject to
advice and constraints that may accompany them. Additional mechanisms and
features include facilities for creating and maintaining shared repositories of data; the
use of triggers to instantiate commitments within and between agents; agent-based
20 provision of multi-modal user interfaces, including natural language; and built-in
support for including the user as a privileged member of the agent community.
Specialized embodiments providing enhanced scalability are also described.

65070-8675250

DECLARATION AND POWER OF ATTORNEY
FOR ORIGINAL U.S. PATENT APPLICATION

Attorney's Docket No. SRI1P016

As a below-named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name.

I believe that I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled: SOFTWARE-BASED ARCHITECTURE FOR COMMUNICATION AND COOPERATION AMONG DISTRIBUTED ELECTRONIC AGENTS, the specification of which is attached hereto.

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the examination of this application in accordance with Title 37, CFR § 1.56.

*AJC DXM
Stephens & Coleman, LLP,*

And I hereby appoint the law firm of Hickman ~~& Martin~~ including Paul L. Hickman (Reg. No. 28, 516); L. Keith Stephens (Reg. No. 32,632); Brian R. Coleman (Reg. No. 39,145); Dawn L. Palmer (Reg. No. 41,238); Jerray Wei (Reg. No. 43,247); and Ian L. Cartier (Reg. No. 38,406) as my principal attorneys to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith:

Send Correspondence To: **Brian R. Coleman
HICKMAN STEPHENS & COLEMAN, LLP
P.O. BOX 52037
Palo Alto, California 94303-0746**

Direct Telephone Calls To: **Brian R. Coleman at telephone number (650) 470-7430**

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application or any patent issuing thereon.

655010 86152260

Typewritten Full Name of Sole or First Inventor: Adam J. Cheyer Citizenship: USA
Inventor's signature: *Adam J. Cheyer* Date of Signature: 1/5/99
Residence: (City) Palo Alto (State/Country) CA
Post Office Address: 757 Cereza Drive Palo Alto CA 94306

Typewritten Full Name of Second Inventor: David L. Martin Citizenship: USA
Inventor's signature: *David L. Martin* Date of Signature: 1/5/99
Residence: (City) Santa Clara (State/Country) CA
Post Office Address: 167 CROWN DR. Santa Clara, CA 95051

PATENT APPLICATION FEE DETERMINATION RECORD

Effective November 10, 1998

Application or Docket Number

09/225,198

CLAIMS AS FILED - PART I

(Column 1) (Column 2)

FOR	NUMBER FILED	NUMBER EXTRA
BASIC FEE		
TOTAL CLAIMS	89	minus 20 = * 69
INDEPENDENT CLAIMS	6	minus 3 = * 3
MULTIPLE DEPENDENT CLAIM PRESENT		

* If the difference in column 1 is less than zero, enter "0" in column 2

CLAIMS AS AMENDED - PART II

(Column 1) (Column 2) (Column 3)

AMENDMENT A	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
Total	* 89	Minus ** 20	= 69
Independent	* 6	Minus *** 3	= 3
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM			

(Column 1) (Column 2) (Column 3)

AMENDMENT B	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
Total	*	Minus **	=
Independent	*	Minus ***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM			

(Column 1) (Column 2) (Column 3)

AMENDMENT C	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
Total	*	Minus **	=
Independent	*	Minus ***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM			

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
 ** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20."
 *** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3."
 The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

SMALL ENTITY TYPE OR

OTHER THAN SMALL ENTITY

RATE	FEE	RATE	FEE
	380.00		760.00
X\$ 9=		X\$18=	1242
X39=		X78=	234
+130=		+260=	
TOTAL		TOTAL	2236

SMALL ENTITY OR

OTHER THAN SMALL ENTITY

RATE	ADDITIONAL FEE	RATE	ADDITIONAL FEE
X\$ 9=		X\$18=	1242
X39=		X78=	234.00
+130=		+260=	
TOTAL ADDIT. FEE		TOTAL ADDIT. FEE	

SMALL ENTITY OR

OTHER THAN SMALL ENTITY

RATE	ADDITIONAL FEE	RATE	ADDITIONAL FEE
X\$ 9=		X\$18=	
X39=		X78=	
+130=		+260=	
TOTAL ADDIT. FEE		TOTAL ADDIT. FEE	

SMALL ENTITY OR

OTHER THAN SMALL ENTITY

RATE	ADDITIONAL FEE	RATE	ADDITIONAL FEE
X\$ 9=		X\$18=	
X39=		X78=	
+130=		+260=	
TOTAL ADDIT. FEE		TOTAL ADDIT. FEE	

Best Available Copy

PATENT APPLICATION SERIAL NO. _____

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE
FEE RECORD SHEET

01/19/1999 MIVILLARI 00000027 500384 09225198

01 FC:101	760.00 CH
02 FC:102	234.00 CH
03 FC:103	1242.00 CH

ARTIFACT SHEET

Enter artifact number below. Artifact number is application number + artifact type code (see list below) + sequential letter (A, B, C ...). The first artifact folder for an artifact type receives the letter A, the second B, etc..
Examples: 59123456PA, 59123456PB, 59123456ZA, 59123456ZB

09225198PA

Indicate quantity of a single type of artifact received but not scanned. Create individual artifact folder/box and artifact number for each Artifact Type.

CD(s) containing:

computer program listing

Doc Code: Computer

pages of specification

and/or sequence listing

and/or table

Doc Code: Artifact

content unspecified or combined

Doc Code: Artifact

Artifact Type Code: P

Artifact Type Code: S

Artifact Type Code: U

Stapled Set(s) Color Documents or B/W Photographs

Doc Code: Artifact Artifact Type Code: C

Microfilm(s)

Doc Code: Artifact Artifact Type Code: F

Video tape(s)

Doc Code: Artifact Artifact Type Code: V

Model(s)

Doc Code: Artifact Artifact Type Code: M

Bound Document(s)

Doc Code: Artifact Artifact Type Code: B

Confidential Information Disclosure Statement or Other Documents marked Proprietary, Trade Secrets, Subject to Protective Order, Material Submitted under MPEP 724.02, etc.

Doc Code: Artifact Artifact Type Code X

Other, description: _____

Doc Code: Artifact Artifact Type Code: Z

March 8, 2004

ARTIFACT SHEET

Enter artifact number below. Artifact number is application number + artifact type code (see list below) + sequential letter (A, B, C ...). The first artifact folder for an artifact type receives the letter A, the second B, etc..
Examples: 59123456PA, 59123456PB, 59123456ZA, 59123456ZB

09 225 198 PB

Indicate quantity of a single type of artifact received but not scanned. Create individual artifact folder/box and artifact number for each Artifact Type.

- | | | | |
|-------------------------------------|---------------------------------|-------------------------------------|-----------------------|
| <input checked="" type="checkbox"/> | CD(s) containing: | | |
| | computer program listing | <input checked="" type="checkbox"/> | Artifact Type Code: P |
| | Doc Code: Computer | | |
| | pages of specification | | |
| | and/or sequence listing | <input type="checkbox"/> | |
| | and/or table | | |
| | Doc Code: Artifact | | Artifact Type Code: S |
| | content unspecified or combined | <input type="checkbox"/> | |
| | Doc Code: Artifact | | Artifact Type Code: U |

Stapled Set(s) Color Documents or B/W Photographs
Doc Code: Artifact Artifact Type Code: C

Microfilm(s)
Doc Code: Artifact Artifact Type Code: F

Video tape(s)
Doc Code: Artifact Artifact Type Code: V

Model(s)
Doc Code: Artifact Artifact Type Code: M

Bound Document(s)
Doc Code: Artifact Artifact Type Code: B

Confidential Information Disclosure Statement or Other Documents marked Proprietary, Trade Secrets, Subject to Protective Order, Material Submitted under MPEP 724.02, etc.
Doc Code: Artifact Artifact Type Code X

Other, description: _____
Doc Code: Artifact Artifact Type Code: Z

March 8, 2004

082755

#RS
2

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re the application of:)
 Cheyer et al.)
 Application No.: 09/225,198)
 Filed: January 5, 1999)
 For: SOFTWARE-BASED ARCHITECTURE)
 FOR COMMUNICATION AND COOPERATION)
 AMONG DISTRIBUTED ELECTRONIC)
AGENTS)



Group: 2755
 Examiner: Unassigned
 Atty. Docket No.: SRI1P016
 Date: May 11, 1999

RECEIVED
 MAY 20 1999
 Group 2700

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as First Class Mail in an envelope addressed to: Assistant Commissioner for Patents, Washington, DC 20231 on May 11, 1999

Signed: Jay Vasudevan
 Jay Vasudevan

INFORMATION DISCLOSURE STATEMENT
UNDER 37 CFR §§1.56 AND 1.97(c)

Assistant Commissioner for Patents
 Washington, DC 20231

Dear Sir:

The references listed in the attached PTO Form 1449, copies of which are attached, may be material to examination of the above-identified patent application. Applicants submit these references in compliance with their duty of disclosure pursuant to 37 CFR §§1.56 and 1.97. The Examiner is requested to make these references of official record in this application.

Reference No. R on Page 4 of PTO form 1449 contains documents downloaded from a web site owned by Dejima, Inc. at <http://www.dejima.com> on April 29, 1999 and March 18, 1999. The applicant makes no representation that this web site has not changed between the dates of downloading or that this web site will not change in the future.

This Information Disclosure Statement is not to be construed as a representation that a search has been made, that additional information material to the examination of this application does not exist, or that these references indeed constitute prior art.

Attny Dkt No. SRI1P016

This Information Disclosure Statement is believed to be filed before the mailing date of a first Office Action on the merits. Accordingly, it is believed that no fees are due in connection with the filing of this Information Disclosure Statement. However, if it is determined that any fees are due, the Commissioner is hereby authorized to charge such fees to Deposit Account 50-0384 (Order No. SRI1P016).

Respectfully submitted,
HICKMAN STEPHENS & COLEMAN, LLP



Brian R. Coleman
Reg. No. 39,145

P.O. Box 52037
Palo Alto, CA 94303-0746
Telephone: (650) 470-7430



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/225,198	01/05/1999	ADAM J. CHEYER	SRI1P016	2756

25696 7590 07/17/2002

OPPENHEIMER WOLFF & DONNELLY
P. O. BOX 10356
PALO ALTO, CA 94303

EXAMINER

BULLOCK JR, LEWIS ALEXANDER

ART UNIT	PAPER NUMBER
2151	

2151

DATE MAILED: 07/17/2002

Please find below and/or attached an Office communication concerning this application or proceeding.

J

Office Action Summary	Application No. 09/225,198	Applicant(s) CHEYER ET AL.	
	Examiner Lewis A. Bullock, Jr.	Art Unit 2151	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --
Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133).
- Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

- 1) Responsive to communication(s) filed on _____.
- 2a) This action is **FINAL**. 2b) This action is non-final.
- 3) Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

- 4) Claim(s) 1-89 is/are pending in the application.
 - 4a) Of the above claim(s) _____ is/are withdrawn from consideration.
- 5) Claim(s) _____ is/are allowed.
- 6) Claim(s) 1-89 is/are rejected.
- 7) Claim(s) _____ is/are objected to.
- 8) Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

- 9) The specification is objected to by the Examiner.
- 10) The drawing(s) filed on _____ is/are: a) accepted or b) objected to by the Examiner.
 Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
- 11) The proposed drawing correction filed on _____ is: a) approved b) disapproved by the Examiner.
 If approved, corrected drawings are required in reply to this Office action.
- 12) The oath or declaration is objected to by the Examiner.

Priority under 35 U.S.C. §§ 119 and 120

- 13) Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
 - a) All b) Some * c) None of:
 - 1. Certified copies of the priority documents have been received.
 - 2. Certified copies of the priority documents have been received in Application No. _____.
 - 3. Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).
- * See the attached detailed Office action for a list of the certified copies not received.
- 14) Acknowledgment is made of a claim for domestic priority under 35 U.S.C. § 119(e) (to a provisional application).
 - a) The translation of the foreign language provisional application has been received.
- 15) Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121.

Attachment(s)

- 1) Notice of References Cited (PTO-892)
- 2) Notice of Draftsperson's Patent Drawing Review (PTO-948)
- 3) Information Disclosure Statement(s) (PTO-1449) Paper No(s) 2.
- 4) Interview Summary (PTO-413) Paper No(s). _____
- 5) Notice of Informal Patent Application (PTO-152)
- 6) Other:

DETAILED ACTION

Claim Rejections - 35 USC § 112

1. Claim 2 is rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention. Applicant claims the recursively applying the last step of claim 1, however the Examiner cannot determine which step applicant is referring to.

Applicant is either referring to the dynamically interpreting step and its substep or the dispatching step of the dynamically interpreting step. Clarification is requested.

2. Claim 3 recites the limitation "from the specific agent to the facilitator agent" in lines 5-6. There is insufficient antecedent basis for this limitation in the claim. There is no mention of the facilitator agent anywhere in the parent claims. In review of the specification the examiner finds the facilitator agent performs the steps of claim 1, however, claim 1 does not detail the facilitator agent as performing the steps. The examiner request Applicant to amend claim 1 to detail that the facilitator agent performs the functionality.

3. Claims 84 and 85 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention. Claims 84 and 85 recite the planning and execution components, however neither component has antecedent basis in the parent claim 71. Correction is requested.

4. Claims 87 and 88 recite the limitation "A data wave carrier as recited in claim 85" in line 1. There is insufficient antecedent basis for this limitation in the claim. Claims 87 and 88 should be dependent on claim 86 not claim 85 and are further examined as such.

Claim Rejections - 35 USC § 102

5. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(a) the invention was known or used by others in this country, or patented or described in a printed publication in this or a foreign country, before the invention thereof by the applicant for a patent.

(b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

6. Claims 1, 2, 5-11, 15-28, 48-89 are rejected under 35 U.S.C. 102(a) as being anticipated by "Building Distributed Software Systems with the Open Agent Architecture" by MARTIN.

As to claim 1, MARTIN teaches a computer-implemented method for communication and cooperative task completion among a plurality of distributed agents (application agent / meta agent / user interface agent), comprising the acts of: registering a description of each client agent's functional capabilities (capabilities specifications), using a platform independent inter-agent language (ICL); receiving a request for service as a base goal (goals created by requesters of service) in the inter-agent language, in the form of an arbitrarily complex goal expression; and dynamically interpreting the goal expression (goals) (via facilitator) comprising: generating one or

more sub-goals using the inter-agent language; and dispatching each of the sub-goals to a selected client agent (service providers) for performance, based on a match between the sub-goal being dispatched and the registered functional capabilities of the selected client agent (pg. 7, Mechanisms of Cooperation; pg. 12-14, Requesting Services; Refining Service Requests, and Facilitation).

As to claim 2, MARTIN teaches receiving a new request (subgoal) for service as a base goal from at least one of the selected client agents in response to the sub-goal and recursively applying the dynamically interpreting (pg. 13, Refining Service Requests).

As to claims 5-10, MARTIN teaches providing an agent registry data structure that can comprise of symbolic names, data declarations, trigger declarations, and task and process characteristics (pg. 13-14, Facilitation; pg. 7, "In processing a request...it can use ICL to request services of other agents, set triggers, and read or write shared data on the facilitator...").

As to claim 11, MARTIN teaches establishing communication between distributed agents (pg. 6, The facilitator is a specialized server agent that is responsible for coordinating agent communications and cooperative problem-solving.").

As to claims 15-25, MARTIN teaches the base goal requires setting a trigger having conditional functionality and consequential functionality which can be stored on the facilitator agent and/or the service providing agent (pgs. 16-17, Autonomous Monitoring Using Triggers).

As to claims 26-28, MARTIN teaches the base goal is a compound goal having sub-goals separated by operators, i.e. conjunction operator, disjunction operator, conditional operator, and a parallel operator (pg. 12-13, Compound goals).

As to claim 48, MARTIN teaches an Inter-agent Communication Language (ICL) providing a basis for facilitated cooperative task completion within a distributed computing environment having a facilitator agent (facilitator) and a plurality of electronic agents (service providing agents / service requesting agents), the ICL enabling agents to perform queries of other agents, exchange information with other agents, set triggers within other agents (pgs. 4-7, Overview of OAA System Structure, Mechanisms of Cooperation; pg. 8, "OAA agents employ ICL to perform queries, execute actions, exchange information, set triggers, and manipulate data in the agent community."), an ICL syntax supporting compound goal expressions such that goals within a single request provided according to the ICL syntax may be coupled by a conjunctive operator, a disjunctive operator, a conditional execution operator, and a parallel operator that indicates that goals are to be performed by different agents (pg. 12, Compound goals).

As to claim 49 and 50, MARTIN teaches the ICL is platform and language independent (pg. 8, "OAA's Inter-agent Communication Language (ICL) is the interface, communication, and task coordination language shared by all agents, regardless of what platform they run on or what computer language they are programmed in.").

As to claims 51-54, MARTIN teaches the ICL supports task completion constraints within goal expressions (pg. 9, "A number of important declarations...we consider each of these elements.").

As to claims 55-60, MARTIN teaches each electronic agent defines and publishes a set of capability declarations or solvables that describe services and an interface to the electronic agent (pg. 9, "A number of important declarations...we consider each of these elements.").

As to claims 61 and 62, reference is made to an agent that performs the method of claim 1 above and is therefore met by the rejection of claim 1 above. However, claim 61 further details an agent register and the construction of a goal satisfaction plan. MARTIN teaches an agent register (knowledge base) (pg. 13-14, Facilitation); and the construction of a goal satisfaction plan (pg. 13, "When a facilitator receives a compound goal, its job is to construct a goal satisfaction plan and oversee its satisfaction in the most appropriate, efficient manner that is consistent with the specified advice.").

As to claim 63, refer to claim 5 for rejection.

As to claim 64-69, refer to claims 15-25 for rejection.

As to claim 70, MARTIN teaches the agent registry (knowledge base) is a database accessible to all electronic agents (via the facilitator) (pg. 13-14, Facilitation).

As to claim 71, reference is made to an architecture that encompasses the agent of claim 61 above, and is therefore met by the rejection of claim 61 above. However claim 71, further details the facilitator agent in bi-directional communication with the electronic agents. MARTIN teaches the facilitator agent in bi-directional communication with the electronic agents (fig 1).

As to claim 72, refer to claim 48 for rejection.

As to claims 73 and 74, refer to claims 49 and 50 for rejection.

As to claims 75-78, refer to claims 51-54 for rejection.

As to claims 79-83, refer to claims 54-60 for rejection.

As to claims 84 and 85, MARTIN teaches the facilitating engine is distributed across at least two processes (pg. 6, "Larger systems can be assembled from multiple facilitator/client groups...").

As to claim 86, MARTIN teaches a data wave carrier (system) providing a transport mechanism (layer of conversational protocol / communication functions) for information communication in a distributed computing environment having at least one facilitator agent (facilitator) and at least one client agent (application agent / user interface agent), the carrier comprising a signal representation of an inter-agent language description of a client agent's functional capabilities (registering by the service provider agents) (pg. 6-9).

As to claim 87, MARTIN teaches a signal representation of a request for service in the inter-agent language from a first agent to a second agent (request for service from an service requesting agent to the facilitator) (pg. 12, Requesting Services).

As to claim 88, MARTIN teaches a signal representation of a goal dispatched to an agent for performance from a facilitator agent (pg. 13-14, Facilitation).

As to claim 89, MARTIN teaches a signal representation of a response to the dispatched goal including results and/or a status report from the agent for performance to the facilitator agent (pg. 13-14, Facilitation).

7. Claims 1, 2, 5-11, and 15-25 are rejected under 35 U.S.C. 102(b) as being anticipated by "Development Tools for the Open Agent Architecture" by MARTIN.

As to claim 1, MARTIN teaches a computer-implemented method for communication and cooperative task completion among a plurality of distributed agents (sub-agents / agents), comprising the acts of: registering a description of each client agent's functional capabilities, using a platform independent inter-agent language (pg. 5, Each facilitator records the published capabilities of their subagents..."); receiving a request as a base goal in the inter-agent language (ICL form), in the form of an arbitrarily complex goal expression; and dynamically interpreting the goal expression comprising: generating one or more sub-goals using the inter-agent language; and dispatching each of the sub-goals to a selected client agent for performance ("pg. 5, "...and when requests arrive (expressed in the Inter-agent Communication Language, described below), the facilitator is responsible for breaking them down and for distributing sub-requests to the appropriate agents; "For example, every agent can...and request solutions for a set of goals,...").

As to claim 2, MARTIN teaches receiving a new request for service as a base goal from at least one of the selected client agents in response to the sub-goal and recursively applying the dynamically interpreting (pg. 5, "An agent satisfying a request may require supporting information, and the OAA provides numerous means of requesting data from other agents or from the user.").

As to claims 5-10, MARTIN teaches providing an agent registry data structure that can comprise of symbolic names, data declarations, trigger declarations, and task and process characteristics (pg. 5, "For example, every agent can install local or remote triggers on data..").

As to claim 11, MARTIN teaches establishing communication between distributed agents (pg. 5, "...the facilitator is responsible for breaking them down and for distributing sub-requests to the appropriate agent..").

As to claims 15-25, MARTIN teaches the base goal requires setting a trigger having conditional functionality and consequential functionality which can be stored on the facilitator agent and/or the service providing agent (pg. 5, "For example, every agent can install local or remote triggers on data..").

Claim Rejections - 35 USC § 103

8. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

9. Claims 3, 29-34, and 38-47 are rejected under 35 U.S.C. 103(a) as being unpatentable over "Building Distributed Software Systems with the Open Agent Architecture" by MARTIN.

As to claim 3, MARTIN teaches the act of registering and transmitting the new agent profile from the specific agent to the facilitator agent (pg. 7, "When invoked, a client agent makes a connection to a facilitator...an agent informs its parent facilitator of the services it is capable of providing."). It would be obvious that an agent that is initially created is instantiated in memory before it is registered.

As to claim 29, MARTIN teaches a method to facilitate cooperative task completion within a distributed computing environment supporting an Inter-agent Communication Language among a plurality of electronic agents (fig 1) comprising: providing an agent registry (knowledge base) as disclosed (pg. 13-14, Facilitation); interpreting a service request in order to determine a base goal (compound goal) comprising: determining any task completion advice provided by the base goal, and determining any task completion constraints provided by the base goal (pg. 14, "It may also use strategies or advice specified by the requester.."); constructing a base goal satisfaction plan (pg. 13, "When a facilitator receives a compound goal, its job is to construct a goal satisfaction plan and oversee its satisfaction in the most appropriate, efficient manner that is consistent with the specified advice.") comprising: determining whether the requested service is available, determining sub-goals required in completing the base goal (delegation), selecting suitable service-providing electronic

agents for performing the sub-goals, and ordering a delegation of sub-goal requests to complete the requested service; and implementing the base goal satisfaction plan (pg. 13-14, Facilitation). However, MARTIN does not explicitly mention that the method is operable in a computer program product. It would be obvious to one skilled in the art to generate program code that would entail the method of Martin and thereby obvious that the method can be entailed in a computer program product.

As to claims 30 and 31, MARTIN teaches registering a specific agent (service provider agents) into the agent registry comprising: establishing a bi-directional communications link between the specific agent and a facilitator agent (facilitator) controlling the agent registry; providing a new agent profile to the facilitator agent; and registering the specific agent with the profile thereby making the capabilities available to the facilitator agent (pgs. 9-10, Providing Services; pg. 7, Mechanisms of Cooperation).

As to claim 32, refer to claim 3 for rejection.

As to claim 33, refer to claim 5 for rejection.

As to claim 34, refer to claim 11 for rejection.

As to claims 38-44, refer to claims 15-25 for rejection.

As to claims 45-47, refer to claims 26-28 for rejection.

10. Claims 4, 12-14 and 35-37 is rejected under 35 U.S.C. 103(a) as being unpatentable over "Building Distributed Software Systems with the Open Agent Architecture" by MARTIN1 in view of "Information Brokering in an Agent Architecture" by MARTIN2.

As to claim 4, MARTIN1 substantially discloses the invention above. However, MARTIN1 does not explicitly mention the cited limitation. MARTIN2 teaches deactivating a client agent no longer available to provide services by deleting the registration (pg. 9, Source agents that need to go offline...so that it can unregister the source and retract its schema mapping rules."). Therefore it would be obvious to combine the teachings of MARTIN1 with the teachings of MARTIN2 in order to provide transparent access to a plurality of independent agents (abstract).

As to claims 12-14, MARTIN1 substantially discloses the invention above. However, MARTIN1 does not explicitly mention the cited limitation. MARTIN2 teaches receiving a request for service in a second language (source shema); selecting a registered agent capable of converting the second language into the inter-agent language (broker schema); and forwarding the request for service in a second language to the registered agent for conversion to be performed and the results returned (pg. 12-13, Queries Expressed in a Source Schema). Refer to claim 4 for the motivation to combine.

As to claims 35-37, refer to claims 12-14 for rejection.

11. Claims 3, 29-34, 38-47, 61-71, and 84-89 are rejected under 35 U.S.C. 103(a) as being unpatentable over "Developing Tools for the Open Agent Architecture" by MARTIN.

As to claim 3, MARTIN teaches the act of registering and transmitting the new agent profile from the specific agent to the facilitator agent (pg. 5, "Every agent participating in an OAA-based system defines and publishes a set of capabilities specifications, expressed in the ICL, describing the services that it provides."). It would be obvious that an agent that is initially created is instantiated in memory before it is registered.

As to claim 29, MARTIN teaches a method to facilitate cooperative task completion within a distributed computing environment supporting an Inter-agent Communication Language among a plurality of electronic agents (sub-agents / agents) comprising: providing an agent registry as disclosed (facilitator storage of published sub-agents capabilities); interpreting a service request in order to determine a base goal (via facilitator) constructing a base goal satisfaction plan comprising: determining whether the requested service is available, determining sub-goals required in completing the base goal (determine solutions for a set of goals) selecting suitable service-providing electronic agents for performing the sub-goals, and ordering a

delegation of sub-goal requests to complete the requested service; and implementing the base goal satisfaction plan (pg. 5, "The facilitator is responsible for breaking them down and for distributing sub-requests to the appropriate agents."). However, MARTIN does not explicitly mention that the method is operable in a computer program product or the sending of advice or constraints. It would be obvious that since an agent can request solutions for a goal to be satisfied under a variety of different control strategies (pg. 5) that the control strategies are the advice and/or constraints. It would also be obvious to one skilled in the art to generate program code that would entail the method of Martin and thereby obvious that the method can be entailed in a computer program product.

As to claims 30 and 31, MARTIN teaches registering a specific agent (agent) into the agent registry (list of agents capabilities) comprising: establishing a bi-directional communications link between the specific agent and a facilitator agent controlling the agent registry; providing a new agent profile to the facilitator agent; and registering the specific agent with the profile thereby making the capabilities available to the facilitator agent (pg. 5, "Each facilitator records the published capabilities of their subagents..."; "Every agent participating in an OAA-based system...describing the services that it provides.").

As to claim 32, refer to claim 3 for rejection.

As to claim 33, refer to claim 5 for rejection.

As to claim 34, refer to claim 11 for rejection.

As to claims 38-44, refer to claims 15-25 for rejection.

As to claims 45-47, refer to claims 26-28 for rejection.

As to claim 61 and 62, reference is made to an agent that performs the method of claim 1 above and is therefore met by the rejection of claim 1 above. However, claim 61 further details an agent register and the construction of a goal satisfaction plan. MARTIN teaches every agent participating in an OAA-based system defines and publishes a set of capabilities describing the services that it provides and that the facilitator records these published capabilities (pg. 5). Therefore, there is an agent register of the capabilities of each agent. MARTIN also teaches an agent can request solutions for a set of goals to be satisfied under a variety of different control strategies. It would be obvious that since solutions are determined based on the goals and control strategies that a goal satisfaction plan is created.

As to claim 63, refer to claim 5 for rejection.

As to claim 64-69, refer to claims 15-25 for rejection.

As to claim 70, MARTIN teaches the agent registry (agent library / list of agent capabilities) is a database accessible to all electronic agents (pg. 5, A collection of agents satisfies requests from users, or other agents...one or more facilitators.”; “An agent satisfying a request may require supporting information...requesting data from other agents or from the user.”).

As to claim 71, reference is made to an architecture that encompasses the agent of claim 61 above, and is therefore met by the rejection of claim 61 above. However claim 71, further details the facilitator agent in bi-directional communication with the electronic agents. MARTIN teaches the facilitator can distribute request to the agents and the agents can request information via the facilitator (pg. 5), therefore it would be obvious that the facilitator and agents are in bi-directional communication.

As to claims 84 and 85, MARTIN teaches the facilitating engine is distributed across at least two processes (pg. 5, “Facilitators can, in turn, be connected as clients of other facilitators.”).

As to claim 86, MARTIN teaches system for information communication in a distributed computing environment having at least one facilitator agent (facilitator) and at least one client agent (sub-agent / agents), the carrier comprising a signal representation of an inter-agent language description (ICL registration of capabilities) of

a client agent's functional capabilities (pg. 5, "Each facilitator records the published capabilities of their subagents.."). It would be obvious that the system has a data wave carrier and a transport mechanism for network communication.

As to claim 87, MARTIN teaches a signal representation of a request for service in the inter-agent language from a first agent (client agent sending a query) to a second agent (facilitator) (pg. 5).

As to claim 88, MARTIN teaches a signal representation of a goal dispatched to an agent for performance from a facilitator agent (every agent can request solutions for a set of goals / facilitator is responsible for breaking them down and for distributing sub-requests to the appropriate agent) (pg. 5).

As to claim 89, It is well known in the art to one skilled in the art that an agent can send back a response after processing the request.

12. Claims 4, 12-14, 26-28, 35-37, 48-60, 72-83 are rejected under 35 U.S.C. 103(a) as being unpatentable over "Development Tools for the Open Agent Architecture" by MARTIN1 in view of "Information Brokering in an Agent Architecture" by MARTIN2.

As to claim 4, MARTIN1 substantially discloses the invention above. However, MARTIN1 does not explicitly mention the cited limitation. MARTIN2 teaches deactivating a client agent no longer available to provide services by deleting the

registration (pg. 9, Source agents that need to go offline...so that it can unregister the source and retract its schema mapping rules."). Therefore it would be obvious to combine the teachings of MARTIN1 with the teachings of MARTIN2 in order to provide transparent access to a plurality of independent agents (abstract).

As to claims 12-14, MARTIN1 substantially discloses the invention above. However, MARTIN1 does not explicitly mention the cited limitation. MARTIN2 teaches receiving a request for service in a second language (source schema); selecting a registered agent capable of converting the second language into the inter-agent language (broker schema); and forwarding the request for service in a second language to the registered agent for conversion to be performed and the results returned (pg. 12-13, Queries Expressed in a Source Schema). Refer to claim 4 for the motivation to combine.

As to claims 26-28, MARTIN1 substantially discloses the invention above. However, MARTIN1 does not explicitly mention the cited limitation. MARTIN2 teaches the base goal is a compound goal having sub-goals (pg. 8, "Queries submitted to the Broker are expression...and backtracking in expressing and processing queries."). It would be obvious that since the base goal (query) is broken down and distributed to as sub-requests to the appropriate agents or solutions are requested for a set of goals as disclosed in MARTIN1 that the base goal as a compound goal is broken down based on

operators disclosing where it can be broken down. Refer to claim 4 for the motivation to combine.

As to claims 35-37, refer to claims 12-14 for rejection.

As to claim 48, MARTIN1 teaches an Inter-agent Communication Language (ICL) providing a basis for facilitated cooperative task completion within a distributed computing environment having a facilitator agent (facilitator) and a plurality of electronic agents (sub-agents / agents), the ICL enabling agents to perform queries of other agents, exchange information with other agents, set triggers within other agents (pg. 5, Agents share a common communication language...and may run on any network linked platform."). However, MARTIN1 does not teach the ICL supporting compound goal expressions. MARTIN2 teaches the query is a base goal stored in as a compound goal having sub-goals (pg. 8, "Queries submitted to the Broker are expression...and backtracking in expressing and processing queries."). It would be obvious that since the base goal (query) is broken down and distributed to as sub-requests to the appropriate agents or solutions are requested for a set of goals as disclosed in MARTIN1 that the base goal as a compound goal is broken down based on operators disclosing where it can be broken down. Refer to claim 4 for the motivation to combine.

As to claim 49 and 50, MARTIN1 teaches the ICL is platform and language independent (pg. 5, "The OAA's Inter-agent Communication Language...they are programmed in.").

As to claims 51-54, MARTIN1 teaches the ICL supports task completion constraints (triggers) within goal expressions (pg. 5).

As to claims 54-60, MARTIN1 teaches each electronic agent defines and publishes a set of capability declarations or solvables that describe services and an interface to the electronic agent (pg. 5, "Every agent participating in an OAA-based system defines and publishes...we refer to these capabilities specifications as solvables.").

As to claim 72, refer to claim 48 for rejection.

As to claims 73 and 74, refer to claims 49 and 50 for rejection.

As to claims 75-78, refer to claims 51-54 for rejection.

As to claims 79-83, refer to claims 54-60 for rejection.


Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Lewis A. Bullock, Jr. whose telephone number is (703) 305-0439. The examiner can normally be reached on Monday-Friday, 8:30 am - 5:00 pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Alvin E. Oberley can be reached on (703) 305-9716. The fax phone numbers for the organization where this application or proceeding is assigned are (703) 746-7239 for regular communications and (703) 746-7238 for After Final communications.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the receptionist whose telephone number is (703) 305-0286.

July 11, 2002


ST. JOHN COURTENAY III
PRIMARY EXAMINER

NOTICE OF DRAFTSPERSON'S PATENT DRAWING REVIEW

The drawing(s) filed (insert date) 01/05/99 are:

- A. approved by the Draftsperson under 37 CFR 1.84 or 1.152.
- B. objected to by the Draftsperson under 37 CFR 1.84 or 1.152 for the reasons indicated below. The Examiner will require submission of new, corrected drawings when necessary. Corrected drawing must be submitted according to the instructions on the back of this notice.

<p>1. DRAWINGS. 37 CFR 1.84(a): Acceptable categories of drawings: Black ink. Color. ___ Color drawings are not acceptable until petition is granted. Fig(s) _____ Pencil and non black ink not permitted. Fig(s) _____</p> <p>2. PHOTOGRAPHS. 37 CFR 1.84 (b) ___ 1 full-tone set is required. Fig(s) _____ ___ Photographs not properly mounted (must use brylston board or photographic double-weight paper). Fig(s) _____ ___ Poor quality (half-tone). Fig(s) _____</p> <p>3. TYPE OF PAPER. 37 CFR 1.84(e) ___ Paper not flexible, strong, white, and durable. Fig(s) _____ ___ Erasures, alterations, overwritings, interlineations, folds, copy machine marks not accepted. Fig(s) _____ ___ Mylar, velum paper is not acceptable (too thin). Fig(s) _____</p> <p>4. SIZE OF PAPER. 37 CFR 1.84(f): Acceptable sizes: ___ 21.0 cm by 29.7 cm (DIN size A4) ___ 21.6 cm by 27.9 cm (8 1/2 x 11 inches) ___ All drawing sheets not the same size. Sheet(s) _____ ___ Drawings sheets not an acceptable size. Fig(s) _____</p> <p>5. MARGINS. 37 CFR 1.84(g): Acceptable margins: Top 2.5 cm Left 2.5cm Right 1.5 cm Bottom 1.0 cm SIZE: A4 Size Top 2.5 cm Left 2.5 cm Right 1.5 cm Bottom 1.0 cm SIZE: 8 1/2 x 11 Margins not acceptable. Fig(s) _____ ___ Top (T) ___ Left (L) ___ Right (R) ___ Bottom (B)</p> <p>6. VIEWS. 37 CFR 1.84(h) REMINDER: Specification may require revision to correspond to drawing changes. Partial views. 37 CFR 1.84(h)(2) ___ Brackets needed to show figure as one entity. Fig(s) _____ ___ Views not labeled separately or properly. Fig(s) _____ ___ Enlarged view not labeled separately or properly. Fig(s) _____</p> <p>7. SECTIONAL VIEWS. 37 CFR 1.84 (h)(3) ___ Hatching not indicated for sectional portions of an object. Fig(s) _____ ___ Sectional designation should be noted with Arabic or Roman numbers. Fig(s) _____</p>	<p>8. ARRANGEMENT OF VIEWS. 37 CFR 1.84(i) ___ Words do not appear on a horizontal, left-to-right fashion when page is either upright or turned so that the top becomes the right side, except for graphs. Fig(s) _____</p> <p>9. SCALE. 37 CFR 1.84(k) ___ Scale not large enough to show mechanism without crowding when drawing is reduced in size to two-thirds in reproduction. Fig(s) _____</p> <p>10. CHARACTER OF LINES, NUMBERS, & LETTERS. 37 CFR 1.84(i) ___ Lines, numbers & letters not uniformly thick and well defined, clean, durable, and black (poor line quality). Fig(s) _____</p> <p>11. SHADING. 37 CFR 1.84(m) ___ Solid black areas pale. Fig(s) _____ ___ Solid black shading not permitted. Fig(s) _____ ___ Shade lines, pale, rough and blurred. Fig(s) _____</p> <p>12. NUMBERS, LETTERS, & REFERENCE CHARACTERS. 37 CFR 1.84(p) ___ Numbers and reference characters not plain and legible. Fig(s) _____ ___ Figure legends are poor. Fig(s) _____ ___ Numbers and reference characters not oriented in the same direction as the view. 37 CFR 1.84(p)(1) Fig(s) _____ ___ English alphabet not used. 37 CFR 1.84(p)(2) Figs _____ ___ Numbers, letters and reference characters must be at least .32 cm (1/8 inch) in height. 37 CFR 1.84(p)(3) Fig(s) _____</p> <p>13. LEAD LINES. 37 CFR 1.84(q) ___ Lead lines cross each other. Fig(s) _____ ___ Lead lines missing. Fig(s) _____</p> <p>14. NUMBERING OF SHEETS OF DRAWINGS. 37 CFR 1.84(t) ___ Sheets not numbered consecutively, and in Arabic numerals beginning with number 1. Sheet(s) _____</p> <p>15. NUMBERING OF VIEWS. 37 CFR 1.84(u) ___ Views not numbered consecutively, and in Arabic numerals, beginning with number 1. Fig(s) _____</p> <p>16. CORRECTIONS. 37 CFR 1.84(w) ___ Corrections not made from prior PTO-948 dated _____</p> <p>17. DESIGN DRAWINGS. 37 CFR 1.152 ___ Surface shading shown not appropriate. Fig(s) _____ ___ Solid black shading not used for color contrast. Fig(s) _____</p>
<p>COMMENTS</p>	

Best Available Copy

REVIEWER LAM DATE 02/18/99 TELEPHONE NO. _____

ATTACHMENT TO PAPER NO. 3

Notice of References Cited	Application/Control No. 09/225,198	Applicant(s)/Patent Under Reexamination CHEYER ET AL.	
	Examiner Lewis A. Bullock, Jr.	Art Unit 2151	Page 1 of 1

U.S. PATENT DOCUMENTS

*	Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
A	US-6,338,081	01-2002	Furusawa et al.	709/202
B	US-5,960,404	09-1999	Chaar et al.	705/11
C	US-6,216,173	04-2001	Jones et al.	135/77
D	US-			
E	US-			
F	US-			
G	US-			
H	US-			
I	US-			
J	US-			
K	US-			
L	US-			
M	US-			

FOREIGN PATENT DOCUMENTS

*	Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
N					
O					
P					
Q					
R					
S					
T					

NON-PATENT DOCUMENTS

*	Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
U	Cheyar, Adam. "Mechanisms of Cooperation." October 19, 1998.
V	DeVoe, Deborah. "SRI distributed agents promise flexibility." InfoWorld. December 30 1996.
W	Sycara, Katia et al. "Distributed Intelligent Agents." IEEE. December 1996.
X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.



Form 1449 (Modified) Information Disclosure Statement By Applicant (Use Several Sheets if Necessary)	Atty Docket No. SRI1P016	Serial No.: 09/225,198
	Applicant: Cheyer et al. Filing Date: January 5, 1999	Group: 2755

U.S. Patent Documents

Examiner Initial	No.	Patent No.	Date	Patentee	Class	Sub-class	Filing Date
	A						
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

RECEIVED
MAY 20 1999
Group 2700

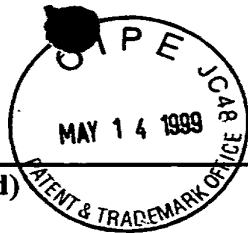
Foreign Patent or Published Foreign Patent Application

Examiner Initial	No.	Document No.	Publication Date	Country or Patent Office	Class	Sub-class	Translation	
							Yes	No
	L							
	M							
	N							
	O							
	P							

Other Documents

Examiner Initial	No.	Author, Title, Date, Place (e.g. Journal) of Publication
<i>Jobs</i>	R	MORAN, Douglas B. and CHEYER, Adam J., "Intelligent Agent-based User Interfaces", Article Intelligence center, SRI International
<i>Jobs</i>	S	MARTIN, David L., CHEYER, Adam J. and MORAN, Douglas B., "Building Distributed Software Systems with the Open Agent Architecture"
<i>Jobs</i>	T	COHEN, Philip R. and CHEYER, Adam, SRI International, WANG, Michelle, Stanford University, BAEG, Soon Cheol, ETRI, "An Open Agent Architecture"
Examiner <i>Amir O. Sulekha Jr</i>		Date Considered <i>7/11/02</i>

Examiner: Initial citation considered. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.



Form 1449 (Modified) Information Disclosure Statement By Applicant (Use Several Sheets if Necessary)	Atty Docket No. SRI1P016	Serial No.: 09/225,198
	Applicant: Cheyer et al. Filing Date: January 5, 1999	Group 2755

U.S. Patent Documents

Examiner Initial	No.	Patent No.	Date	Patentee	Class	Sub-class	Filing Date
	A						
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

RECEIVED
MAY 20 1999
Group 2700

Foreign Patent or Published Foreign Patent Application

Examiner Initial	No.	Document No.	Publication Date	Country or Patent Office	Class	Sub-class	Translation	
							Yes	No
	L							
	M							
	N							
	O							
	P							

Other Documents

Examiner Initial	No.	Author, Title, Date, Place (e.g. Journal) of Publication
<i>JCS</i>	R	JULIA, Luc E. and CHEYER, Adam J., SRI International "Cooperative Agents and Recognition Systems (CARS) for Drivers and Passengers",
<i>JCS</i>	S	MORAN, Douglas B., CHEYER, Adam J., JULIA, Luc E., MARTIN, David L., SRI International, and PARK, Sangkyu, Electronics and Telecommunications Research Institute, "Multimodal User Interfaces in the Open Agent Architecture",
<i>JCS</i>	T	CHEYER, Adam and LULIA, Luc, SRI International "Multimodal Maps: An Agent-based Approach",
Examiner	Date Considered	
<i>Kevin A. Sullivan Jr</i>	<i>7/11/02</i>	

Examiner: Initial citation considered. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.



Form 1449 (Modified) Information Disclosure Statement By Applicant (Use Several Sheets if Necessary)	Atty Docket No. SRI1P016	Serial No.: 09/225,198
	Applicant: Cheyer et al. Filing Date: January 5, 1999	Group 2755

U.S. Patent Documents

Examiner Initial	No.	Patent No.	Date	Patentee	Class	Sub-class	Filing Date
	A						
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

RECEIVED

MAY 20 1999

Group 2700

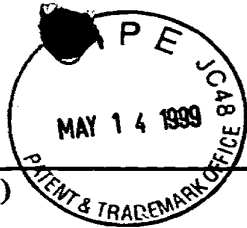
Foreign Patent or Published Foreign Patent Application

Examiner Initial	No.	Document No.	Publication Date	Country or Patent Office	Class	Sub-class	Translation	
							Yes	No
	L							
	M							
	N							
	O							
	P							

Other Documents

Examiner Initial	No.	Author, Title, Date, Place (e.g. Journal) of Publication
<i>fab</i>	R	CUTKOSKY, Mark R., ENGELMORE, Robert S., FIKES, Richard E., GENESERETH, Michael R., GRUBER, Thomas R., Stanford University, MARK, William, Lockheed Palo Alto Research Labs, TENENBAUM, Jay M., WEBER, Jay C., Enterprise Integration Technologies, "An Experiment in Integrating Concurrent Engineering Systems",
<i>fab</i>	S	MARTIN, David L., CHEYER, Adam, SRI International, LEE, Gowang-Lo, ETRI, "Development Tools for the Open Agent Architecture", The Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM96), London, April 1996
<i>fab</i>	T	CHEYER, Adam, MARTIN, David and MORAN, Douglas, "The Open Agent architecture™", SRI International, AI Center
Examiner <i>Lewis A. Bulluck Jr</i>		Date Considered <i>7/11/02</i>

Examiner: Initial citation considered. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.



Form 1449 (Modified) Information Disclosure Statement By Applicant (Use Several Sheets if Necessary)	Atty Docket No. SRI1P016	Serial No.: 09/225,198
	Applicant: Cheyer et al.	Group 2755
	Filing Date: January 5, 1999	

U.S. Patent Documents

Examiner Initial	No.	Patent No.	Date	Patentee	Class	Sub-class	Filing Date
	A						
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

RECEIVED
 MAY 20 1999
 Group 2700

Foreign Patent or Published Foreign Patent Application

Examiner Initial	No.	Document No.	Publication Date	Country or Patent Office	Class	Sub-class	Translation	
							Yes	No
	L							
	M							
	N							
	O							
	P							

Other Documents

Examiner Initial	No.	Author, Title, Date, Place (e.g. Journal) of Publication
<i>jab.</i>	R	Dejima, Inc., http://www.dejima.com/
<i>jab</i>	S	COHEN, Philip R, CHEYER, Adam, WANG, Michelle, Stanford University, BAEG, Soon Cheol ETRI; "An Open Agent Architecture," AAAI Spring Symposium, pp1-8, March 1994
<i>jab</i>	T	MARTIN, David; OOHAMA, Hiroki; MORAN, Douglas; CHEYER, Adam; "Information Brokering in an Agent Architecture," Proceeding of the 2 nd International Conference on Practical Application of Intelligent Agents & Multi-Agent Technology, London, April 1997.
Examiner	Date Considered	
<i>Kevin A. Bulluck Jr</i>	7/11/02	

Examiner: Initial citation considered. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

2755 \$
2151

I hereby certify that this correspondence is being deposited with the U.S. Postal Service with sufficient postage as First Class Mail in an envelope addressed to: Assistant Commissioner for Patents, Washington, D.C., 20231, on:

Date: August 6, 2002

By: Jamie L. Hughes
Jamie L. Hughes



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

PATENT

IN RE APPLICATION OF:

Cheyer

APPLICATION NO.: 09/225,198

FILED: 01/05/1999

FOR: **SOFTWARE-BASED ARCHITECTURE FOR COMMUNICATION AND COOPERATION AMONG DISTRIBUTED ELECTRONIC AGENTS**

EXAMINER: UNKNOWN

ART UNIT: 2755

4

RECEIVED

AUG 15 2002

Technology Center 2100

Information Disclosure Statement After First Office Action but Before Final Action or Notice of Allowance – 37 CFR 1.97(c)

Assistant Commissioner for Patents
Washington, D.C. 20231

Sir:

1. Timing of Submission

The information transmitted herewith is being filed *after* three months of the filing date of this application or after the mailing date of the first Office action on the merits, whichever occurred last, but *before* the mailing date of either a final action under 37 CFR 1.113 or a Notice of Allowance under 37 CFR 1.311, whichever occurs first. The references listed on the enclosed Form PTO/SB/08A may be material to the examination of this application; the Examiner is requested to make them of record in the application.

09/14/002 CHEYER 00000007 502207 09225198

01 FC:126 100.00 CH

2. Cited Information

Copies of the following references are enclosed:

All cited references

3. Effect of Information Disclosure Statement (37 CFR 1.97(h))

This Information Disclosure Statement is not to be construed as a representation that: (i) a search has been made; (ii) additional information material to the examination of this application does not exist; (iii) the information, protocols, results and the like reported by third parties are accurate or enabling; or (iv) the cited information is, or is considered to be, material to patentability. In addition, applicant does not admit that any enclosed item of information constitutes prior art to the subject invention and specifically reserves the right to demonstrate that any such reference is not prior art.

4. Fee Payment (37 CFR 1.97(c)) or Certification (37 CFR 1.97(e))

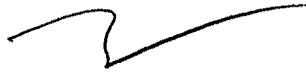
Applicant elects to pay the fee under 37 CFR 1.17(p) \$180.00.

Check enclosed for \$

Please charge the above fee(s) to Deposit Account No. 50-2207
this paper is provided in triplicate.

Date: 6 Aug 2002

Respectfully submitted,
Perkins Coie LLP



Brian R. Coleman
Registration No. 39,145

Correspondence Address:

Customer No. 22918
Perkins Coie LLP
P.O. Box 2168
Menlo Park, California 94026
(650) 838-4300

RECEIVED
AUG 15 2002
Technology Center 2100



2. Cited Information

- Copies of the following references are enclosed:
 - All cited references

Effect of Information Disclosure Statement (37 CFR 1.97(h))

This Information Disclosure Statement is not to be construed as a representation that: (i) a search has been made; (ii) additional information material to the examination of this application does not exist; (iii) the information, protocols, results and the like reported by third parties are accurate or enabling; or (iv) the cited information is, or is considered to be, material to patentability. In addition, applicant does not admit that any enclosed item of information constitutes prior art to the subject invention and specifically reserves the right to demonstrate that any such reference is not prior art.

4. Fee Payment (37 CFR 1.97(c)) or Certification (37 CFR 1.97(e))

- Applicant elects to pay the fee under 37 CFR 1.17(p) \$180.00.
 - Check enclosed for \$
 - Please charge the above fee(s) to Deposit Account No. 50-2207 this paper is provided in triplicate.

Date: 6 Aug 2002

Respectfully submitted,
Perkins Coie LLP

Brian R. Coleman
Registration No. 39,145

Correspondence Address:

Customer No. 22918
Perkins Coie LLP
P.O. Box 2168
Menlo Park, California 94026
(650) 838-4300

This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images,
please do not report the images to the
Image Problems Mailbox.**



INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

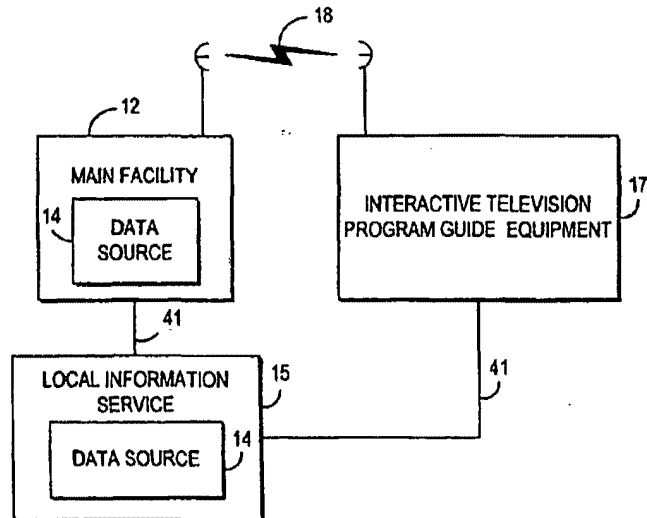
(51) International Patent Classification: H04N 7/16	A1	(11) International Publication Number: WO 00/11869 (43) International Publication Date: 02 March 2000 (02.03.2000)
(21) International Application Number: PCT/US99/19051 (22) International Filing Date: 20 August 1999 (20.08.1999) (30) Priority Data: 60/097,538 21 August 1998 (21.08.1998) US not furnished 30 July 1999 (30.07.1999) US (60) Parent Application or Grant UNITED VIDEO PROPERTIES, INC. [/]; O. ELLIS, Michael, D. [/]; O. LEMMONS, Thomas, R. [/]; O. THOMAS, William, L. [/]; O. TREYZ, G., Victor ; O.	Published	
(54) Title: CLIENT-SERVER ELECTRONIC PROGRAM GUIDE (54) Titre: GUIDE DE PROGRAMMES ELECTRONIQUE CLIENT-SERVEUR		
(57) Abstract <p>A client-server interactive television program guide system is provided. An interactive television program guide client is implemented on user television equipment. The interactive television program guide provides users with an opportunity to define expressions that are processed by the program guide server. The program guide server may provide program guide data, schedules reminders, schedules program recordings, and parentally locks programs based on the expressions. Users' viewing histories may be tracked. The program guide server may analyze the viewing histories and generates viewing recommendations, targets advertising, and collects program ratings information based on the viewing histories.</p> (57) Abrégé <p>L'invention concerne un système de guide de programmes de télévision interactif entre un client et un serveur. Un client de guide de programmes de télévision interactif est mis en application sur l'installation télévisuelle d'un utilisateur. Ce guide de programmes permet aux utilisateurs de définir des expressions traitées par le serveur de guide de programmes. Ce serveur peut produire des données de guide de programmes, des rappels de programmation, des enregistrements de programmes et, de même, verrouille des programmes en fonction des expressions. Il est possible de rechercher l'historique de visualisation des utilisateurs. Le serveur de guide de programmes peut analyser les historiques de visualisation et générer des recommandations de visualisation, des publicités ciblées et recueillir des informations d'évaluation de programmes en fonction de ces historiques de visualisation.</p>		



INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

<p>(51) International Patent Classification ⁷ : H04N 7/16</p>	<p>A1</p>	<p>(11) International Publication Number: WO 00/11869 (43) International Publication Date: 2 March 2000 (02.03.00)</p>
<p>(21) International Application Number: PCT/US99/19051 (22) International Filing Date: 20 August 1999 (20.08.99) (30) Priority Data: 60/097,538 21 August 1998 (21.08.98) US not furnished 13 August 1999 (13.08.99) US (71) Applicant: UNITED VIDEO PROPERTIES, INC. [US/US]; 7140 South Lewis Avenue, Tulsa, OK 74136 (US). (72) Inventors: ELLIS, Michael, D.; 1300 Kingwood Place, Boulder, CO 80304 (US). LEMMONS, Thomas, R.; Route 2, Box 1178, Sand Springs, OK 74063 (US). THOMAS, William, L.; 11611 South 70th East Avenue, Bixby, OK 74008 (US). (74) Agents: TREYZ, G., Victor et al.; Fish & Neave, 1251 Avenue of the Americas, New York, NY 10020 (US).</p>	<p>(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).</p> <p>Published With international search report.</p>	

(54) Title: CLIENT-SERVER ELECTRONIC PROGRAM GUIDE



(57) Abstract

A client-server interactive television program guide system is provided. An interactive television program guide client is implemented on user television equipment. The interactive television program guide provides users with an opportunity to define expressions that are processed by the program guide server. The program guide server may provide program guide data, schedules reminders, schedules program recordings, and parentally locks programs based on the expressions. Users' viewing histories may be tracked. The program guide server may analyze the viewing histories and generates viewing recommendations, targets advertising, and collects program ratings information based on the viewing histories.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LJ	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

Description

5

10

15

20

25

30

35

40

45

50

55

5

10

15

CLIENT-SERVER ELECTRONIC PROGRAM GUIDE

20

25

Background of the Invention

This invention relates to interactive television program guide systems, and more particularly, to interactive television program guide systems based on client-server arrangements.

Cable, satellite, and broadcast television systems provide viewers with a large number of television channels. Users have traditionally consulted printed television program schedules to determine the programs being broadcast at a particular time. More recently, interactive television program guides have been developed that allow television program information to be displayed on a user's television. Interactive television program guides, which are typically implemented on set-top boxes, allow users to navigate through television program listings using a remote control. In a typical program guide, various groups of television program listings are displayed in predefined or user-selected categories. Program listings are typically displayed in a grid or

55

5

- 2 -

10

table. On-line program guides have been proposed that require users to navigate the Internet to access program listings.

15

Client-server based program guides have been proposed in which program listings are stored on a server at a cable system headend. The server provides the program listings to program guide clients implemented on the set-top boxes of a number of users associated with each headend. As users navigate within a program listings grid, the server provides program listings to the client for display. Such systems, may be limited in their functionality due to their limited use of the resources of the server.

20

25

It is therefore an object of the present invention to provide an interactive television program guide system in which server resources are used to provide enhanced program guide features not provided by conventional set-top-box-based or client-server-based program guides.

30

35

20 Summary of the Invention

This and other objects of the present invention are accomplished in accordance with the principles of the present invention by providing a client-server based interactive television program guide system in which a main facility (e.g., a satellite uplink facility or a facility that feeds such an uplink facility) provides data from one or more data sources to a number of television distribution facilities such as cable system headends, broadcast distribution facilities, satellite television distribution facilities, or other suitable distribution facilities. Some of the data sources may be located at

40

45

50

55

5

- 3 -

10

different facilities and have their data provided to the main facility for localization and distribution or may provide their data to the television distribution facilities directly. The data provided to the

15

5 television distribution facilities includes television programming data (e.g., titles, channels, content information, rating information, program identifiers, series identifiers, or any other information associated with television programming), and other program guide

20

10 data for additional services other than television program listings (e.g., weather information, associated Internet web links, computer software, etc.). The main

25

facility (and other sources) may provide the program guide data to the television distribution facilities

15 via a satellite link, a telephone network link, a cable or fiber optic link, a microwave link, an Internet link, a combination of such links, or any other suitable communications link.

30

Each television distribution facility has a
20 program guide server. If desired, program guide servers may also be located at cable system network nodes or other facilities separate from the television distribution facilities or other distribution facilities. Each program guide server stores the
40 25 program guide data provided by the main facility and provides access to the program guide data to program guide clients implemented on the user television equipment of a number of users associated with each television distribution facility. The program guide
45 30 servers may also store user data, such as user preference profiles, parental control settings, record and reminder settings, viewing history, and other suitable data.

50

55

5

- 4 -

10

Providing program guide data with a program guide server and storing user data on the server may provide users with opportunities to perform various functions that may enhance the users' television

15

5 viewing experience. Users may, for example, set user preference profiles or other favorites that are stored by the program guide server and used by the server to customize the program guide viewing experience for the user. The program guide server may filter program

20

10 guide data based on the user preference profiles. Only data that is of interest to the user may then be provided to the guide client, thereby tending to minimize the memory requirements of the user's television equipment and lessen the bandwidth

25

15 requirements of the local distribution network.

30

A client-server based architecture may also provide users with the ability to search and sort through program related information in ways that might not otherwise be possible due to the limited processing

20 and storage capabilities of the users' television

35

equipment. If desired, users may be provided with access to program guide data without requiring them to navigate the Internet. Users may, for example, define sophisticated boolean or natural language expressions

40

25 having one or more criteria for searching through and sorting program guide data, scheduling reminders, automatically recording programs and parentally controlling programs. The criteria may also be derived

45

30 from user profiles or by monitoring usage of the program guide. The criteria may be stored on the program guide server. Users may be provided with an

50

55

5

- 5 -

10

opportunity to access, modify, or delete the expressions.

15

The program guide server may also track the users' viewing histories to provide a user-customized program guide experience. Programs or series of episodes users have watched may be identified and used by the program guide, for example, to inform users when there are showings in the series that the users have not watched. The program guide may, for example, provide viewing recommendations based on a user's viewing history and, if appropriate, on user preference profiles or other criteria stored by the program guide server. The program guide may also target advertisements toward users based on the viewing histories or criteria, and may track the viewing of programs to generate viewership ratings.

20

25

30

Further features of the invention, its nature and various advantages will be more apparent from the accompanying drawings and the following detailed description of the preferred embodiments.

35

Brief Description of the Drawings

40

FIG. 1 is a schematic block diagram of an illustrative system in accordance with the present invention.

25

FIGS. 2a, 2b, and 2c show illustrative arrangements for the interactive program guide equipment of FIG. 1 in accordance with the principles of the present invention.

45

FIG. 3 is an illustrative schematic block diagram of a user television equipment of FIGS. 2a and 2b in accordance with the principles of the present invention.

50

55

5

- 6 -

10

FIG. 4 is a generalized schematic block diagram of portions of the illustrative user television equipment of FIG. 3 in accordance with the principles of the present invention.

15

5 FIG. 5 is an illustrative main menu screen in accordance with the principles of the present invention.

20

FIG. 6 is an illustrative program listings by time screen in accordance with the principles of the present invention.

10

FIG. 7 is an illustrative program listings by channel screen in accordance with the principles of the present invention.

25

FIGS. 8a-8c are illustrative program listings by category screens in accordance with the principles of the present invention.

30

FIG. 9a is an illustrative boolean type criteria screen in accordance with the principles of the present invention.

20

FIG. 9b is an illustrative natural language criteria screen in accordance with the principles of the present invention.

35

FIG. 10 shows an illustrative agents screen in accordance with the principles of the present invention.

40

25

FIG. 11 is an illustrative program listings screen in which program listings found according to the illustrative expressions of FIGS. 9a and 9b are displayed in accordance with the principles of the present invention.

45

30

FIG. 12 shows an illustrative setup screen in accordance with the principles of the present invention.

50

55

5

- 7 -

10

FIGS. 13a-13f show illustrative preference profile screens in accordance with the principles of the present invention.

15

FIG. 14 shows an illustrative profile activation screen in accordance with the principles of the present invention.

20

FIG. 15 shows a table containing an illustrative list of programs that might be available to a user after defining the preference profiles of FIGS. 13a-13f in accordance with the principles of the present invention.

25

FIGS. 16a-16c are illustrative program listings screens that may be displayed according to the preference profiles of FIGS. 13a-13f in accordance with the principles of the present invention.

30

FIGS. 17a and 17b show illustrative criteria screens in accordance with the principles of the present invention.

35

FIGS. 18 and 19 show illustrative program reminder lists generated according to the expressions of FIGS. 17a and 17b in accordance with the principles of the present invention.

40

FIGS. 20a and 20b show an illustrative viewer recommendation overlay, in accordance with the principles of the present invention.

45

FIG. 20c shows an illustrative additional information screen in accordance with the principles of the present invention.

50

FIG. 21 is a flowchart of illustrative steps involved in providing users with an opportunity to define preference profiles and access program guide data according to the preference profiles in accordance with the principles of the present invention.

55

5

- 8 -

10

FIG. 22 is a flowchart of illustrative steps involved in providing users with an opportunity to search program guide data, other information, and videos in accordance with the principles of the present invention.

15

FIG. 23 is a flowchart of illustrative steps involved in processing and using expressions in accordance with the principles of the present invention.

20

FIG. 24 is a flowchart of illustrative steps involved in tracking and using viewing histories in accordance with the principles of the present invention.

25

Detailed Description of the Preferred Embodiments

An illustrative system 10 in accordance with the present invention is shown in FIG. 1. Main facility 12 may provide program guide data from data source 14 to interactive television program guide equipment 17 via communications link 18. There may be multiple program guide data sources in main facility 12 but only one has been shown to avoid over-complicating the drawing. If desired, program guide data sources may be located at facilities separate from main facility 12 such as at local information services 15, and may have their data provided to main facility 12 for localization and distribution. Data sources 14 may be any suitable computer or computer-based system for obtaining data (e.g., manually from an operator, electronically via a computer network or other connection, or via storage media) and placing the data into electronic form for distribution by main facility 12. Link 18 may be a satellite link, a telephone

30

35

40

45

50

55

5

- 9 -

10

network link, a cable or fiber optic link, a microwave link, an Internet link, a combination of such links, or any other suitable communications link. Video signals may also be transmitted over link 18 if desired.

15

5 Local information service 15 may be any suitable facility for obtaining data particular to a localized region and providing the data to main facility 12 or interactive television program guide equipment 17 over communications links 41. Local information service 15 may be, for example, a local weather station that measures weather data, a local newspaper that obtains local high school and college sporting information, or any other suitable provider of information. Local information service 15 may be a local business with a computer for providing main facility 12 with, for example, local ski reports, fishing conditions, menus, etc., or any other suitable provider of information. Link 41 may be a satellite link, a telephone network link, a cable or fiber optic link, a microwave link, an Internet link, a combination of such links, or any other suitable communications link. Additional data sources 14 may be located at other facilities for providing main facility 12 with non-localized data (e.g., non-localized program guide data) over link 41.

20

25

30

35

40

45

50

The program guide data transmitted by main facility 12 to interactive television program guide equipment 17 may include television programming data (e.g., program identifiers, times, channels, titles, descriptions, series identifiers, etc.) and other data for services other than television program listings (e.g., help text, pay-per-view information, weather information, sports information, music channel

55

5

- 10 -

10

information, associated Internet web links, associated software, etc.). There are preferably numerous pieces or installations of interactive television program guide equipment 17, although only one is shown in

15

5 FIG. 1 to avoid over-complicating the drawing.

20

Program guide data may be transmitted by main facility 12 to interactive television program guide equipment 17 using any suitable approach. Data files may, for example, be encapsulated as objects and

25

10 transmitted using a suitable Internet based addressing scheme and protocol stack (e.g., a stack which uses the user datagram protocol (UDP) and Internet protocol (IP)). Systems in which program guide data is transmitted from a main facility to television
15 distribution facilities are described, for example, in Gollahon et al. U.S. patent application Serial No. 09/332,624, filed June 11, 1999 (Attorney Docket No. UV-106), which is hereby incorporated by reference herein in its entirety.

30

20 A client-server based interactive television program guide is implemented on interactive television program guide equipment 17. Three illustrative arrangements for interactive television program guide equipment 17 are shown in FIGS. 2a-2c. FIG. 2a shows
35 an illustrative arrangement for interactive television program guide equipment 17 in which a program guide server obtains program guide data directly from main facility 12. FIG. 2b shows an illustrative arrangement for interactive television program guide equipment 17
40 25 in which a program guide server obtains program guide data from main facility 12 or some other facility (e.g., local information service 15) via the Internet. In either of these approaches, users may be provided

45

50

55

5

- 11 -

10

with opportunities to access program guide data without having to navigate the Internet, if desired. As shown in FIGS. 2a and 2b, interactive program guide television equipment 17 may include television distribution facility 16 and user television equipment 22.

15

20

Television distribution facility 16 may have program guide distribution equipment 21 and program guide server 25. Distribution equipment 21 is equipment suitable for providing program guide data from program guide server 25 to user television equipment 22 over communications path 20. Distribution equipment 21 may include, for example, suitable transmission hardware for distributing program guide data on a television channel sideband, in the vertical blanking interval of a television channel, using an in-band digital signal, using an out-of-band digital signal, over a dedicated computer network or Internet link, or by any other data transmission technique suitable for the type of communications path 20. Analog or digital video signals (e.g., television programs) may also be distributed by distribution equipment 21 to user television equipment 22 over communications paths 20 on multiple analog or digital television channels. Alternatively, videos may be distributed to user television equipment 22 from some other suitable distribution facility, such as a cable system headend, a broadcast distribution facility, a satellite television distribution facility, or any other suitable type of television distribution facility.

25

30

35

40

45

30 other suitable type of television distribution facility.

50

Communications paths 20 may be any communications paths suitable for distributing program

55

5

- 12 -

10

guide data. Communications paths 20 may include, for example, a satellite link, a telephone network link, a cable or fiber optic link, a microwave link, an Internet link, a data-over-cable service interface

15

5 specification (DOCSIS) link, a combination of such links, or any other suitable communications link.

20

Communications paths 20 preferably have sufficient bandwidth to allow television distribution facility 16 or another distribution facility to distribute

10 television programming to user television equipment 22.

25

There are typically multiple pieces of user television equipment 22 and multiple associated communications paths 20, although only one piece of user television equipment 22 and communications path 20 are shown in

15 FIGS. 2a and 2b to avoid over-complicating the

30

drawings. If desired, television programming and program guide data may be provided over separate communications paths.

Program guide server 25 may be based on any
20 suitable combination of server software and hardware.

35

Program guide server 25 may retrieve program guide data or video files from storage device 56 in response to program guide data or video requests generated by an interactive television program guide client implemented

40

25 on user television equipment 22. As shown in FIGS. 2a and 2b, program guide server 25 may include processing circuitry 54 and storage device 56. Processing

45

circuitry 54 may include any suitable processor, such as a microprocessor or group of microprocessors, and other processing circuitry such as caching circuitry, video decoding circuitry, direct memory access (DMA)

30 circuitry, input/output (I/O) circuitry, etc.

50

55

5

- 13 -

10 Storage device 56 may be a memory or other
storage device, such as random access memory (RAM),
flash memory, a hard disk drive, etc., that is suitable
for storing the program guide data transmitted to
5 television distribution facility 16 by main facility
12. User data, such as user preference profiles,
preferences, parental control settings, record and
reminder settings, viewing histories, and other
suitable data may also be stored on storage device 56
20 by program guide server 25. Program guide data and
user data may be stored on storage device 56 in any
suitable format (e.g., a Structured Query Language
(SQL) database). If desired, storage 56 may also store
25 video files for playing back on demand.

15 Processing circuitry 54 may process requests
for program guide data by searching the program guide
data stored on storage device 56 for the requested
data, retrieving the data, and providing the retrieved
30 data to distribution equipment 21 for distribution to
20 user television equipment 22. Processing circuitry 54
may also process storage requests generated by the
program guide client that direct program guide
server 25 to store user data. Alternatively, program
guide server 25 may distribute program guide data to
40 25 and receive user data from user television equipment 22
directly. If communications paths 20 include an
Internet link, DOCSIS link, or other high speed
computer network link (e.g., 10BaseT, 100BaseT,
45 10BaseF, T1, T3, etc.), for example, processing
30 circuitry 54 may include circuitry suitable for
transmitting program guide and user data and receiving
program guide data and storage requests over such a
50 link.

55

5

- 14 -

10

Program guide server 25 may communicate with user television equipment 22 using any suitable communications protocol. For example, program guide server 25 may use a communications protocol stack that

5 includes transmission control protocol (TCP) and Internet protocol (IP) layers, sequenced packet exchange (SPX) and internetwork packet exchange (IPX) layers, Appletalk transaction protocol (ATP) and datagram delivery protocol (DDP) layers, DOCSIS, or any

15 other suitable protocol or combination of protocols. User television equipment 22 may also include suitable hardware for communicating with program guide server 25 over communications paths 20 (e.g., Ethernet cards, modems (digital, analog, or cable), etc.)

15

20

25

15 The program guide client on user television equipment 22 may retrieve program guide data from and store user data on program guide server 25 using any suitable client-server based approach. The program guide may, for example, pass SQL requests as messages

20 to program guide server 25. In another suitable approach, the program guide may invoke remote procedures that reside on program guide server 25 using one or more remote procedure calls. Program guide server 25 may execute SQL statements for such invoked

30 remote procedures. In still another suitable approach, client objects executed by the program guide may communicate with server objects executed by program guide server 25 using, for example, an object request broker (ORB). This may involve using, for example,

35 Microsoft's Distributed Component Object Model (DCOM) approach. As used herein, "record requests" and "storage requests" are intended to encompass any of

40 these types of inter-process or inter-object

30

35

40

45

50

55

5

- 15 -

10

communications, or any other suitable type of inter-process or inter-object communication.

15

FIG. 2b shows an illustrative arrangement for interactive television program guide equipment 17 in which program guide server 25 obtains program guide data via the Internet. The program guide data obtained by program guide server 25 may be provided by main facility 12 or from some other source (e.g., local information service 15) and made available on the Internet. Internet service system 61 may use any suitable combination of hardware and software capable of providing program guide data from the Internet to program guide server 25 using an Internet based approach (e.g., using the HyperText Transfer Protocol (HTTP), File Transfer Protocol (FTP), etc.). FIG. 2b shows Internet service system 61 as being encompassed by television distribution facility 16. If desired, Internet service system 61 may be located at a facility that is separate from television distribution facility 16. Internet service system 61 may, for example, be located at main facility 12 or at some other Internet node suitable for providing program guide data from the Internet to program guide server 25. The functionality of Internet service system 61 and program guide server 25 may be integrated into one system if desired.

20

25

30

35

40

45

Interactive television program guide equipment 17 may include, for example, television distribution facility 16 having program guide server 25 and Internet service system 61. A program guide client application may run on personal computer 23. The client may access

50

55

5

- 16 -

10

program guide server 25 via Internet service system 61 and communications path 20. Personal computer 23 may include processing circuitry 27, memory 29, storage device 31, communications device 35, and monitor 39.

15

5 Processing circuitry 27 may include any suitable processor, such as a microprocessor or group of microprocessors, and other processing circuitry such as caching circuitry, direct memory access (DMA) circuitry, input/output (I/O) circuitry, etc.

20

10 Processing circuitry 27 may also include suitable circuitry for displaying television programming. Personal computer 23 may include, for example, a PC/TV card. Memory 29 may be any suitable memory, such as random access memory (RAM) or read only memory (ROM),
25 that is suitable for storing the computer instructions and data. Storage device 31 may be any suitable storage device, such as a hard disk, floppy disk drive, flash RAM card, recordable CD-ROM drive, or any other suitable storage device. Communications device 35 may
30 be any suitable communications device, such as a conventional analog modem or cable modem.

35

40

An illustrative arrangement for user television equipment 22 of FIGS. 2a and 2b is shown in FIG. 3. User television equipment 22 of FIG. 3
25 receives analog video or a digital video stream and data, program guide data, or any suitable combination thereof, from television distribution facility 16 (FIG. 1) at input 26. During normal television viewing, a
45 user tunes set-top box 28 to a desired television
30 channel. The signal for that television channel is then provided at video output 30. The signal supplied at output 30 is typically either a radio-frequency (RF) signal on a predefined channel (e.g., channel 3 or 4),

50

55

5

- 17 -

10

15

or a analog demodulated video signal, but may also be a digital signal provided to television 36 on an appropriate digital bus (e.g., a bus using the Institute of Electrical and Electronics Engineers (IEEE) 1394 standard, (not shown)). The video signal at output 30 is received by optional secondary storage device 32.

20

25

30

35

The interactive television program guide client may run on set-top box 28, on television 36 (if television 36 has suitable processing circuitry and memory), on a suitable analog or digital receiver connected to television 36, or on digital storage device 31 if digital storage device 31 has suitable processing circuitry and memory. The interactive television program guide client may also run cooperatively on a suitable combination of these devices. Interactive television application systems in which a cooperative interactive television program guide application runs on multiple devices are described, for example, in Ellis U.S. patent application Serial No. 09/186,598, filed November 5, 1998, which is hereby incorporated by reference herein in its entirety.

40

45

50

Secondary storage device 32 can be any suitable type of analog or digital program storage device or player (e.g., a videocassette recorder, a digital versatile disc (DVD) player, etc.). Program recording and other features may be controlled by set-top box 28 using control path 34. If secondary storage device 32 is a videocassette recorder, for example, a typical control path 34 involves the use of an infrared transmitter coupled to the infrared receiver in the videocassette recorder that normally

55

5

- 18 -

10

accepts commands from a remote control such as remote control 40. Remote control 40 may be used to control set-top box 28, secondary storage device 32, and television 36.

15

5 If desired, a user may record programs, program guide data, or a combination thereof in digital form on optional digital storage device 31. Digital storage device 31 may be a writeable optical storage device (such as a DVD player capable of handling
20 10 recordable DVD discs), a magnetic storage device (such as a disk drive or digital tape), or any other digital storage device. Interactive television program guide systems that have digital storage devices are
25 described, for example, in Hassell et al. U.S. patent application Serial No. 09/157,256, filed September 17,
15 1998, which is hereby incorporated by reference herein in its entirety.

30

35

40

45

50

Digital storage device 31 can be contained in set-top box 28 or it can be an external device
20 connected to set-top box 28 via an output port and appropriate interface. Digital storage device 31 may,
35 for example, be contained in local media server 29. If necessary, processing circuitry in set-top box 28 formats the received video, audio and data signals into
40 25 a digital file format. Preferably, the file format is an open file format such as the Moving Picture Experts Group (MPEG) MPEG-2 standard or the Moving Joint
Photographic Experts Group (MJPEG) standard. The
45 resulting data is streamed to digital storage device 31
30 via an appropriate bus (e.g., a bus using the Institute Electrical and Electronics Engineers (IEEE) 1394
standard), and is stored on digital storage device 31.
50 In another suitable approach, an MPEG-2 data stream or

55

5

- 19 -

10

series of files may be received from distribution equipment 21 and stored.

15

Television 36 receives video signals from secondary storage device 32 via communications path 38.

20

5 The video signals on communications path 38 may either be generated by secondary storage device 32 when playing back a prerecorded storage medium (e.g., a videocassette or a recordable digital video disc), by digital storage device 31 when playing back a pre-
10 recorded digital medium, may be passed through from set-top box 28, may be provided directly to television 36 from set-top box 28 if secondary storage device 32 is not included in user television equipment 22, or may
25 be received directly by television 36. During normal television viewing, the video signals provided to
15 television 36 correspond to the desired channel to which a user has tuned with set-top box 28. Video signals may also be provided to television 36 by set-
30 top box 28 when set-top box 28 is used to play back information stored on digital storage device 31.

25

30

35

Set-top box 28 may have communications device 37 for communicating with program guide server
25 25 over communications path 20. Communications device 37 may be a modem (e.g., any suitable analog or digital standard, cellular, or cable modem), network interface
40 card (e.g., an Ethernet card, Token ring card, etc.), a combination of such devices, or any other suitable communications device. Television 36 may also have
45 such a suitable communications device if desired.

45

50

30 Set-top box 28 may have memory 44. Memory 44 may be any memory or other storage device, such as a random access memory (RAM), read only memory (ROM), flash memory, a hard disk drive, a combination of such

55

5

- 20 -

10

devices, etc., that is suitable for storing program guide client instructions and program guide data for use by the program guide client.

15

A more generalized embodiment of user television equipment 22 of FIG. 3 is shown in FIG. 4. As shown in FIG. 4, program guide data from television distribution facility 16 (FIG. 1) and programming are received by control circuitry 42 of user television equipment 22. The functions of control circuitry 42 may be provided using the set-top box arrangement of FIGS. 2a and 2b. Alternatively, these functions may be integrated into an advanced television receiver, personal computer television (PC/TV) such as shown in FIG. 2c, or any other suitable arrangement. If desired, a combination of such arrangements may be used.

20

25

User television equipment 22 may also have secondary storage device 47 and digital storage device 49 for recording programming. Secondary storage device 47 can be any suitable type of analog or digital program storage device (e.g., a videocassette recorder, a digital versatile disc (DVD), etc.). Program recording and other features may be controlled by control circuitry 42. Digital storage device 49 may be, for example, a writeable optical storage device (such as a DVD player capable of handling recordable DVD discs), a magnetic storage device (such as a disk drive or digital tape), or any other digital storage device.

30

35

40

45

User television equipment 22 may also have memory 63. Memory 63 may be any memory or other storage device, such as a random access memory (RAM), read only memory (ROM), flash memory, a hard disk

50

55

5

- 21 -

10

drive, a combination of such devices, etc., that is suitable for storing program guide client instructions and program guide data for use by control circuitry 42.

15

User television equipment 22 of FIG. 4 may also have communications device 51 for supporting communications between the program guide client and program guide server 25 and via communications path 20. Communications device 51 may be a modem (e.g., any suitable analog or digital standard, cellular, or cable modem), network interface card (e.g., an Ethernet card, Token ring card, etc.), a combination of such devices, or any other suitable communications device.

20

25

A user controls the operation of user television equipment 22 with user interface 46. User interface 46 may be a pointing device, wireless remote control, keyboard, touch-pad, voice recognition system, or any other suitable user input device. To watch television, a user instructs control circuitry 42 to display a desired television channel on display device 45. To access the functions of the program guide, a user instructs the program guide implemented on interactive television program guide equipment 17 to generate a main menu or other desired program guide display screen for display on display device 45. If desired, the program guide client running on user television equipment 22 may provide users with access to program guide features without requiring them to navigate the Internet.

30

35

40

45

The program guide may provide users with an opportunity to access program guide features through a main menu. A main menu screen, such as illustrative main menu screen 100 of FIG. 5, may include menu 102 of selectable program guide features 106. If desired,

50

55

5

- 22 -

10

program guide features 106 may be organized according to feature type. In menu 102, for example, program guide features 106 have been organized into three columns. The column labeled "TV GUIDE" is for listings

15

5 related features, the column labeled "MSO SHOWCASE" is for multiple system operator (MSO) related features, and the column labeled "VIEWER SERVICES" is for viewer related features. The interactive television program guide may generate a display screen for a particular

10 program guide feature when a user selects that feature from menu 102.

20

25

Main menu screen 100 may include one or more selectable advertisements 108. Selectable

30

15 advertisements 108 may, for example, include text and graphics advertising pay-per-view programs or other programs or products. When a user selects a selectable advertisement 108, the program guide may display information (e.g., pay-per-view information) or take other actions related to the content of the

20 advertisement. Pure text advertisements may be presented, if desired, as illustrated by selectable advertisement banner 110.

35

40

Main menu screen 100 may also include other screen elements. The brand of the program guide

25 product may be indicated, for example, using a product brand logo graphic such as product brand logo graphic 112. The identity of the television service provider may be presented, for example, using a service provider logo graphic such as service provider logo

45 graphic 114. The current time may be displayed in clock display region 116. In addition, a suitable indicator such as indicator graphic 118 may be used to

50 indicate to a user that mail from a cable operator is

45

50

55

5

- 23 -

10

waiting for a user if the program guide supports messaging functions.

15

The interactive television program guide may provide a user with an opportunity to view television program listings. A user may indicate a desire to view program listings by, for example, positioning highlight region 120 over a desired program guide feature 106. Alternatively, the program guide may present program listings when a user presses a suitable key (e.g., a "guide" key) on remote control 40. When a user indicates a desire to view television program listings, the program guide client requests listings from program guide server 25 and generates an appropriate program listings screen for display on display device 45 (FIG. 4). Program listings screens may be overlaid on a program being viewed by a user or overlaid on a portion of the program in a "browse" mode. Program listings screens are described, for example, in Knudson et al. U.S. patent application Serial No. 09/357,941, filed July 16, 1999 (Attorney Docket No. UV-114), which is hereby incorporated by reference herein in its entirety.

25

30

35

40

45

A program listings screen may contain one or more groups or lists of program listings organized according to one or more organization criteria (e.g., by time, by channel, by program category, etc.). The program guide may, for example, provide a user with an opportunity to view listings by time, by channel, according to a number of categories (e.g., movies, sports, children, etc.), or may allow a user to search for a listing by title. Program listings may be displayed using any suitable list, table, grid, or other suitable display arrangement. If desired,

50

55

5

- 24 -

10

program listings screens may include selectable advertisements, product brand logo graphics, service provider brand graphics, clocks, or any other suitable indicator or graphic.

15

5 A user may indicate a desire to view program listings by time, channel, or category by, for example, selecting a selectable feature 106 from menu 102. In response, the program guide client may issue one or more requests to program guide server 25 for listings
20 10 in the selected category if such listings are not already cached in memory 63 (FIG. 4). Program guide server 25 may retrieve program guide data stored on storage device 56, on another server, or from Internet service system 61, and provide the data to the program
25 15 guide client via program guide distribution equipment 21.

30

The program guide client may display program listings in a suitable program listings screen on user television equipment 22. FIG. 6 illustrates the
20 display of program listings by time. Program listings screen 130 of FIG. 6 may include highlight region 151, which highlights the current program listing 150. A user may position highlight region 151 by entering appropriate commands with user interface 46. For
35 40 example, if user interface 46 has a keypad, a user can position highlight region 151 using "up" and "down" arrow keys on remote control 40. A user may select a listing by, for example, pressing on the "OK" or "info" key on remote control 40. Alternatively, a touch
45 30 sensitive screen, trackball, voice recognition device, or other suitable device may be used to move highlight region 151 or to select program listings without the use of highlight region 151. In still another
50

55

5

- 25 -

10 approach, a user may speak a television program listing
into a voice request recognition system. These methods
of selecting program listings are merely illustrative.
Any other suitable approach for selecting program
5 listings may be used if desired.

15 A user may view additional listings for the
time slot indicated in timebar 111 by, for example,
pressing an "up" or "down" arrow, or a "page up" or
"page down" key on remote control 40. The user may
20 also see listings for the next 24 hour period, or the
last 24 hour period, by pressing a "day forward" or
"day backward" key on remote control 40, respectively.
If there are no listings starting exactly 24 hours in
25 the indicated direction, the program guide may pick
15 programs starting at either closer or further than 24
hours away. If desired, the program guide may require
a user to scroll through advertisement banner 110. A
30 user may view program listings for other time slots by,
for example, pressing "right" and "left" arrows on
20 remote control 40.

35 FIG. 7 illustrates the display of program
listings by channel. A user may scroll up and down to
view program listings for additional time slots, and
may scroll left and right to view program listings for
40 25 other channels. If desired, the day for which program
listings are displayed may be included in display
area 147 with the channel number as shown.

45 The program guide may provide users with an
opportunity to view program listings sorted by
30 category. A user may, for example, press a special
category key on remote control 40 (e.g., "movies",
"sports", "children", etc.), select a selectable
50 category feature from main menu screen 100 (FIG. 5), or

55

5

- 26 -

may indicate a desire to view program listings by
category using any other suitable approach. FIG. 8a is
an illustrative program listings screen in which
program listings for movies are displayed. FIG. 8b is
an illustrative program listings screen in which
program listings for sports-related programming are
displayed. FIG. 8c is an illustrative program listings
screen in which program listings for children's
programs are displayed.

In program listings display screens such as
those shown in FIGS. 7a and 8a-8c for example, program
listings within lists 129 may be divided into
predefined time slots, such as into 30 minute time
slots. Between each time slot, separator 128 may be
displayed to indicate to a user that a user has
scrolled or paged program listings from one time slot
to the next. In FIG. 7 for example, a user is
scrolling from program listings in the 11:30 PM to the
12:00 AM time slot. This is indicated by the display of
the name of the next week day. In FIGS. 8a-8c, for
example, a user is scrolling from program listings in
the 12:30 PM time slot to program listings in the 1:00
PM time slot. If desired, separators 128 may be
displayed only for those timeslots for which there are
listings. When the user scrolls within listings,
highlight region 151 may skip separator 128. FIGS. 6,
7, and 8a-8c also illustrate how the program guide may
display an advertisement banner so that a user is
required to scroll past the banner to access additional
program listings.

The program listings screens of FIGS. 6, 7,
8a, 8b, and 8c have also been shown as including
various other screen elements. Program listings

55

5

- 27 -

10

display screens may include, for example, selectable advertisements, advertisement banners, brand logos, service provider logos, clocks, message indicators, or any other suitable screen element. The program guide

15

5 may provide users with access to selectable advertisements in response to, for example, a user pressing left arrows to move highlight region 151 to highlight a selectable advertisement. In the illustrative program listings screens of FIGS. 6, 8a, 8b, and 8c, the program guide may also adjust the time displayed in timebar 123 as the user scrolls or pages through program listings to reflect the time of the program listing at the top of the list.

20

25

30

35

40

45

50

The program guide client may provide a user with an opportunity to define sophisticated boolean or natural language expressions of one or more criteria. Such criteria may include, for example, attribute type and attribute information that is provided by program guide server 25. The user defined expressions may be stored by program guide server 25 for searching through and sorting program guide data, scheduling reminders, automatically recording programs, and parentally controlling programs. Criteria may also be derived by the program guide server or program guide client from user profiles or by monitoring usage of the program guide or advertising. Program guide server 25 may also use expressions to obtain other types of information or programs. Program guide server 25 may obtain, for example, video-on-demand programs, web site links, games, chat group links, merchandise information, or any other suitable information or programming from data sources 14 located at main facility 12 or other facilities. The program guide client may provide users

55

5

- 28 -

with an opportunity to access, modify, or delete the expressions if desired.

10

A user may indicate a desire to search program guide data by, for example, selecting

5 selectable Search feature 106 of main menu 102 (FIG. 5). In response, the program guide client may display a criteria screen, such as illustrative criteria screen 141 and 149 of FIGS. 9a and 9b. The program guide client may display criteria screen 141 of FIG. 9a to 10 provide a user with an opportunity to define a boolean expression. The user may construct a boolean expression by selecting criteria such as attribute types, attributes, logical operators, and sorting criteria. User selectable criteria may also include 15 what program guide server 25 searches for such as, for example, program listings, program information, web sites, video-on-demand videos, software, or any other suitable program guide data, other information, or videos.

15

20

25

30

20 Users may define expressions by, for example, arrowing up or down between criteria, arrowing left or 35 right to choose an attribute, attribute type or logical operator, and pressing a suitable key to indicate that the user is finished (e.g., an "OK" key). In the 40 example of FIG. 9a, the user has constructed a boolean expression for all action programs that have the actor Bruce Willis, that start between 7:00P and 11:00P, and that end between 9:00P and 1:30A on the current day. 45 FIG. 9a has not been shown as including criteria for selecting what program guide server 25 searches for to 30 avoid over-complicating the drawing.

40

45

The program guide client may display criteria screen 149 of FIG. 9b to provide a user with an

50

55

5

- 29 -

10

opportunity to construct a natural language expression. The user may enter a natural language phrase, such as "List in alphabetical order all action programs starring Bruce Willis and that start today between 7:00P and 11:00P and that end between 9:00P and 1:30A" using user interface 46 (FIG. 4).

15

The program guide client may submit the user defined boolean expression or the natural language expression to program guide server 25 for processing.

20

10 Program guide server 25 may process the expression, and provide the resulting program guide data (e.g., program listings, program information, software, Internet links, etc.) or video programs to the program guide client for display. FIG. 11 shows an illustrative program listings screen that may be displayed by the program guide client in response to the expressions defined in FIGS. 9a and 9b.

25

30

Users may also indicate a desire to have program guide server 25 automatically process expressions by, for example, saving defined expressions as agents. A user may indicate a desire to save an expression as an agent by, for example, selecting Save As Agent selectable feature 147 of FIGS. 9a and 9b after defining a boolean or natural language expression. The program guide client may automatically highlight Save As Agent selectable feature 147 when a user indicates that the user is finished defining an expression (e.g., by pressing an "OK" key). If desired the program guide client may provide the user with an opportunity to name the agent.

35

40

45

Users may access saved expressions or agents by, for example, selecting selectable Agent feature 106 of main menu 102. In response, the program guide

50

55

5

- 30 -

10

client may display a list of saved expressions or agents. An illustrative agents screen 1101 is shown in FIG. 10. A user may indicate a desire to view program listings by, for example, positioning highlight region 151 over the desired expression and pressing an "OK" key on remote control 40. In response to a user indicating a desire to access an expression, the program guide client may submit the user defined expression to program guide server 25 for processing. Program guide server may process the expression, and provide program listings to the program guide client for display in a program listings screen. For example, if a user saved the boolean expression of FIG. 9a, named it "Bruce Willis", and then indicated a desire to access listings for the expression the program guide client may display the listings screen of FIG. 10.

15

20

25

30

In still another approach, the program guide client may provide the expression to program guide server 25 in response to the user saving the expression as an agent. Program guide server 25 may store the expression and monitor the data stored on storage device 56 for program guide listings, program information, other information, software, videos, etc., that match the expression. Program guide server 25 may also query other sources for program guide data and videos that match the expression via, for example, the Internet. Program guide server 25 may obtain the program guide data, other information or videos from storage device 56 or other sources and provide them to the program guide client when the user indicates a desire to access the agent. Alternatively, program guide server 25 may provide the program guide data, other information, or videos to the program guide

35

40

45

50

55

5

- 31 -

10

15

client automatically when the user accesses a feature of the program guide that would display such information. In still another suitable approach, program guide server 25 may provide, for example, program identifiers and air times to the program guide client for use in generating program reminders that indicate found programs.

20

25

30

35

The program guide may also provide users with an opportunity to define user preferences that allow users to customize their program guide experience. Systems in which interactive television program guides provide users with opportunities to define user preference profiles are described, for example, in Ellis et al. U.S. patent application Serial No. 09/034,934, filed March 4, 1998 (Attorney Docket No. UV-43), which is hereby incorporated by reference herein in its entirety. Users may indicate a desire to set up user preference profiles, for example, by selecting a selectable Setup feature 106 from main menu 102 of FIG. 5. When a user selects a selectable Setup feature 106 from main menu 102, the program guide client may display a setup screen, such as illustrative setup screen 411 of FIG. 12.

40

45

50

Setup screen 411 may provide a user with an opportunity to set up various guide features, set parental control features, set features of set-top box 28 (FIG. 3), set audio features, set the screen position, set user preference profiles, or to set up any other feature or suitable combination of features. The user may indicate a desire to set up a user preference profile by, for example, selecting User Profile feature 417. When the user indicates a desire to set up a user preference profile, the program guide

55

5

- 32 -

client may display a user preference profile setup
screen, such as the preference profile setup screens
shown in FIGS. 13a-13f. This method of defining user
profiles is only illustrative, as any suitable method
5 may be used.

15

In practice, there may be multiple users
associated with each user television equipment 22. The
program guide may provide users with the ability to set
up multiple user preference profiles. Users may switch
20 between user preference profiles by, for example,
selecting preference profile selector 109 and arrowing
right or left to select the desired user preference
profile. In FIGS. 13a-13f, for example, the user has
selected Preference profile #1, which may correspond to
25 a particular user.

30

User preference profiles may include criteria
such as preference attributes 104 and preference levels
106. Preference attributes 104 may be organized by
type. Attribute types and attributes may be programmed
20 into the program guide client, or may be retrieved by
the program guide client from program guide server 25.
In the former approach, the available attribute types
and attributes may remain static until the program
guide client is updated. In the latter approach, the
40 25 available attribute types and attributes may be
dynamic. Suitable attribute types and attributes may
be provided at any time by main facility 12 or
television distribution facility 16. Each time a user
indicates a desire to set up a user preference profile,
45 30 the program guide client may query program guide server
25 for the available attribute types and attributes.
When a user indicates a desire to set up a user
50 preference profile in either approach, the program

55

5

- 33 -

10

guide client may query program guide server 25 for the user preference profiles associated with that program guide client.

15

FIGS. 13a-13f show six illustrative views of preference profile setup screens in which the user has selected attribute types by, for example, selecting attribute selector 111 and arrowing right or left until a desired preference attribute type is displayed. For example, FIGS. 13a-13f illustrate how the program guide may provide a user with an opportunity to set preference levels for series, genres, channels, actors and actresses, ratings, and other types of preference attributes, respectively. The user may select preference attributes by, for example, arrowing down after selecting an attribute type. The user may then arrow right or left until a desired attribute is displayed. After the desired preference attribute is displayed, the user may, for example, arrow down to set a preference level for the attribute. The user may then, for example, arrow right or left to select a suitable preference level.

20

25

30

35

40

45

Preference levels that may be used to indicate the user's interest or disinterest in a given preference attribute include strong like, weak like, strong dislike, weak dislike, mandatory (appropriate, e.g., for closed-captioning for a deaf person), illegal (appropriate, e.g., for R-rated programs for a child) and don't care (neutral). After the user indicates that he or she is finished defining a profile (e.g., by pressing an "OK" key or remote control 40), the program guide client may provide the preference profile data to program guide server 25 for use in providing program guide data. The user may arrow down again to select

50

55

5

- 34 -

10

additional criteria, or arrow up to edit criteria that has already been selected. The user may delete an attribute by, for example, setting its preference level to "don't care."

15

5 The user may activate or deactivate one or more defined preference profiles by, for example, selecting selectable Profile feature 106 from main menu 102 of FIG. 5. The program guide client may respond by, for example, querying program guide server 25 for 10 any defined preference profiles, providing the user with a list of preference profiles, and providing the user with an opportunity to activate or deactivate one or more preference profiles as shown in FIG. 14. A 25 user may deactivate a preference profile by, for 15 example, setting the profile to non-active. A user may set a preference profile as active to varying degrees. For example, a user may set a profile as active by 30 setting the profile to "wide", "moderate", or "narrow" scope.

30

25

35

20 The program guide client may also indicate to program guide server 25 which profiles are activated or deactivated. The program guide server may use, for example, the attributes of one or more user preference profiles as additional criteria when retrieving data in 40 response to data requests from the program guide client. If multiple preference profiles are used simultaneously, program guide server 25 may reconcile any conflicts using any suitable approach. Interactive 45 television program guide systems that resolve conflicts among multiple active user preference profiles are 30 described, for example, in above-mentioned Ellis et al. U.S. patent application Serial No. 09/034,934, filed 50 March 4, 1998.

50

55

5

- 35 -

10

15

20

FIG. 15 is a table containing an illustrative list of programs that might be available to a user. The results that appear under the columns labeled "narrow scope", "moderate scope", and "wide scope", show which programs satisfy the preference attributes and preference levels of, for example, Profile #1 as illustratively defined in FIGS. 13a-13f. In practice, a listings screen generated based on a profile that is set to widest scope may typically include a larger number of program listings depending on the mandatory attributes set by the user.

25

30

35

40

45

When the user activates Profile #1 and sets it to the widest scope, program guide server 25 may provide program guide data for programs that have all mandatory attributes and no illegal attributes. For example, Seinfeld, The Shining, ER, Terminator, and My Stepmother is an Alien are included in the widest preference scope because they have the only mandatory attribute that is specified in Profile #1 -- closed-captioning (as set in FIG. 13f). In addition, they have no preference attributes with a preference level of illegal (R rating, TV-MA rating, or NC-17 rating (as set in FIG. 13e). The Night at the Opera is not included because it does not have a mandatory attribute (closed-captioning). Dante's Peak is not included because it has a illegal rating (R). An illustrative program listings screen that may be displayed by the program guide client with such limited data is shown in FIG. 16a (ER has not been listed because, presumably, it would be in a different time block).

50

55

When the user activates Profile #1 and sets it to the moderate scope, program guide server 25 may provide program guide data for programs that have no

5

- 36 -

10

15

20

25

30

preference attributes with an associated preference level of disliked, that have all mandatory attributes, and that have no illegal attributes. The Shining is not included because horrors have a preference level of "weak dislike" (as set in FIG. 13b). Dante's Peak is not included because it has an R-rating, which has an attribute level of illegal (as set in FIG. 13e). Night at the Opera is not included because it is not closed-captioned, which is a mandatory attribute (as set in FIG. 13f). The Terminator, for example is not within the moderate scope of Profile #1 because the preference attribute of horror in Profile #1 has an associated preference level of "weak dislike" and the preference attribute of Schwarzenegger (an actor in the program Terminator) has an associated preference level of "strong dislike" (as set in FIGS. 13b and 13d, respectively). Seinfeld and ER are included because they do not have any disliked attributes.

35

40

45

When faced with two different preference levels associated with the same program, the program guide uses the stronger of the two. My Stepmother is an Alien is included, for example, because it has a "strong like" preference attribute that outweighs the "weak dislike". An illustrative program listings screen that may be displayed by the program guide client with such limited program guide data is shown in FIG. 16b. In practice, a listings screen generated based on a profile that is set to moderate scope may typically include a larger number of program listings depending on the mandatory attributes set by the user.

50

When the user activates Profile #1 and sets it to the narrow preference scope, program guide server may provide program guide data for all liked

55

5

- 37 -

10

programs that are not more disliked and that have all mandatory attributes and no illegal attributes. The Shining is not included because it has a weakly disliked attribute, horror. Terminator is not included

15

5 because it has a strongly disliked attribute, Arnold Schwarzenegger. My Stepmother is an Alien is included because the strongly liked attribute of comedy has priority over the weakly disliked attribute of horror. Dante's Peak is not included because it has a rating of

20

10 R. Night at the Opera is not included because it is not closed-captioned. ER is not within the narrow scope because it does not have any liked attributes. It is at best, neutral. An illustrative program listings screen that may be displayed by the program

25

15 guide client with such limited program guide data is shown in FIG. 16c.

30

The program guide may also provide users with an opportunity to schedule reminders using boolean or natural language expressions having one or more

20 criteria. If desired, program guide server 25 may schedule reminders based on user preference profiles and agents. Reminders may be scheduled for individual programs or series of programs. Systems in which reminders are set for series of programs are described,

35

40 25 for example, in Knudson et al. U.S. patent application Serial No. 09/330,792, filed June 11, 1999 (Attorney Docket No. UV-56), which is hereby incorporated by reference herein in its entirety.

45

A user may indicate a desire to schedule a

30 reminder by, for example, selecting a selectable Reminders feature 106 from main menu 100 of FIG. 5. In response, the program guide may display a criteria screen. Illustrative criteria screens 161 and 169 are

50

55

5

- 38 -

10

15

20

25

shown in FIGS. 17a and 17b. The program guide client may display criteria screen 161 of FIG. 17a to provide a user with an opportunity to set reminders according to a boolean type expression. The user may construct a boolean expression by selecting criteria such as attribute types, attributes, and logical operators. The user may make such selections, for example, using any suitable combination of right, left, up, or down arrow key sequences to sequence through the attribute types, attributes and logical operators. In the example of FIG. 17a, the user has defined a boolean expression to schedule reminders for comedies that star Gary Shandling and that have a rating less than R. In the example of FIG. 17b, the user has defined a similar natural language expression.

30

35

40

45

The program guide client may submit the user defined boolean or natural language expression to program guide server 25 for processing. Program guide server 25 may process the expression and schedule reminders for all of the programs that meet the expression. Program reminders may be scheduled using any suitable approach. In one suitable approach, program guide server 25 may store program identifiers and air times and send messages to the program guide client at an appropriate time before a program starts. In another suitable approach, program guide server 25 may process an expression and provide program identifiers and air times to the program guide client. The program guide client may, for example, maintain a list of program identifiers and display program reminders at an appropriate time before the programs start.

50

55

5

- 39 -

10

15

20

25

30

35

40

45

50

55

The program guide may remind a user that a program is airing at the time a program airs. In an alternative approach, the program guide may remind a user at some predetermined period of time before the program airs that a program is going to air. FIGS. 18 and 19 show illustrative program reminder lists 171. In FIG. 18, reminder list 171 is overlaid on top of the currently display television program to provide a user with the opportunity to view a reminder while still viewing a portion of the television program that a user is watching. In FIG. 19, reminder list 171 is shown overlaid on top of a program listings display screen. The program guide may provide a user with an opportunity to scroll through reminder list 171 by, for example, using remote control arrow keys. The program guide may hide the reminder list when, for example, a user selects hide reminder feature 172. The guide may also display reminder list 171 if, for example, the user presses an "OK" key at any time while watching TV.

The program guide may also provide users with an opportunity to schedule programs for recording by secondary storage device 47 or digital storage device 49 (FIG. 4) using boolean or natural language expressions. If desired, program guide server 25 may schedule programs for recording based on user preference profiles or agents. Programs may also be scheduled for recording by program guide server 25. Program guide systems in which programs are recorded by a remote server are described, for example, in Ellis et al. U.S. patent application Serial No. 09/332,244, filed June 11, 1999 (Attorney Docket No. UV-84), which is hereby incorporated by reference herein in its entirety.

5

- 40 -

10

15

20

25

30

35

40

45

50

55

A user may indicate a desire to schedule a program for recording by, for example, selecting a selectable Record feature 106 from main menu 102 of FIG. 5. In response, the program guide may display a criteria screen, such as illustrative criteria screens 161 and 169 of FIGS. 17a and 17b. The program guide client may display criteria screen 161 of FIG. 17a to provide a user with an opportunity to schedule a program for recording according to a boolean type expression. The user may construct a boolean expression by selecting criteria such as attribute types, attributes, and logical operators. The user may make such selections, for example, using any suitable combination of right, left, up, or down arrow key sequences to sequence through the attribute types, attributes and logical operators. In the example of FIG. 17a, the user has defined a boolean expression to schedule for recording comedies that star Gary Shandling and that have a rating less than R. In the example of FIG. 17b, the user has defined a similar natural language expression with similar criteria.

The program guide client may submit the user defined boolean or natural language expression to program guide server 25 for processing. Program guide server 25 may process the expression and schedule all of the programs that meet the expression for recording. Recording by program guide server 25 may be performed, for example, as described in above-mentioned Ellis et al. U.S. patent application Serial No. 09/332,244, filed June 11, 1999 (Attorney Docket No. UV-84). In another suitable approach, program guide server 25 may process the expression and provide program identifiers and air times to the program guide client. The program

5

- 41 -

guide client may, for example, maintain a list of
program identifiers and program air times and may
instruct optional secondary storage device 47 or
digital storage device 49 to record the programs.

5 The program guide may also provide users with
an opportunity to parentally control titles, programs,
or channels using boolean or natural language
expressions. If desired, program guide server 25 may
parentally control programs based on user preference
10 profiles. A user may indicate a desire to parentally
control titles, programs, or channels by, for example,
selecting a selectable Parents feature 106 from main
menu 102 of FIG. 5. In response, the program guide may
display a criteria screen, such as illustrative
25 criteria screens 161 and 169 of FIGS. 17a and 17b. The
program guide client may display criteria screen 161 of
FIG. 17a to provide a user with an opportunity to
control programs, for example, according to a boolean
type expression. The user may construct a boolean type
20 expression by selecting criteria such as attribute
types, attributes, and logical operators. The user may
make such selections, for example, using any suitable
combination of right, left, up, or down arrow key
sequences to sequence through the attribute types,
30 attributes and logical operators. In the example of
FIG. 17a, the user has defined a boolean expression to
lock out comedies that star Gary Shandling and that
have a rating less than R. In the example of FIG. 17b,
the user has defined a similar natural language
45 expression with similar criteria.

The program guide client may submit the user
defined boolean or natural language expression to
50 program guide server 25 for processing. Program guide

55

5

- 42 -

10

server 25 may process the expression, determine all of the programs that meet the expression, and indicate the programs that are locked to the program guide client when providing program listings to the program guide

15

5 client using a suitable indicator (e.g., "locked" tag contained in the listings information). The program guide client may, for example, indicate that a program is locked by displaying lock indicator 161 when displaying locked listings in a listing screen, as

20

10 shown, for example, in FIG. 7. By placing the processing and storage burdens of locking programs on program guide server 25 instead of user television equipment 22, more titles may be locked than would otherwise because of the limited processing and storage resources of user television equipment 22. If desired, titles, programs, or channels may also be locked using conventional parental control techniques. Program guide systems that provide users with an opportunity to parentally control titles, programs, or channels are described, for example, in above-mentioned Knudson et al. U.S. patent application Serial No. 09/357,941 filed July 16, 1999 (Attorney Docket No. UV-114).

25

30

35

Program guide server 25 may also record the viewing histories of users on storage device 56.

40

25 Viewing histories may be created using any suitable approach. The program guide client may, for example, keep track of all of the programs that a user watches for longer than a predefined time, and record the household that the guide client is running in, the current active preference profile or profiles, the program (or its identifier), and how long the user watched the program. The program guide client may also track when users order pay-per-view programs, record

45

50

55

5

- 43 -

10

programs, and schedule reminders for programs, and may also provide this information to program guide server 25 as part of the viewing histories. Other types of information may also be included in the viewing

15

5 histories. User defined expressions, for example, may be stored by program guide server 25 to track what types of programs users search for. In addition, user demographic values may be calculated by program guide server 25 and used to more accurately target

20

10 advertisements or recommend programs. Systems in which user demographic values are calculated are described, for example, in Knudson et al. U.S. patent application Serial No. 09/139,777, filed August 25, 1998 (Attorney Docket NO. UV-58), which is hereby incorporated by

25

15 reference herein in its entirety.

30

The program guide client may provide the viewing history information to program guide server 25 continuously (e.g., each time the program guide client determines that a user has watched a program for the

20 predefined time), periodically, in response to polls or requests from program guide server 25, or with any other suitable frequency. If desired, the program guide client may also monitor advertisement usage, such as what selectable advertisements users have selected.

35

40

25 Program guide systems in which user viewing activities and advertisement usage are tracked are described, for example, in Thomas et al. U.S. patent application Serial No. 09/139,798, filed August 25, 1998 (Attorney Docket No. UV-57), which is hereby incorporated by

45

30 reference herein in its entirety.

50

The program guide may process user profiles along with the viewer histories to present a more customized viewing experience to the user. The program

55

5

- 44 -

10

guide may, for example, identify which programs or series episodes users have watched. Program guide server 25 may, for example, identify episodes that users have not yet watched and may indicate such

15

5 episodes to the program guide client when the program guide client requests program listings. The program guide client in turn may indicate that a program is new to a household by, for example, displaying a suitable icon or changing the display characteristics of a listing (e.g., changing its color). FIG. 7 shows, for example, the display of New indicator 159 in list 129 to indicate to a user that the user has not seen a particular episode of Saturday Night Live. Program guide server 25 may also calculate ratings, such as Nielsen ratings, based on the viewing histories and provide such information to interested parties.

20

25

30

The program guide may also use the viewing history and user preferences to target the user with advertisements. Program guide systems in which users are targeted with advertisements are described, for example, in Knudson et al. U.S. patent application Serial No. 09/034,939, filed March 4, 1998 (Attorney Docket No. UV-42), which is hereby incorporated by reference herein in its entirety. Targeted

35

40

25 advertisements may contain text, graphics, or video. Targeted advertisements may also be active objects containing various user-selectable options. For example, a targeted advertisement may allow the user to request that additional information on a product be mailed to the user's home, may allow the user to purchase a product, or may allow the user to view additional information on a product using the program guide. Targeted advertisements may be displayed in any

45

50

55

5

- 45 -

10

suitable program guide display screen. The program guide client may, for example, display targeted advertisements in criteria or profile screens based on a displayed criteria, profile, or agent. Selectable advertisements 108 and advertisement banner 110, for example, may be targeted advertisements.

15

20

The program guide may make personalized viewing recommendations based on the viewing histories, preference profiles, or any suitable combination thereof. Program guide server 25 may, for example, construct relational database expressions from the viewing histories that define expressions for the program categories and ratings for programs that users have watched, scheduled reminders for, searched for, or ordered the most. Program guide server 25 may then apply user preference profile criteria to the programs, and generate personal viewing recommendations. In still another suitable approach, program guide server 25 or the program guide client may filter viewing recommendations that are generated by main facility 12 or television distribution facility 16 based on similar expressions, profiles, viewing histories, etc.

25

30

35

40

45

Assume, for the purpose of illustration, that a user has run the expression illustrated in FIGS. 9a and 9b, and has set the user profiles of FIGS. 13a-13f, program guide server 25 may determine that the movie Armageddon meets the criteria of the expression that was run, and also meets the criteria of the current user profile. Armageddon is a movie (strong like), an action (strong like), and does not have an illegal rating (it is rated PG-13). Program guide server 25 may indicate the movie Armageddon (or its identifier) and its air time to the program guide client and

50

55

5

- 46 -

10 indicate to the client (e.g., using a second
10 identifier) that a viewer recommendation for the movie
is to be displayed. The program guide client may
display a viewer recommendation overlay, such as
5 overlay 2111 shown in FIGS. 20a and 20b, over a program
15 the user is watching or over a program guide display
screen, respectively. The user may press a suitable
key on remote control 40 (e.g., an "info" key) to
access additional information for a recommended
20 program. An illustrative additional information screen
is shown in FIG. 20c. Additional program information
screens are described, for example, in above-mentioned
25 Knudson et al. U.S. patent Application Serial
No. 09/357,941 filed July 16, 1999 (Attorney Docket
15 No. UV-114). The program guide client may tune user
television equipment 22 to the channel on which a
recommended viewing is aired when, for example, a user
30 selects "Yes". If desired, recommendations may include
a suitable graphic, such as a graphic indicating the
20 recommended program.

35 FIGS. 21-24 show flowcharts of illustrative
steps involved in performing various aspects of the
present invention. The steps shown in FIGS. 21-24 are
only illustrative, and may be performed in any suitable
40 25 order.

FIG. 21 shows a flowchart of illustrative
steps involved in storing preference profiles on
program guide server 25. If desired, the steps shown
45 may be performed in a client-server interactive program
30 guide system in which users are not required to
navigate the Internet. At step 2000, the program guide
client running on user television equipment 22 provides
50 a user with an opportunity to define a preference

55

5

- 47 -

10

15

20

profile. The preference profile may include user selected or defined levels of desirability of various program characteristics, such as genre and rating. Users may define preference profiles by, for example, selecting a profile (step 2002) and selecting criteria (step 2004) such as attribute types (step 2006) and attributes (step 2008). Preference profiles may, for example, be created as database files (e.g., SQL files) containing suitable database expressions that are provided to program guide server 25. Program guide server 25 may store the preference profiles at step 2012.

25

30

35

40

45

Program guide data is provided from program guide server 25 to the program guide client and is displayed by the program guide client at steps 2020 and 2030, respectively. Program guide server 25 or the program guide client may use preference profiles to filter out undesirable program guide data. This may be accomplished using any suitable approach. Program guide server 25 may, for example, only provide program listings information or other program guide data that meets the preference profile or profiles to the program guide client (step 2025). Alternatively, program guide server 25 may provide program guide data, other information, or videos to the program guide client and the program guide client may filter the data, other information, or videos by displaying only those elements that meet the preference profile or profiles (step 2035).

50

Program guide server 25 may perform additional functions based on preference profiles if desired. Program guide server 25 may, for example, lock programs according to preference profiles (step

55

5

- 48 -

10

2040), automatically record programs according to preference profiles (step 2050), schedule reminders based on preference profiles (step 2060), or target advertising based on preference profiles (step 2070).

15

viewing recommendations based on preference profiles at step 2080. Step 2080 may also include filtering viewing recommendations based on preference profiles provided by main facility 12 or television distribution facility 16 (step 2085).

20

FIG. 22 is a flowchart of illustrative steps involved in providing users with an opportunity to search program guide data in accordance with the principles of the present invention. If desired, the steps shown may be performed in a client-server interactive program guide system in which users are not required to navigate the Internet. At step 2100, the program guide client provides a user with an opportunity to define an expression, such as a boolean or natural language expression. This may include, for example, providing a user with an opportunity to select attribute types, attributes, and logical operators (steps 2102, 2104, and 2106, respectively). The user may also be provided with an opportunity to save the expression as an agent (step 2110). The program guide client provides the expression to program guide server 25 for processing at step 2120. The program guide client may for example, provide a boolean or natural language expression in a text file. Alternatively, the program guide client may construct suitable database expressions and provide the expressions to program guide server 25 as one or more suitable database files (e.g., as SQL files).

25

15

25

30

35

40

45

50

30

35

40

45

50

55

5

- 49 -

10

15

20

25

30

35

40

45

50

55

If the user indicated a desire to save an expression as an agent at step 2110, program guide server 25 may save the expression as an agent at step 2130. Otherwise, program guide server 25 may process the expression (step 2140) using any suitable approach. This may depend on how the expression was provided by the program guide client. If boolean or natural language expressions were provided as text files, for example, program guide server 25 may parse the expressions and construct a suitable database expression. Alternatively, database expressions may have been provided by the program guide client. In either approach, program guide server 25 may search its database or databases at other facilities for program guide data (e.g., program listings, additional program information, etc.), other information (e.g., software, Internet links, etc.), or videos (e.g., video-on-demand videos) and may provide the results to the program guide client at step 2150. At step 2160 the program guide client may display the results on user television equipment 22.

If the user indicated a desire to save the expression as an agent at step 2110. Program guide server 25 may save the expression as an agent using any suitable approach. Agents may be maintained, for example, in a database that program guide server 25 monitors periodically. If desired, the agent may be forwarded to other servers at other facilities, thereby providing a user with the ability to monitor multiple databases for program guide data, other information, or videos. Agents may be run automatically (e.g., databases may be queried) on one or more servers at step 2145. Step 2145 may be performed periodically,

5

- 50 -

10

each time a database is updated, or with any other suitable frequency. Program guide server 25 may provide its results and the results of other servers (if desired) to the program guide client at step 2155.

15

5 The program guide client may display the results at 2165. The results may be displayed, for example, in the form of reminders for which reminder information was provided at step 2155.

20

FIG. 23 shows a flowchart of illustrative 10 steps involved in processing and using expressions on program guide server 25 in accordance with the principles of the present invention. If desired, the steps shown may be performed in a client-server 25 interactive program guide system in which users are not required to navigate the Internet. The program guide 15 client provides users with an opportunity to define an expression (e.g., boolean or natural language expressions) at step 2100. This may include, for 30 example, providing a user with an opportunity to select attribute types, attributes and logical operators 20 (steps 2102, 2104, and 2106, respectively). The program guide client provides the expression to program 35 guide server 25 for processing at step 2210 as any suitable type of file. The program guide client may 40 for example, provide a boolean or natural language expression in a text file. Alternatively, the program guide client may construct suitable database expressions and provide the expressions to program 45 guide server 25 as one or more suitable database files 30 (e.g., as SQL files).

35

45

50

Program guide server 25 may process the expression (step 2220) using any suitable approach depending on how the expression was provided to program

55

5

- 51 -

10

15

20

guide server 25 from the program guide client. If boolean or natural language expressions were provided as text files, for example, program guide server 25 may parse the expressions and construct a suitable database expression. Alternatively, database expressions may have been provided to program guide server 25 from the program guide client. In either approach, program guide server 25 may search its database or databases at other facilities and may provide the results to the program guide client or use the results to perform any suitable program guide function.

25

30

35

Reminders may be scheduled based on the results of the search (step 2230). Program guide server 25 may, for example, store reminder information (e.g., program identifiers and air times) at step 2235 and send messages to the program guide client at an appropriate time before a program starts. In another suitable approach, program guide server 25 may process an expression and provide program identifiers and air times to the program guide client. The program guide client may, for example, maintain a list of program identifiers and display program reminders at an appropriate time before the programs start.

40

45

50

Programs may also be automatically recorded by program guide server 25 or user television equipment 22 based on the results of the expression (step 2240). Program guide server 25 may, for example, provide program identifiers and air times to the program guide client. The program guide client may, for example, maintain a list of program identifiers and program air times and may instruct optional secondary storage device 47 or digital storage device 49 to record the programs at the appropriate time.

55

5

- 52 -

10

Programs may be parentally locked based on the expression results (step 2250). Program guide server 25 may, for example, store parental control information (e.g., program identifiers in a database, table, or list of programs to be locked) at step 2260. Program guide server 25 may indicate to the program guide client that programs are locked when providing program listings to the program guide client.

15

20

Alternatively, program guide server 25 may indicate to the program guide client the programs that were found as a result of the expression. The program guide client may lock the programs locally using any suitable approach. The program guide client may, for example, indicate that a program is locked by displaying lock indicator 161 when displaying locked listings in a listing screen, as shown, for example, in FIG. 7.

25

30

FIG. 24 shows a flowchart of illustrative steps involved in tracking and using viewing histories in accordance with the principles of the present invention. If desired, the steps shown may be performed in a client-server interactive program guide system in which users are not required to navigate the Internet. Viewing histories are tracked at step 2300. This may include tracking programs that users watch (step 2310), tracking reminders scheduled by a user with program guide server 25 or using conventional techniques (step 2320), tracking pay-per-view programs that the user orders (step 2330), advertisement usage (step 2335), track recorded programs (step 2337), track any other suitable user activity, or any suitable combination thereof. The program guide client may provide the viewing history information to program guide server 25 continuously (i.e., each time the

35

40

45

50

55

5

- 53 -

10

program guide client determines that a user has watched a program for the predefined time), periodically, in response to polls or requests from program guide server 25, or with any other suitable frequency.

15

5 The viewing history tracked in steps 2310-2335 may be stored on program guide server 25 at step 2340. If desired, user-defined expressions that are processed by program guide server 25 may also be stored on program guide server 25 (step 2345). User
20 10 demographic values may be calculated by program guide server 25 at step 2347. The viewing history and its expressions and user demographic values may be used by program guide server 25 to perform any suitable
25 function. Program guide server 25 may, for example, 15 collect program rating information (step 2350), or target advertising (step 2360).

30

Program guide server 25 may search its or another server's database for programs that are consistent with the viewing history (step 2370). If
20 20 desired, program guide server 25 may find programs that are also consistent with preference profiles stored by program guide server 25 (step 2375). Program guide server may perform any suitable function using the
35 results of the search. Program guide server 25 may, 40 25 for example, identify episodes of programs that are new to a user (step 2380), or provide viewing recommendations in the form of, for example, reminders or recommendations for non-program items (e.g.,
45 software, Internet links, etc.) (step 2390).

45

30 The foregoing is merely illustrative of the principles of this invention and various modifications can be made by those skilled in the art without
50 departing from the scope and spirit of the invention.

50

55

Claims

5

10

15

20

25

30

35

40

45

50

55

5

- 54 -

What is claimed is:

10

1. A method for use in a client-server interactive television program guide system comprising:
providing a user with an opportunity to define user preferences using an interactive television program guide client that is implemented on user television equipment, without requiring the user to navigate the Internet;
providing the user preferences to a program guide server; and
providing program guide data to the program guide client according to the user preferences.

15

20

25

2. The method defined in claim 1 further comprising:
generating a viewing recommendation based on the user preferences with the program guide server; and
displaying the user preferences with the interactive television program guide client on the user television equipment.

30

35

40

3. The method defined in claim 1 wherein providing a user with an opportunity to define user preferences comprises providing a user with an opportunity to designate a preference level for a plurality of preference attributes.

45

4. The method defined in claim 1 further comprising providing software to the program guide client according to the user preferences.

50

55

5

- 55 -

10

5. The system defined in claim 1 further comprising providing Internet links to the program guide client according to the user preferences.

15

6. A method for use in a client-server interactive television program guide system for scheduling reminders according to user defined expressions, comprising:

20

providing a user with an opportunity to define an expression with an interactive television program guide client implemented on user television equipment without requiring the user to navigate the Internet;

25

storing the expression on a program guide server;

30

processing the expression with the program guide server to find programs that satisfy the expression; and

scheduling with the program guide server reminders for programs that satisfy the expression.

35

7. The method defined in claim 6 wherein scheduling with the program guide server reminders for programs that satisfy the expression comprises providing at least one message from the program guide server to the program guide client before each of the programs that satisfy the expression begin.

40

45

8. The method defined in claim 6 wherein scheduling with the program guide server reminders for programs that satisfy the expression comprises providing program identifiers for each of the programs

50

55

5

- 56 -

10

that satisfy the expression from the program guide server to the program guide client.

15

9. A method for use in a client-server interactive television program guide system for scheduling programs for recording according to user defined expressions, comprising:

20

providing a user with an opportunity to define an expression with an interactive television program guide client implemented on user television equipment without requiring the user to navigate the Internet;

25

storing the expression on a program guide server;

30

processing the expression with the program guide server to find programs that satisfy the expression; and

scheduling with the program guide server the programs that satisfy the expression for recording.

35

10. The method defined in claim 9 wherein scheduling with the program guide server the programs that satisfy the expression for recording comprises scheduling with the program guide server the programs that satisfy the expression for recording by the user television equipment.

40

45

11. The method defined in claim 9 wherein scheduling with the program guide server the programs that satisfy the expression for recording comprises scheduling with the program guide server the programs that satisfy the expression for recording by the program guide server.

50

55

5

- 57 -

10

12. A method for use in a client-server interactive television program guide system for parentally controlling programs according to user defined expressions, comprising:

15

providing a user with an opportunity to define an expression with an interactive television program guide client implemented on user television equipment without requiring the user to navigate the Internet;

20

storing the expression on a program guide server;

25

processing the expression with the program guide server to find programs that satisfy the expression; and

locking with the program guide server programs that satisfy the expression.

30

13. The method defined in claim 12 wherein locking with the program guide server programs that satisfy the expression comprises indicating to the program guide client that the programs that satisfy the expression are locked.

35

40

14. A method for use in a client-server interactive television program guide system for tracking a user's viewing history, comprising:

45

tracking a user's viewing history;
storing the user's viewing history on a program guide server;

finding programs with the program guide server that are consistent with the user's viewing history; and

50

indicating on user television equipment the programs found by the program guide server that are consistent with the user's viewing history and that the

55

5

- 58 -

10

user has not watched, with an interactive television program guide client implemented on the user television equipment.

15

15. The method defined in claim 14 wherein storing the user's viewing history comprises storing a user defined expression with the program guide server.

20

16. The method defined in claim 14 wherein storing the user's viewing history comprises calculating user demographic values with the program guide server.

25

17. The method defined in claim 14 further comprising:

30

providing a user with an opportunity to define a user preference profile with the interactive television program guide client implemented on user television equipment;

35

storing the user preference profile on a program guide server; and

40

finding programs with the program guide server that are consistent with the user preference profile, wherein:

45

indicating on user television equipment the programs found by the program guide server that are consistent with the user's viewing history and that the user has not watched comprises indicating on user television equipment the programs found by the program guide server that are consistent with the user's viewing history and the user preference profile and that the user has not watched.

50

18. The method defined in claim 14 further comprising:

55

5

- 59 -

10

targeting advertising with the program
guide server based on the user's viewing history; and
displaying the advertising with the
interactive television program guide client on the user
television equipment.

15

20

19. The method defined in claim 14 further
comprising collecting program ratings information with
the program guide server based on the user's viewing
history.

25

20. A client-server interactive television
program guide system comprising:

30

means for providing a user with an
opportunity to define user preferences using an
interactive television program guide client that is
implemented on user television equipment, without
requiring the user to navigate the Internet;
means for providing the user preferences
to a program guide server; and

35

means for providing program guide data
from the program guide server to the program guide
client according to the user preferences.

40

21. The system defined in claim 20 further
comprising:

45

means for generating a viewing
recommendation based on the user preferences with the
program guide server; and

50

means for displaying the user
preferences with the interactive television program
guide client on the user television equipment.

55

22. The system defined in claim 20 wherein
the means for providing a user with an opportunity to

03/17/99
Jc640 U.S. PTO

PROVISIONAL APPLICATION COVER SHEET

A/prov

This transmittal and the documents and/or fees itemized hereon and attached hereto have been deposited as "Express Mail Post Office to Addressee" in accordance with 37 C.F.R. §1.10 with Express Mail Mailing Label Number EL285395871US

Attorney Docket No.: SRI1P023+

First Named Inventor: CHEYER, Adam J.

Assistant Commissioner for Patents
Box Patent Application
Washington, DC 20231

Duplicate for fee processing

Jc541 U.S. PTO
60/124718
03/17/99

Sir: This is a request for filing a PROVISIONAL APPLICATION under 37 CFR 1.53(c).

INVENTOR(S)/APPLICANT(S)

LAST NAME	FIRST NAME	MIDDLE INITIAL	RESIDENCE (CITY AND EITHER STATE OR FOREIGN COUNTRY)
CHEYER JULIA	Adam Luc	J. E.	Menlo Park, CA Menlo Park, CA

TITLE OF INVENTION (280 characters max)

USING A COMMUNITY OF DISTRIBUTED ELECTRONIC AGENTS TO SUPPORT A HIGHLY MOBILE, AMBIENT COMPUTING ENVIRONMENT

CORRESPONDENCE ADDRESS

HICKMAN STEPHENS & COLEMAN, LLP
P.O. Box 52037
Palo Alto, CA 94303-0746
(650) 470-7430

ENCLOSED APPLICATION PARTS (check all that apply)

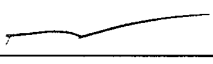
Specification Number of Pages _____ Small Entity Statement
 Drawing(s) Number of Sheets _____ Other (specify) 13 Pages of White Paper Article

A check or money order is enclosed to cover the Provisional filing fees. Provisional Filing Fee Amount \$150

The commissioner is hereby authorized to charge any additional fees which may be required or credit any overpayment to Deposit Account No. 50-0384 (Order No. SRI1P023+).

The inventions made by an agency of the United States Government or under a contract with an agency of the United States Government.
 No Yes, the name of the U.S. Government agency and the contract number are: _____

Respectfully Submitted,

SIGNATURE 
TYPED NAME Brian R. Coleman

DATE 3/17/99
REGISTRATION NO. 39,145

PROVISIONAL APPLICATION FILING ONLY

TITLE OF THE INVENTION:

Using a Community of Distributed Electronic Agents to Support a Highly Mobile, Ambient Computing Environment

ABSTRACT

Douglas Engelbart asked 30 years ago, at SRI: How can knowledge workers (both individuals and groups) get maximum leverage from personal, networked, interactive computing devices? The twist in the present invention is to redirect this inquiry to the emerging post-desktop world of ubiquitous, highly mobile "information appliances" and PDA's. For example, what sort of computing environment will best serve the PDA-equipped knowledge worker away from the desktop in his/her car, airplane seat, or in a conference room with others? And what software architecture is required to provide that environment effectively? We believe that an "OAA-style" architecture (facilitated collaboration among distributed agents with declared capabilities in a high-level interagent communication language) has tremendous potential for addressing this challenge. The present invention envisions a new application of this collaborative architecture to address the post-desktop, mobile/ubiquitous computing environment, by incorporating elements like: (a) GPS agents, (b) speech recognition (+ other hands-free UI, multi-modal UI), and (c) opportunistic connectivity among meeting participants (e.g., think of docked or IR-linked PDA's, not just Internet sites). In the specific context of such emerging, ambient computing environments, the distinctive advantages of OAA-style architecture (contrasted with lower-level distributed object approaches like CORBA standing alone), especially with respect to hands-free and multi-modal UI, are even more pronounced.

SUPPLEMENTAL INFORMATION

A November, 1988 OZCHI paper written by Adam Cheyer and Luc Julia entitled: "Cooperative Agents and Recognition Systems (CARS) for Drivers and Passengers" (copy attached) illustrates one example of a possible automobile-based realization of this invention, including GPS and multi-modal UI.

The attached OAA "Scenario" one-page PowerPoint slide illustrates some scenarios for potential interaction and collaboration among PDA-holders in a non-desktop environment like a car (or a conference room), using the technology of the present invention.

A description of multi-modal whiteboard-style collaboration (entitled "SCRIBE") is also attached and may be helpful in preferred embodiments of the present invention.

Pending patent application serial no. _____ assigned to SRI (docket no. 3949-2) provides a detailed description of the underlying OAA platform architecture, and also specific descriptions of several applications including "multi-modal maps" which may be helpful in preferred embodiments. The referenced pending patent application is incorporated herein by reference in its entirety.

The published paper Multimodal Maps: An Agent-based Approach, Cheyer & Julia, International Conference on Cooperative Multimodal Communication (CMC/95), 24-26 May 1995 (Eindhoven, The Netherlands), may also be useful for preferred embodiments, and is also incorporated herein by reference in its entirety.

652750" BT 4/19/95

Cooperative Agents and Recognition Systems (CARS) for Drivers and Passengers

Luc E. JULIA
 STAR Laboratory
 SRI International
 333 Ravenswood Avenue
 Menlo Park, CA 94025
 USA
 julia@speech.sri.com

Adam J. CHEYER
 Artificial Intelligence Center
 SRI International
 333 Ravenswood Avenue
 Menlo Park, CA 94025
 USA
 cheyer@ai.sri.com

Abstract

In this paper we present SRI's vision of the human-machine interface for a car environment. This interface leverages our work in human-computer interaction, speech, speaker and gesture recognition, natural language understanding, and intelligent agents architecture. We propose a natural interface that allows the driver to interact with the navigation system, control electronic devices, and communicate with the rest of the world much as would be possible in the office environment. Passengers would be able to use the system to watch TV or play games in their private spaces. The final prototype will be fully configurable (languages, voice output, and so forth), and will include speaker recognition technology for resetting preferences and/or for security.

Keywords

Multimodal Interfaces, Speech and Speaker Recognition, Gesture Recognition, Natural Language Understanding, Cooperative Agents.

1. Introduction

New technologies such as Global Positioning System (GPS), wireless phones or wireless internet and electronic controls inside cars are available to improve the way we drive and manage the time spent in our automobiles. To manage this heavy flow of data and to keep the cognitive load as low as possible for the driver, we propose a solution based on our previous developments: a small, speech-enabled, touch display device that provides a combination of the best features of several interfaces we have developed over the past few years. This device can be used according to the specific task that has to be completed by the driver or the passenger.

The interfaces we have developed are the front ends to SRI's powerful framework, the Open Agent Architecture™ (OAA) [18], which allows a community of intelligent agents to work together to achieve user goals. To build multimodal systems, the key agents are those that recognize human signals such as speech or gestures and those that extract the meaning: the natural language understanding agent and the multimodal interpretation agent, for instance.

2. Natural Interfaces

The first prototype we built using Java™ combines different reused interfaces that were chosen according to the task. For each section of the system, we reference the full project for which it was developed. The user can select the tabs using both speech or deictic gestures. Each panel provides its own vocabulary and set of commands in addition to the main commands that allow navigation between the tabs.

2.1. Navigation System

The Multimodal Maps [1] allow the user to navigate maps naturally and query associated databases using speech, handwriting, and 2D gestures in a synergistic fashion on a pen computer. Using the same interface (replacing the pen with the finger) to query the navigation system and to display the GPS information gives the driver or the passengers the ability to plan the route and to get information from local or remote databases displayed on the map ("I want to go to Menlo Park." "Show me the restaurants around here.") (Figure 1). The GPS system guides the car along the chosen route using both the map display and a text-to-speech output. The interactions among all the agents belonging to the system, even if they

do not seem to be in use by the current visual interface, enables a great degree of proactivity from the system. For example, it could ask questions such as: "The tank is almost empty; would you like to find the nearest gas station?". As well as a multimodal synergistic input interface, the system provides multimedia outputs such as iconic sounds, discriminative talking voices, images, or videos.

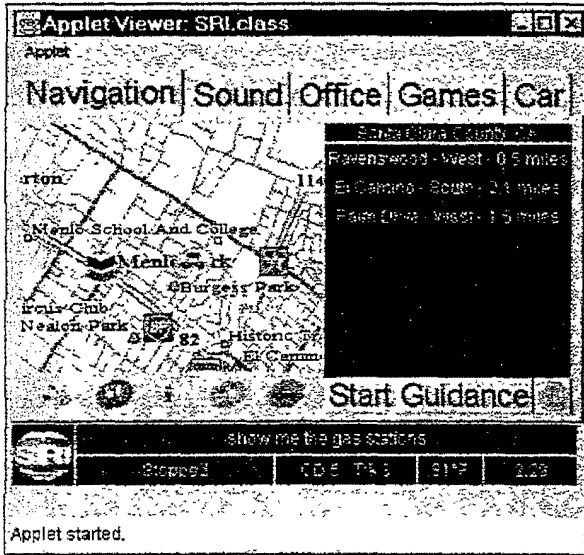


Figure 1. Navigation Panel

2.2. Electronic Device Control

Most of the cars will have numerous electronic devices that are accessible through a serial port using a predefined protocol. By connecting a computer and its multimodal interface, it will then be possible for the driver to control critical electronic devices such as cruise control or lights, and for everyone in the car to access comfort devices such as air conditioning, windows, sound, and entertainment ("Play CD 2, track 1.") (Figure 2).

Priority should be given to the driver, possibly through speaker identification. Moreover, an interesting study [11] has shown that it is also possible to use the touch screen in blind condition (for the driver) to enter simple command gestures (down arrow to turn the volume down for instance).

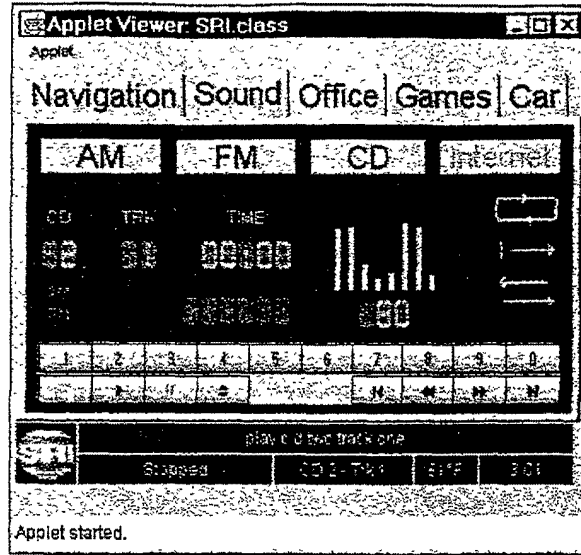


Figure 2. Sound System Panel

2.3. Communication Center

The communication center is a remote office accessible by voice (Figure 3).

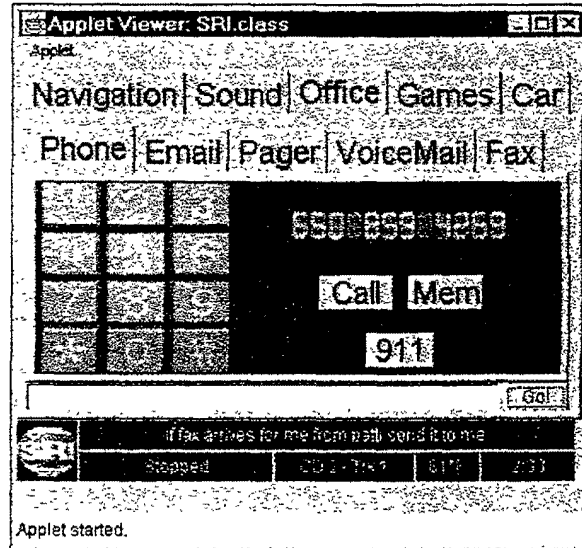


Figure 3. Communication Center Panel

This is an instance of the Automated Office (Unified Messaging), developed to show some capabilities of OAA [18]. The driver or passengers are able to browse the incoming emails by voice (even multipart/multimedia MIME messages), make phone calls, or send spoken notes. As basic features of OAA, filtering and triggering capabilities are included in each connected agent: "If email arrives for me about OzCHI, read it to me." Plugging in an agent using speaker identification tech-

662760" ST 2121205

niques allows commands such as "If voicemail arrives for me from Larry, send an email to Patti" [9]. Intelligent cross-media conversion and adaptability to the current set of available or desired output channels is a key characteristic of the Unified Messaging prototype.

2.4. Recreation Area

The recreation center gathers some of the innovative speech-enabled prototypes developed by SRI International. Passenger-oriented, it assumes that each passenger creates a private multimodal/multimedia area (close talking microphone, personal touch screen and headsets). The passenger can play impressive 3D games enhanced with speech commands, search the Internet by voice, talk to an animated avatar, and watch TV in a more interactive way by asking naturally for the available programs with specific features. In addition, speech-based educational systems, such as WebGrader™ [16], provide a fun and effective way to learn foreign languages and their pronunciations (Figure 4).

SECRET 03/24/98 03:47:59

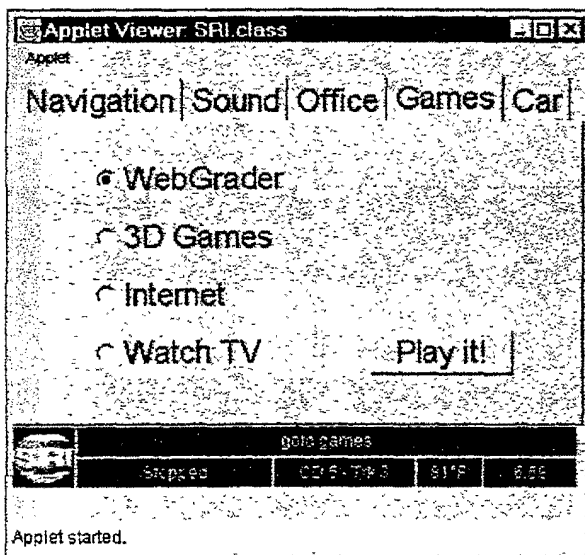


Figure 4. Recreation Panel

2.5. Technical Information Access

The entire documentation of the car will be available on the Internet, making it easy to keep it up to date and possibly to personalize (via cookies) and control (via certificates) data for the car. This section extends the idea of the dialog with an avatar, or actor, implemented using the Microsoft™ Agent graphics [19].

If a warning message appears from a monitored device, a dialog with an automobile expert, played by the actor, will help to diagnose and fix the problem (Figure 5). The

expert may also answer common questions such as "How much air should I put in my tires?" or "How should I talk to you?"

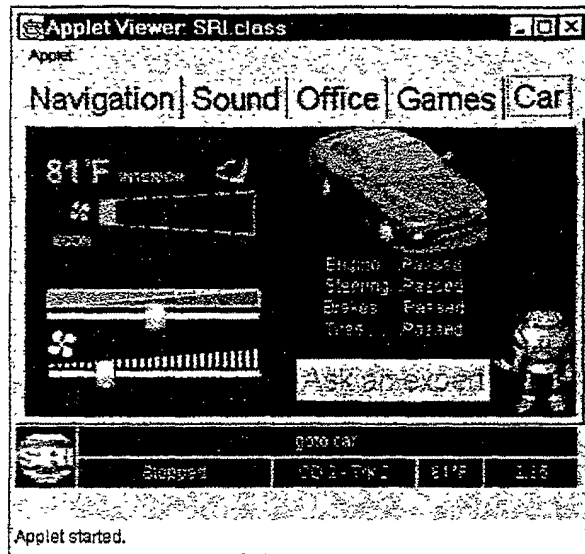


Figure 5. Diagnostic Panel

2.6. Setups

Speaker verification techniques can be used to access the setup panel and private areas (to configure and define the passwords for email and voice mail accounts, for instance). It will also be used to automatically retrieve the preferences for the current driver with respect to seat position, radio selections, temperature, mirror direction, and so forth.

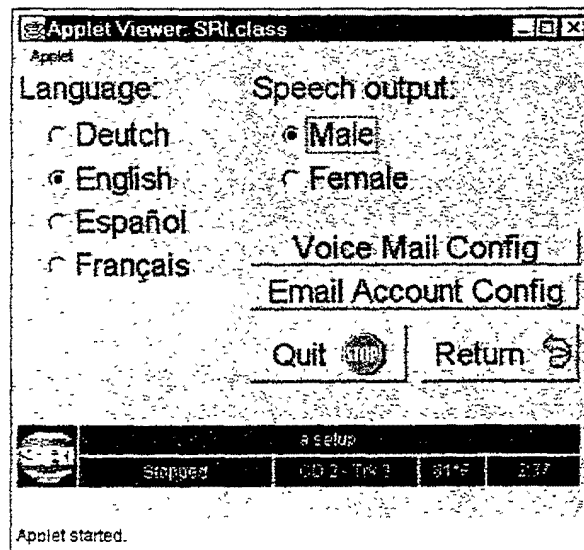


Figure 6. Setup Panel

SRI INTERNATIONAL

3. Behind the Scene: the Agents

The functionality described above requires multiple Artificial Intelligence (AI) technologies (e.g., speech and gesture recognition, natural language understanding) to interact with each other and with commercial, off the shelf components such as email systems, map databases, and car electronics. SRI's Open Agent Architecture provides an infrastructure for integrating distributed components in a more flexible way than can be done through other distributed technologies such as CORBA, COM, or Java's RMI. The key difference in OAA's approach is that instead of components writing code to specify (and fix) their interactions and dependencies with other components, each agent (component) expresses its capabilities and needs in terms of a higher-level Interagent Communication Language (ICL). Each request for information or action is handled by one or more "facilitator agents," who break the request into subtasks, allocate subtasks to agents able to perform them, and then coordinate the flow of data and control among the participants. The architecture offers built-in support for creating natural user interfaces to the distributed services, since the logic-based ICL can be translated from and to natural language; users can speak a request in English, and the request can be acted upon by the community of agents, without requiring the user to specify or even know which agents are involved.

The advantages of the OAA approach include true plug-and-play, with new agents able to join the community of services at runtime; managed coordination of cooperative and competitive parallelism among components; heterogeneous computing, with components existing on diverse platforms, written in many programming languages; and enhance code reuse. Since components do not have hard-code dependencies written into them, we will be able to incorporate many existing agents from previous OAA-enabled systems [13, 18].

4. Recognition and Interpretation

4.1. Speech Recognition

Speech recognition, along with natural language, is a huge component of the multimodal user interface. While it is possible to use any speech recognition product available on the market to make an agent, we prefer the Nuance Communications¹ recognizer. Nuance is a real-time version of the SRI STAR Laboratory's continuous speech recognition system using context-dependent genonic

¹ SRI spin-off: <http://www.nuance.com>

hidden Markov models (HMMs) [4]. This technology recognizes natural speech without requiring the user to train the system in advance (i.e., speaker-independent recognition) and can be distinguished from the few other leading-edge speech recognition technologies by its detailed modeling of variations in pronunciation and its robustness to background noise and channel distortion. We plan to investigate automobile environments in more detail.

4.2. Natural Language Understanding

In most OAA-based systems, prototypes are initially constructed with relatively simple natural language (NL) components, and as the vocabulary and grammar complexities grow, more powerful technologies can be incrementally added. It is easy to integrate different levels of NL understanding, depending upon the requirements of the system, just by plugging in an adequate engine. The available engines are two of our low-end NL systems: Nuance's template-slot tools and DCG-NL, a Prolog-based top-down parser. SRI's GEMINI [5] and FASTUS [7] are more powerful tools, used for complex NL tasks. To design the dialog on the fly, a visual tool is under development (Figure 7). It simulates the behaviors of the NL engine and creates the necessary code and data for the final NL agent.

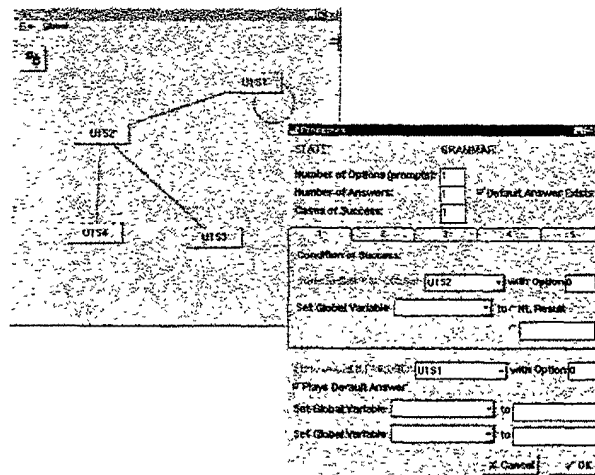


Figure 7. Visual Design Tool

4.3. Speaker Identification

Speaker identification technology has seen significant progress over the past several years. Although good performance can be achieved, several parameters affect accuracy. For example, systems trained on larger amounts of speech from the users will be more accurate. Similarly, variety in the training data (collecting over several days) will improve system robustness, and accuracy is higher for

66250 "SECRET"

longer test utterances. Computational limitations of the onboard platform will also be a performance factor. Perhaps the most significant of the factors is the variety in training data. The effects of mismatches between training and testing conditions can be dramatic [17]. A severe example, in the context of cars, of mismatching conditions would occur when a user trains the system with the engine off, in the garage, and then uses it with the top down on the freeway at high speed. We have made significant progress in reducing adverse effects of mismatches between training and testing conditions. In particular, SRI has developed a technology that reduces the effect from a factor of 30 to a factor of less than 3 [6]. The technology enables the user to train in a single session (one acoustic environment).

4.4. Gesture Recognition

The gesture modality is usually used in conjunction with speech to add spatial or deictic data to issue commands to the system. But sometimes a gesture (like a crossout) can carry both a location and semantic content. A set of current gestures (Figure 8) can be recognized using algorithms developed in [8].

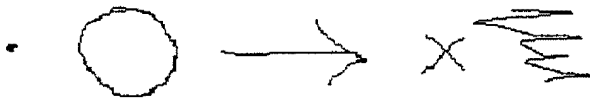


Figure 8. Gesture Set

In our experience [2], most gestures produced by users fall into this set. Since handwriting has rarely been used but we want to provide as many modalities as possible, we incorporated Communications Intelligence Corporation (CIC²) recognition routines. The handwriting recognizer is of interest in the navigation task where out-of-vocabulary names may appear, which are normally difficult for speech recognition systems to handle. Both the gesture recognizer and the handwriting recognizer are competing on the same data to find the right meaning.

4.5. Multimodal Fusion

Even if we consider speech as a privileged modality [10], numerous user studies [e.g., 12] have shown that most subjects prefer combinations of spoken and gestural inputs. In such examples, whereas speech plays a strong role in the acquisition of commands, combining it with a pointing device provides significant (8%) improvement in

performance (recognition and understanding) over the use of speech in isolation. Not surprisingly, gestures provide a fast and accurate means of locating specific objects, while voice commands are more appropriate for selecting describable sets of objects or for referring to objects not visible on the screen. Many of these studies also attempt to enumerate and classify the relationships between the modalities arriving for a single command (complementary, redundancy, transfer, equivalence, specialization, contradiction). To model interactions where blended and unsorted modalities may be combined in a synergistic fashion with little need for time stamping, we first proposed a three-slot model known as VO*V* (Figure 9), such that

V or Verb is a word or a set of words expressing the action part of a command.

O* or Object[s] is zero or more objects to which the verb applies (zero if it is a system command).

V* or Variable[s] is zero or more attributes or options necessary to complete the command.

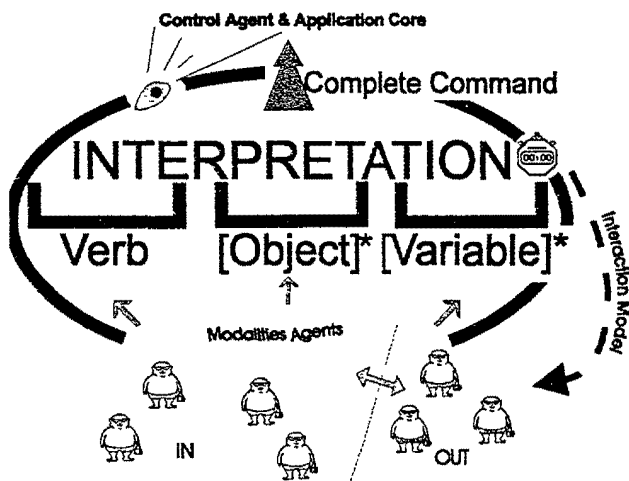


Figure 9. VO*V* Model

Input modalities produced by the user (handwriting, speech, gestures) fill slots in the model, and interpretation occurs as soon as the triplets produce a complete command. A multimedia prompting mechanism is also provided to assist the user in fulfilling an incomplete command. In addition, multiple information sources may compete in parallel for the right to fill a slot, given scored modality interpretations. This model has been shown to be easily generalizable, and has been applied to various application domains

² SRI spin-off: <http://www.cic.com/>

5. Evaluation

When building complex systems, it is important to perform user experiments to validate the design and implementation of the application. As described in [2], we have developed a novel "hybrid Wizard-of-Oz" approach for evaluating how well the implemented system functions for an experienced user, while simultaneously gathering information about future extensions or improvements as dictated by new users. The technique promotes incremental development of a complex system, from initial prototype through tested product, and provides a means for logging user interactions and quantifying system improvements at every stage of development.

6. Conclusions and Future Work

The unique feature of the proposed approach is that we integrate several very distinctive pieces, thanks to the OAA, even though they were not intended for this purpose. Further, we unify those pieces through a common, natural, multimodal interface using as much as possible human-to-human communication to avoid adding cognitive overload to the user. We achieved most of this aim by using good recognition systems and effective fusion and presentation techniques. But to improve the reliability and robustness of the speech recognizer in real cars, we still have to address considerable noise and speaker adaptation issues (see, e.g. [3, 14, 15]). Finally, within a short period of time we plan to hook up real GPS and navigation systems and install our system in a moving car so that we can conduct user testing in real-life conditions.

7. Acknowledgments

Many thanks to Patti Price, Director of the Speech Technology and Research lab who spent a lot of time helping to design the CARS system.

8. References

- [1] A. CHEYER and L. JULIA, "Multimodal Maps: An Agent-based Approach," Proc. of Cooperative Multimodal Communication (CMC'95): Eindhoven, The Netherlands, 1995.
- [2] A. CHEYER, L. JULIA and J. C. MARTIN, "A Unified Framework for Constructing Multimodal Experiments and Applications." Proc. of Cooperative Multimodal Communication (CMC'98): Tilburg, The Netherlands, 1998.

[3] V. DIGALAKIS and L. NEUMEYER, "Speaker Adaptation Using Combined Transformation and Bayesian Methods," Proc. of Intl. Conference on Acoustics, Speech and Signal Processing (ICASSP'95): Detroit, USA, 1995

[4] V. DIGALAKIS, P. MONACO and H. MURVEIT, "Genones: Generalized Mixture Tying in Continuous Hidden Markov Model-Based Speech Recognizers," IEEE Transactions of Speech and Audio Processing, Vol.4, Num. 4, 1996

[5] J. DOWDING, J.M. GAWRON, D. APPELT, J. BEAR, L. CHERNY, R. MOORE and D. MORAN, "GEMINI: A natural language system for spoken-language understanding," Proc. of 31st Annual Meeting of the Association for Computational Linguistics (ACL'96): Columbus, USA, 1996.

[6] L. HECK and M. WEINTRAUB, "Handset dependent background models for robust text-independent speaker recognition," Proc. of Intl. Conference on Acoustics, Speech and Signal Processing (ICASSP'97): Munich, Germany, 1997.

[7] J. HOBBS, D. APPELT, J. BEAR, D. ISRAEL, M. KAMEYAMA, M. STICKEL, and M. TYSON, "FASTUS: a cascaded finite-state transducer for extracting information from natural-language text," in Finite State Devices for Natural Language Processing (E. Roche and Y. Schabes, eds.) MIT Press, Cambridge, USA, 1996.

[8] L. JULIA and C. FAURE, "Pattern Recognition and Beautification for a Pen Based Interface," Proc. of Intl. Conference on Document Analysis and Recognition (ICDAR'95): Montréal, Canada, 1995.

[9] L. JULIA, L. HECK and A. CHEYER, "A Speaker Identification Agent," Proc. Audio and Video-based Biometric Person Authentication (AVBPA'97): Crans-Montana, Switzerland, 1997.

[10] L. JULIA and A. CHEYER, "Speech: A Privileged Modality," Proc. of EuroSpeech'97: Rhodes, Greece, 1997.

[11] J. F. KAMP, F. POIRIER and P. DOIGNON, "A New Idea to Efficiently Interact with In-Vehicle Systems. Study of the use of the Touchpad in "Blind Condition," Poster Proc. of Human Computer Interaction (HCI'97): San Francisco, USA, 1997.

[12] B.A. MELLOR, C. BABER and C. TUNLEY, "In goal-oriented multimodal dialogue systems," Proc. of Intl. Conference on Spoken Language Processing (ICSLP'96): Philadelphia, USA, 1996.

[13] D. MORAN, A. CHEYER, L. JULIA, D. MARTIN and S. PARK, "The Open Agent Architecture and Its Multimodal User Interface," Proc. of Intelligent User Interfaces (IUI'97): Orlando, USA, 1997.

[14] L. NEUMEYER and M. WEINTRAUB, "Probabilistic Optimum Filtering for Robust Speech Recognition," Proc. of Intl. Conference on Acoustics, Speech and Signal Processing (ICASSP'94): Adelaide, Australia, 1994.

[15] L. NEUMEYER and M. WEINTRAUB, "Robust Speech Recognition in Noise Using Adaptation and Mapping Techniques," Proc. of Intl. Conference on Acoustics, Speech and Signal Processing (ICASSP'95): Detroit, USA, 1995.

[16] L. NEUMEYER, H. FRANCO, V. ABRASH, L. JULIA, O. RONEN, H. BRATT, J. BING and V. DIGALAKIS, "WebGrader: A multilingual pronunciation practice tool," Proc. of Speech Technology in Language Learning (STILL'98): Stockholm, Sweden, 1998.

[17] NIST Speaker Recognition Workshop: Linthicum Heights, USA. 1996.

[18] SRI International web site on the Open Agent Architecture:
<http://www.ai.sri.com/~oaa/applications.html>

[19] Microsoft web site about their agents and their animations:
<http://www.microsoft.com/workshop/imedia/agent/>

SECRET

SRI INTERNATIONAL

Microsoft Property Office

P-3967

Copyright 1998 IEEE. Published in the Proceedings of OZCHI'98,
29 November - 3 December 1998 in Adelaide, South Australia.
Personal use of this material is permitted. However, permission to
reprint/republish this material for advertising or promotional
purposes or for creating new collective works for resale or
redistribution to servers or lists, or to reuse any copyrighted
component of this work in other works, must be obtained from the IEEE.
Contact: Manager, Copyrights and Permissions / IEEE Service
Center / 445 Hoes Lane / P.O. Box 1331 / Piscataway, NJ
08855-1331, USA. Telephone: + Intl. 732-562-3966.

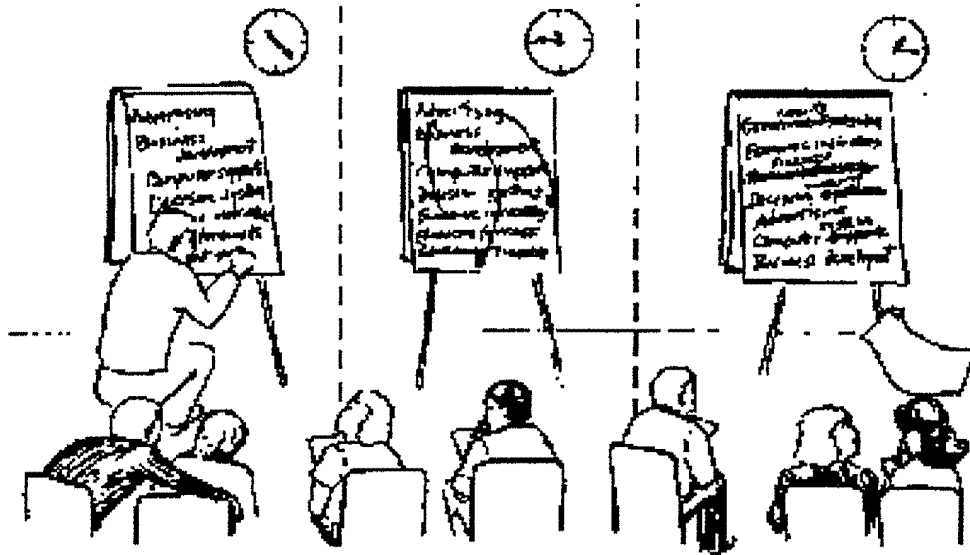
56470"BT42T03

SRI INTERNATIONAL

A first step toward the MAGIC*: SMARTMeetings

You don't have to be miles away to collaborate, but if you are that's OK!

PAST



552750-8742103

SRI INTERNATIONAL

INTELLECTUAL PROPERTY OFFICE

VISION



The SMARTMeetings room: each seat provides an internet connection and a pen tablet. The large screen display is the shared space. A broader vision accepts heterogeneous machines (PCs, PalmPilots, etc...)

The SCRIBE system, a reactive board metaphor. It gives immediate, beautified, feedback on the collaborative space for handwriting and drawings. Erase/Correction functions are available through natural gestures. Usage of colors possible.

The leader of the meeting, a privileged user, interacts with the board at the podium, and can also gather information from the other participants by giving them electronic access to the board in order to share their ideas. He or she is the facilitator.

More informal meetings as well as many meeting styles are falling in this paradigm. For instance, every attendant might be a privileged user who talks, writes and draws carefully around the table in order to be well understood by the other participants, and the recognizers. The shared, collaboratively built, document is projected

Location of the attendants is indifferent, they need a connected pen/multimedia computer.

SRI INTERNATIONAL

Automatic production of clean documents, minutes, etc... History of production available. Immediate distribution, copies are stored on attendants computers. Users can import pre-meeting notes, specific backgrounds (maps, charts, etc...) in order to produce nicer documents. But we anticipate that most of the time all data will be produced on site, during the meeting.

TECHNOLOGIES

Collaborative Application.

Handwriting Recognition.

Drawing Recognition - Boxes, charts, tables... - Meta gestures (erase, move, etc...).

Speech Recognition (? Or limited) - Speaker ID.

(?Stereo) Vision - 3D gestures, meeting layout.

OUTCOMES

Working prototype.

Patent for the (ri's ollaborative, eactive and ntelligent oard nhancer).

BACKGROUND

The CDL is a good start. Tablets and Internet hub to add.

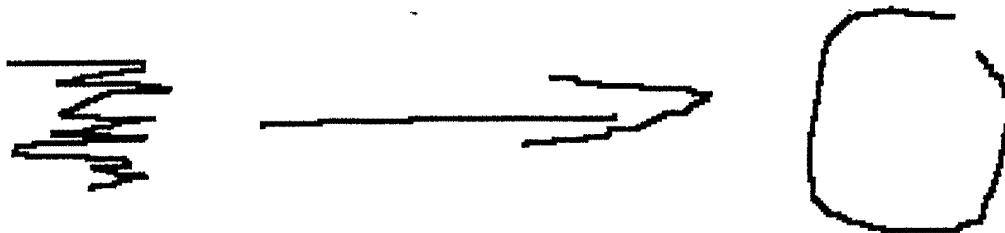
In Multimodal Maps, using OAA, we have some sharing/collaboration mechanisms.

Avery-Dennison "smart pen" idea. Pieces were there, but wasn't the technology too complex?

CIC provides good Handwriting Recognition engine. Already integrated in most projects.

STAR's Speech Recognition and Speaker ID expertise.

Gesture Recognition in use in all SRI's Multimodal projects.

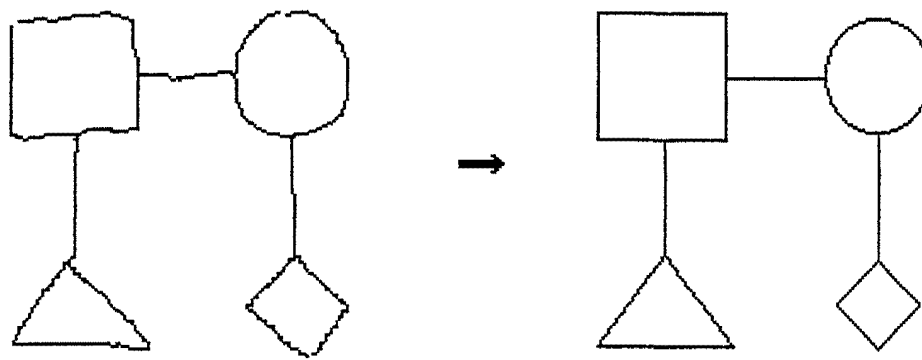
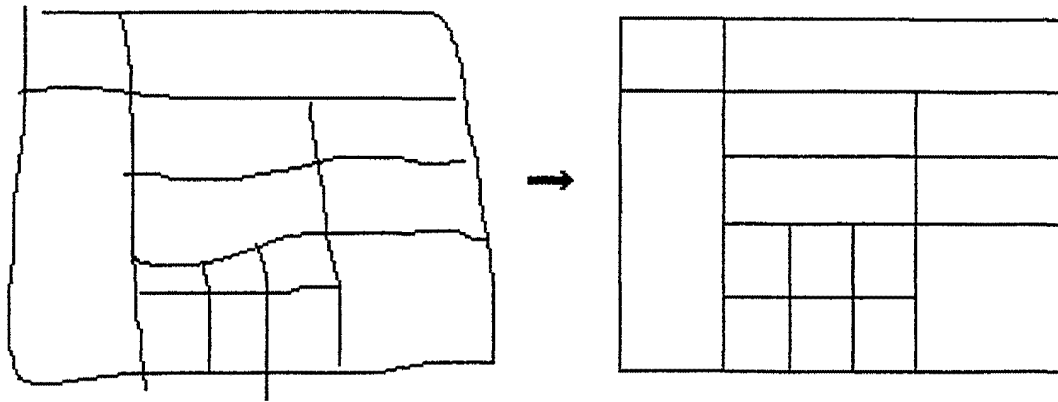


Multimodal fusion, resolving ambiguities, algorithms competition and complementary

TAPAGE and DERAPAGE: on the fly graphics recognition and semantic interpretation.

SRI INTERNATIONAL

CONFIDENTIAL



65750 "BTF02" 02

: ultimodal ccess and eneration for nteraction and ollaboration

CHIC!

Computer Human Interaction Crew

Send comments and suggestions to Dr. Luc JULIA Luc.Julia@sri.com

Copyright © 1998 SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025 USA. All rights reserved.

SRI INTERNATIONAL

INTELLECTUAL PROPERTY OFFICE

PROVISIONAL APPLICATION COVER SHEET

Alpro

This transmittal and the documents and/or fees itemized hereon and attached hereto have been deposited as "Express Mail Post Office to Addressee" in accordance with 37 C.F.R. §1.10 with Express Mail Mailing Label Number EL285395899US

Attorney Docket No.: SRIIP025+

First Named Inventor: CHEYER ET AL.

Assistant Commissioner for Patents
 Box Patent Application
 Washington, DC 20231

Duplicate for fee processing

jc541 U.S. PTO
 60/124719
 03/17/99

Sir: This is a request for filing a PROVISIONAL APPLICATION under 37 CFR 1.53(c).

INVENTOR(S)/APPLICANT(S)

LAST NAME	FIRST NAME	MIDDLE INITIAL	RESIDENCE (CITY AND EITHER STATE OR FOREIGN COUNTRY)
CHEYER	Adam	J.	Menlo Park, California
JULIA	Luc	E.	Menlo Park, California
GUZZONI	Didier	None	Menlo Park, California

TITLE OF INVENTION (280 characters max)

USING A COMMUNITY OF DISTRIBUTED ELECTRONIC AGENTS TO DYNAMICALLY MONITOR AND SUPORT THE NEGOTIATION OF ELECTRONIC TRANSACTIONS

CORRESPONDENCE ADDRESS

HICKMAN STEPHENS & COLEMAN, LLP
 P.O. Box 52037
 Palo Alto, CA 94303-0746
 (650) 470-7430

ENCLOSED APPLICATION PARTS (check all that apply)

___ Specification Number of Pages _____ Small Entity Statement
 ___ Drawing(s) Number of Sheets _____ Other: White Paper + Cover Sheet (20 Pages)


A check or money order is enclosed to cover the Provisional filing fees. Provisional Filing Fee Amount \$150

The commissioner is hereby authorized to charge any additional fees which may be required or credit any overpayment to Deposit Account No. 50-0384 (Order No. SRIIP025+).

The inventions made by an agency of the United States Government or under a contract with an agency of the United States Government.

_____ No _____ Yes, the name of the U.S. Government agency and the contract number are:

Respectfully Submitted,

SIGNATURE 

DATE 3/17/99

TYPED NAME Brian R. Coleman

REGISTRATION NO. 39,145

PROVISIONAL APPLICATION FILING ONLY

TITLE OF THE INVENTION

Using a Community of Distributed Electronic Agents to Dynamically Monitor and Support the Negotiation of Electronic Transactions

ABSTRACT

Complex transactions entail multiple steps over time, possibly non-linear path, with dynamic decisions at each point among multiple alternatives. For example, consider the process of buying a product available at different simultaneous auctions; or obtaining an online mortgage loan; or buying or renting real estate. Currently dominant paradigm of automated support for electronic transactions is typically linear/static: search for prospects, display list, and user is then on his/her own. But numerous alternatives and decisions are possible during the period between the placing of a bid/reservation/application and the closing of the transaction, and there is a real need for on-going monitoring of changing alternatives and for analytical decision support.

The present invention uses "OAA-style" collaborating distributed agents (facilitated collaboration among distributed agents with declared capabilities in a high-level interagent communication language) to monitor and support the negotiation of electronic transactions. One way to outline the work flow in such a system could be:

- define a purchase order profile for a desired product;
 - search multiple, heterogeneous database(s) for product availability
 - enter one or more "bids" (a formal offer or application to transact business)
 - automatically continue to periodically monitor databases for new information about
 - alternatives and status
 - analyze the new information with respect to triggers, notify user and present options
 - sometimes interactively decide to change/withdraw bid or enter new bid, based on the new informatio/triggers
 - ==> preferably this last step is fully automatic
- in some cases (bidding agents)

SUPPLEMENTAL INFORMATION

A detailed write-up of a preferred embodiment, in the context of online auctions, is attached.

Pending patent application serial no. _____ assigned to SRI (docket no. 3949-2) provides a detailed description of the underlying OAA platform architecture, and also specific descriptions of several applications including "multi-modal maps" which may be helpful in preferred embodiments. The referenced pending patent application is incorporated herein by reference in its entirety.

OFFICE OF THE ATTORNEY GENERAL

MetaAuction Alpha Preliminary Design

P-3970

SRI International

Overview

Online auctions are increasingly popular on the Internet – more than one hundred web sites exist simply to help buyers and sellers exchange goods through auction-based mechanisms. Although auction meta-sites are beginning to emerge (e.g. www.bidfind.com), these sites provide only search engine capabilities. Simply finding an auction site to participate in, although useful, is not enough: managing the auction process is a time-consuming effort. We believe that a more thorough automation of the auction-buying process is technically feasible, and could be provided as a mass-market service through portal such as the proposed MetaAuction.

SRI carried out an approximately 10 man-week background study and software prototyping effort to guide the MetaAuction design. The resulting system design, based on SRI's Open Agents Architecture™, will provide MetaAuction customers with the following services:

Product finding. Locate candidate products and their auction sites through familiar hierarchical and/or keyword searches. Provide links to manufacturer product descriptions, as well as pricing information, including average cost via auction and prices on fixed price sites for purposes of comparison.

Automated bidding. Execute multi-auction bidding strategy against user-selected products and sites. Automatically registers customers on auction sites and carries out bid placement.

Auction monitoring. Provide real-time reports on bid status and product availability. Monitors both specific items and auctions currently of interest to users, as well as global status of closing prices and other data of interest for all other items within *MetaAuction's* product categories.

Visual monitoring of auction status at the *MetaAuction* website.

- Phone/pager/e-mail notification of major change of status (e.g., when one successfully buys).
- Phone/pager/e-mail notification when items meeting interest criteria come available

SRI Confidential

SRI INTERNATIONAL

INTELLECTUAL PROPERTY OFFICE

562720"574709

Functional Requirements

User Interactions

Registration

1. UI for profile entry/review/editing.
 - name/password
 - credit card
 - ship-to, bill-to
 - notification means (e-mail, pager, phone)
2. Secure (SSL) communications to user browser.

Product Finder

1. UI for product location
 - Hierarchical selection of products
 - keyword search (require search engine capability)
2. Rapid lookup & presentation of product information
 - description
 - sku
 - current bid price
 - current best fixed-price
 - auction sites
 - links to manufacturer information sites.
3. Bid control
 - product/auctions to pursue
 - bidding behavior: start, max bid price; aggressiveness

Monitoring/Notification Service

Multi-modal notification: webpage, banner, e-mail, pager, phone (text-to-speech)

1. Change of status of existing auction actions (buy, lose)
2. Periodic status updates (current bid price, auction sites)
3. As new items become available [optional for *Alpha*]

SRI Confidential

SRI INTERNATIONAL

INTELLECTUAL PROPERTY OFFICE

Data Management Tasks**Product Database**

1. Information on preselected products and auction sites
2. Product query support
3. Periodic information updates via web crawler

Bid/monitoring management agents

1. Coordinate bids across multiple auction sites
2. Automatically schedule monitoring and bid updates

Decisions about when a bid should be considered based on user-definable "aggressiveness" parameters, time remaining until the auction closes, importance of a particular auction is based on a prediction of potential success, etc.

User profile database

Maintain secure database of user profiles with links to current auction status.

1. User name/password
2. Credit card
3. Ship-to, Bill-to, account status
4. Historic (audit trail)
5. [user preferences, personalized homepage layout ..]

Information Extractors

1. Site-specific information extraction in support of bidding, monitoring and crawling (product database)
2. Multi-threaded (multiple simultaneous access)
3. Page and information caching

Other Design Issues

1. Browser compatibility (standards: HTML & httpd rev, use of Java applets and JavaScript vs. server-side HTML generation, etc.)
2. Modem/network load (bandwidth)
3. MetaAuction server scaling

SRI Confidential

SRI INTERNATIONAL

INTELLECTUAL PROPERTY OFFICE

- 500,000-1,000,000 registered users
 - 50,000-100,000 simultaneous bidders
 - 10,000-100,000 monitored products
4. MetaAuction security
 - Customer (browser-server link)
 - Site security (site firewall, etc)
 5. Extensibility for future services
 - New sites, products (mods probably on weekly basis)
 - Adaptations to new auction software (direct auction database access, XML, etc.)
 - Speech interface
 - Phone notification (text to speech)
 - Phone-based monitoring, eventually bidding (speech recognition, NLU)
 - Physical telephony requirements (T1 lines vs. Digital terminals, etc.)
 - Advanced bidding strategies
 - Optimized for single bidder
 - Optimized for blocks of bidders
 - Natural language understanding
 - Product information parsing
 - User description of product
 - Other.
 6. Flexibility to support new processing architectures (e.g., highly distributed with significant client-side processing)

Proposed Alpha System Design

Goal: Rapid, low-risk / low-cost, rapid development of an operational MetaAuction site. Maximize the code carryover to *beta* version. Provide full suite of features for evaluation and demonstration.

Strategy: Make extensive use of OAA rapid-prototyping capability. Focus on new functionality for MetaAuction application, minimize system integration effort.

Scalability: Implement web page and site information caching to enhance performance for large-scale deployments. Modules compatible with later (*beta*) integration under OAA, CORBA, or blend where appropriate.

Security: Alpha architecture firewall-ready, relies on Proxy, SSL communications for sensitive information.

SECRET 09/24/09

Theory of Operation

The user interactions in the proposed system design parallels the stages of the general buying process, prefaced with user re-entry or registration:

MetaAuction Site Entry. MetaAuction users will be greeted by a main page with MetaAuction branding, advertising, and either a login/password, or registration request for new users (including entry as a limited Guest). The opening page may also provide the product finder interface, major status information for that user (active bids, etc.), and links to help pages. Current users will be able to step into the main bidding channels; new users will be required to input name/password, credit card, ship-to/bill-to, and notification information (e-mail address, pager number, etc.).

Product Search. Locate products based on key-word search, hierarchical product definitions or combination thereof (similar to Junglee or Jango). MetaAuction will carry out a search of its *local* product/auction database and respond with a scrolling list of items meeting user objectives. List will include short product description, auction site, current bid prices, and links to vendor information. For reference, prices on similar items at fixed-cost sites can also be included (in this way, MetaAuction could essentially supersede Jango and Junglee). The system will allow the user to refine their search while viewing the current results in a fashion similar to the commercial search engines.

Bid launch. A user establishes what amounts to a Purchase Order (PO) for a product. The PO specifies the list of acceptable products (that is, what the user considers interchangeable) and auction sites. Parameters for aggressiveness, max bid price, and bidding end date will also be established.

Auction monitoring. The auction monitor provides status information on current active bids for this user, showing the site where recent bids have been placed, the corresponding bid amounts, and times. This view will also allow cancellation of any active PO (within limitations of auction site rules, and the status of that user's outstanding bids).

A more thorough Use Case analysis for these tasks will be carried out during the detailed design phase of the Alpha development program.

System Design Overview

Design Overview

The system design follows the client/server model standard for web applications. The client-side user interfaces provides monitoring and control over a server side that carries out the product/auction finding, extraction, and bidding services of the system.

The Open Agent Architecture™ (OAA™) was selected as the integration framework for MetaAuction. OAA provides two major features relevant to intelligent, web-based applications. First, its use of independent, cooperating agents and an open interface enables new capabilities to be inserted with minimum effort, and without system reconfiguration or downtime. This feature will allow MetaAuction to add support for new auction sites, new bidding strategies, and new user interfaces as a matter of course.

SRI Confidential

SRI INTERNATIONAL

INTELLECTUAL PROPERTY OFFICE

Second, OAA is based on a collaborative problem-solving scheme, and has great long-term promise as more complex auction negotiations and user interactions are required. Finally, OAA also brings a wealth of existing tools (agents) to provide important user services, including multi-modal input, output (text-to-speech, paging, etc), natural language understanding, along with sophisticated planning/reasoning tools.

In short, OAA provides short-term development cost reductions, and in the longer term, the opportunity for more sophisticated capabilities to be quickly integrated as necessary.

General Architecture

The MetaAuction design, sketched in Figure 1, is based on the OAA™ model for collaborative problem solving. In this context the problem is locating and buying one or more user-selected item from a (potentially vast) list of suppliers. To achieve these ends, the system must elicit information from the user, maintain a database of information on current auction sites and their products, and place and monitor bids according to some buying strategy.

Two basic layouts are possible within the OAA framework. The first, and probably most practical for near-term application, is a coarse-grained architecture. In this design, each agent is a specialist in some aspect of the product monitoring and buying process, and simultaneously manages all purchase order requests. Multi-threaded processing allows each agent to efficiently manage thousands of POs in their various stages. If additional parallelism is required, multiple identical agents can be operated across processors on a network. This parallelism can be achieved at both a single site, or from multiple points of presence on the internet.

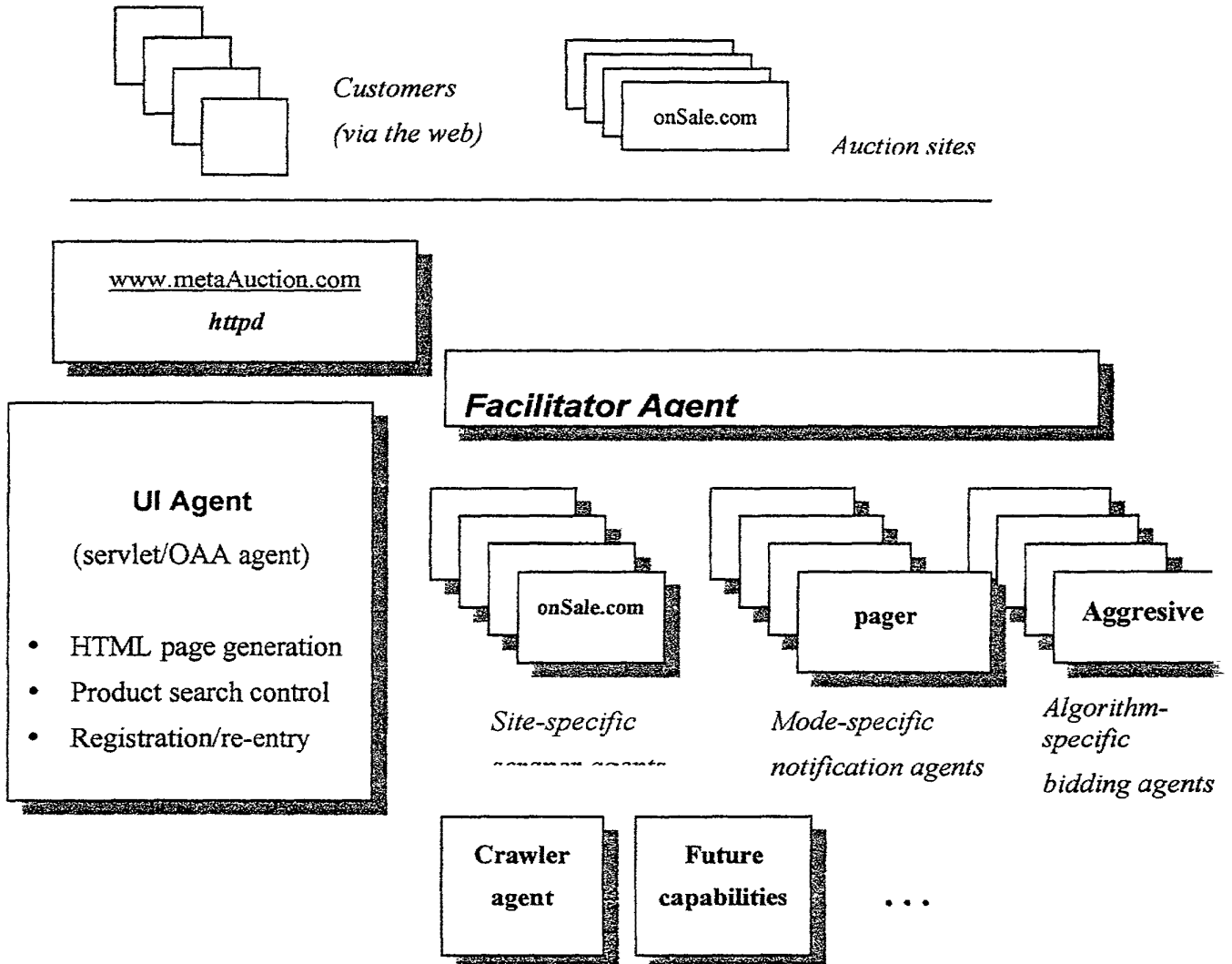
An alternative design approach achieves parallelism at the PO level. In this realization an independent family of cooperating agents is instantiated for each bidding request. The functionality of the agents would be identical to that in the coarse design, with the exception that each agent would be somewhat lighter as it need only manage a single buying thread. The primary advantage of this design, beyond general elegance, is its extensibility to highly distributed processing. For example, this would directly support client-side auction processing. This feature may one day be important as consumers are afforded full-time connectivity to the internet.

SECRET

SRI Confidential

SRI INTERNATIONAL

INTELLECTUAL PROPERTY OFFICE



SECRETED SRI 2003

Figure 1: A coarse grained agent-based MetaAuction architecture.

Agents

Page Scrapers. Implements parsing and text submission required for interaction with specific auction sites. (1 per site)

Auction Crawler control. Creates the product information database. Maintains a list of auction sites that are visited periodically and searched for products of interest.

UI. Controls access and interaction with users. Implemented as both an OAA agent and Java Servlet, thus coupling *httpd* facilities directly within OAA framework.

Bid strategy. Implements bidding process. Expected to be a rather fluid module with frequent updates and augmented with adjunct agents as strategies are refined. (1 per product category and/or strategy)

Product/auction search. Controls searches of Web and local database to satisfy user requests for product availability information.

Notification. Automatically notifies metaAuction users of major changes in their bids or product availability, including successful purchases and newly available items. (1 per modality)

Major Data Stores

User database. User ID/passwords, bill-to/ship-to, account status, list of active bids.

Product database. Store of product and corresponding auction site information. Maintained for both the items under bid, as well as speculative requests for products of general interest posted by the crawler agent.

Both databases could be implemented by conventional commercial RDBMS products. Regardless of the parallel-processing architecture, the agents will place heavy demand on a central database of product information, user profiles, and cached web pages. Although we can take advantage of parallelism in data processing, a transaction processing bottleneck will still exist. Careful attention should be given to the choice of database product, the manner in which it is accessed, and the specific data models used. Maintaining speedy database access for common transactions and cached pages at the MetaAuction server site will reduce the apparent latency to the user and compensate for delays in accessing remote auction sites. This problem is not unique to the meta-auction problem, and a variety of commercial products and widely accepted approaches exist for tackling it. However, the success or failure of the final design will hinge on how well it is handled. Anticipating this fundamental problem in the design phase, and applying best practices to its solution, will smooth the way for scalability as the system grows.

Scaling Issues

Communications between extractors and their web sites. To address this problem we propose a two-level caching scheme that minimizes downloads from auction websites, as well as processing for 'scraping' (Figure 2). In this design, information gleaned from pages is time-stamped and stored in an intermediate database. A second level of cache provides rapid access to entire (time-stamped) web pages. While this strategy will provide limited benefit to light user loads, it should recoup significant gain as hundreds or thousands of simultaneous users review or bid on similar products.

OAA scalability. Our initial design will make full use of the OAA fast prototyping capabilities. After the *alpha* design gels, however, we may pursue performance gains by combining smaller agents and other streamlining. OAA will still provide the system framework for adding new functionality and carrying out more advanced problem-solving tasks.

SRI Confidential

SRI INTERNATIONAL

INTELLECTUAL PROPERTY OFFICE

Security Issues

Site Security. We recommend that a firewall stand between the MetaAuction system and the internet. The proxy web-server, used in conjunction with the firewall, will also improve site security.

MetaAuction-auction site and browser-MetaAuction communications. The Alpha system will rely on SSL to secure user information and bid transactions. SSL support is currently required by most auction sites, and is standard on all widely used commercial browsers. MetaAuction would be registered with VeriSign® or other verification/authentication services.

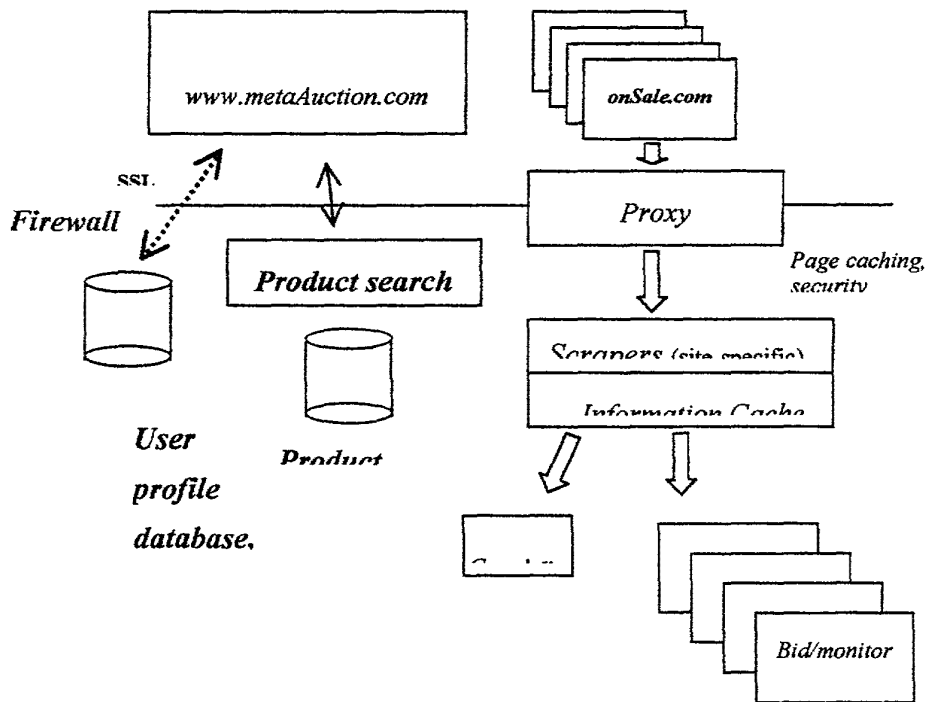


Figure 2: Major data storage components and data flow.

SECRET

APPENDIX: A MetaAuction PROTOTYPE

Adam Cheyer, Luc Julia, Didier Guzzoni

To better understand the requirements, feasibility and design alternatives for MetaAuction, SRI implemented an experimental prototype known as FAAAB (for "Find All Auctions And Bid"). FAAAB was developed within the Open Agents Architecture™ (OAA™) and implements key elements for Product Finding and Auction Monitoring, and their corresponding user interface. This prototype demonstrated the general feasibility of automatically locating available products, as well as monitoring the bidding process. The FAAAB prototype also provided important experience on user interface issues, and provides the basis for the current UI design.

Problem Statement

The FAAAB prototype effort investigated the feasibility of applying agent technology to an electronic auction domain. Automated software agents would be responsible for providing the following capabilities to users of the service:

1. Find online auctions selling products the user is interested in.
2. Find the best market price available from commercial vendors for these products – this gives the user a baseline value against which to compare the value of an auctioned item.
3. Monitor relevant auction sites, acquiring knowledge about the patterns of buying and selling at that site (e.g., how often does a user truly find a bargain at a particular site).
4. Manage a collection of automated bidding agents who cooperate to achieve the user's objectives in obtaining products at a good price.

The ideas outlined here have been implemented in the FAAAB prototype system, which demonstrates (and allows experimentation with) many of the facets required for accomplishing the vision.

Requirements

To implement auction finding and bidding service, at least the following major components are required:

- web crawlers and information extractors to retrieve information about and from auction sites;
- a user interface to enable users to task and control monitoring and bidding agents;
- and a sophisticated agent scheduler to efficiently coordinate the efforts across auction agents.

SRI Confidential

SRI INTERNATIONAL

INTELLECTUAL PROPERTY OFFICE

Crawlers and Extractors

There are at least two types of web data involved in our auction process: data which is highly dynamic in nature (e.g., an ongoing auction changes frequently as new bidders take action), and data which is more stable (e.g., the structure of a given auction site, or the set of auction sites themselves). The more static data can be efficiently cached by web-crawlers which refresh the cache every day or two, while the more dynamic data must be retrieved on demand from the source web page (by extractors). Crawlers generally have an associated database which caches their findings to allow rapid access.

Except for when the information to be extracted is generic in nature, such as a URL or email finder, or a keyword-based search index, knowledge will need to be generated about the format of the information to extract from a particular web site. Since most of the development time associated with this effort is related to encoding this knowledge, having the right tools and languages with which to do so is essential.

Domain-specific knowledge encodes two types of information : how to *navigate* web sites (e.g., go to URL X, find a button labeled Y and click on it, and then fill out a form at the resulting page with the specified information); and how to *extract* information found at the site. Encoding languages should be able to represent both sorts of knowledge in as readable and concise a form as possible.

It is desirable that the tools that interpret or execute the domain-specific knowledge have the following properties:

- Multi-threaded (or multi-tasking) to be able to manage many knowledge-extraction requests simultaneously
- Replicate-able and/or mobile, so that new instances can be created and distributed according to load requirements
- Able to communicate with other components of the system, such as databases for caching information, user interfaces, etc.

User Interface

The User Interface (UI) to the target system should have the following properties:

- Be portable and accessible from any modern web browser.
- Be rich enough to visually express a complex space of information: many agents will bid and monitor at auctions with changing prices, varied closing dates, etc. A user should have a global understanding of the current status of an entire multi-agent auction portfolio, and the ability to modify or control any aspect of the process.
- Intermittent operation: a user should be able to disconnect and reconnect at will.
- Lightweight: Additional low-profile UIs (e.g., banners) can update the user of portfolio events without requiring connection to the full user interface or focused attention by the user.

SRI Confidential

SRI INTERNATIONAL

INTELLECTUAL PROPERTY OFFICE

Agent Scheduler

An agent-scheduler must be imbued with the ability to efficiently manage and schedule information retrieval tasks for the auction monitoring and bidding agents. The scheduler will make decisions about when a bid should be considered based on user-definable "aggressiveness" parameters, the amount of time remaining until the auction closes, how important a particular auction is based on a prediction of potential success, etc.

Implementation

The ideas and requirements have, for the most part, been implemented in a prototype system called FAAAB (for "Find All Auctions And Bid"). Features and issues not yet accomplished by this prototype are discussed in the next section.

Integration Framework

From our requirements section, it is clear that the implemented system must use a client/server model, with client user interfaces providing monitoring and control of a server side that provides the finding, extraction, and bidding services of the system.

The Open Agent Architecture™ (OAA™) was selected as the integration framework for FAAAB, as the OAA enables rapid development of both Java-based client user interfaces and complex server applications made out of distributed components.

Crawlers and Extractors

In the requirements section, we spoke of the need for both tools and languages for expressing domain-specific extraction and navigational logic. After evaluating several in-house (DIFF-parse, DCG-parse, plus web agents) and commercial (AgentSoft's LiveAgent Pro [2]), we chose Digital's WebL product [3] as the best tool and language for our needs. Implemented in Java, WebL provides powerful features (parallelism concepts, markup algebra combining query sets over regular expressions and structured HTML and XML representations, specialized web-related exception handling, and so forth). In addition, source code is provided for free, allowing us to easily incorporate the technology as an OAA agent, and to make extensions to it as necessary (e.g., add mobility).

The WebLOAA agent provides a generic OAA-enabled tool which can dynamically load knowledge scripts encapsulating a particular web site or service; each script becomes an agent in the OAA sense. Scripts can serve both as extractors, and when used in

SRI Confidential

SRI INTERNATIONAL

INTELLECTUAL PROPERTY OFFICE

conjunction with an OAA database agent, crawlers, which cache their results for fast retrieval.

Auction Finders

The first task of the FAAAB prototype requests a user to input a description of a product that they are interested in, and then attempts to find auctions which are selling comparable products. This task was accomplished in two ways:

1. an extractor agent for an existing auction search site, BidFind.com [4];
2. a web crawler for a site not currently indexed by BidFind (www.webauction.com [5]) to demonstrate that we need not be reliant on the BidFind service.

Both the WebAuction agent and the BidFind agent were rapidly implemented as WebL scripts managed by the WebLOAA agent. See Appendix A for source code of the BidFind extractor agent, to get a sense of the power and elegance of the WebL language for web wrapping and extraction tasks.

Market Price Finder

Once a list of interesting auctions have been returned and displayed to the user, he or she should choose which sites are to be managed by FAAAB auction agents. For each auction returned, the user may visit the website or may request a search for the real market price of the auctioned object. Note that even though most auctions returned for a given search will offer relatively similar products, the products may have varying brands, optional features, and so forth, so it might be desirable to find the market price for each individual auction and not just for the group.

Even though product search engines are starting to appear (e.g., Jango [6], Junglee [7]), finding a good guess for the real market price of an object given only its description is not an easy task. Here is the approach that we are using for the moment:

Given a description of an object for sale at an auction, we first try to guess the major category (e.g., desktop computer, camera, flowers, etc.) for the product. Jango, the best product finder currently available, uses a yahoo-like hierarchical category scheme, with pulldown menus for different choices. For instance, if looking for a laptop computer, you choose this category and then select criteria such as brand, model and processor speed from a preselected list. These criteria, both headings and values, are different for each category.

To guess the product category, we wrote a WebLOAA crawler that traverses all of the categories from Jango and pulls out the criteria and values for each category. Then for the given object description, we choose its category by taking the one which has the highest number of values present in the object description. The values are augmented by a hand-coded synonym list to increase the likelihood of positive matches. Note: a future enhancement would be to automatically generate pertinent keywords from the corpus of category items using statistical methods.

SRI Confidential

SRI INTERNATIONAL

INTELLECTUAL PROPERTY OFFICE

Once the major category has been determined, the findMarketPrice agent tries to fill out the search form for that category with relevant criteria taken from the target description. Resulting descriptions are then compared against the target description for similarity, and the price, description and URL of the best guess are returned to the user interface for display to the user. Note: this step is still under development, and in the meantime, a simple price-by-category result is returned as the answer.

Agent Scheduler

The agent scheduler, implemented in Prolog, is responsible for efficiently managing update requests for an entire community of auction bidding and monitoring agents and for webcrawler agents. As auction agents are created or modified, the agent scheduler plans future checkup times for the site based on:

- Closing date: scheduled checkup times are proportional to the amount of time remaining until the auction closes. If the auction closes in a week from now, it doesn't make sense to check the auction page every minute. However as the deadline approaches, more frequent checks are necessary.
- Auction importance: some auctions are more desirable than others for a variety of reasons. For instance, if one site has 500 copies of a product to sell and another site only has one, placing a winning bid at the first site is much more interesting because 500 other users will need to bid higher before your bid is surpassed.
- Users might also indicate a preference for a particular auction object over another.
- Agressiveness parameter: a user can tailor an aggressiveness parameter which influences how often an auction agent bids.
- Real-world notifications: some auctions send an email when someone has outbid you, and an email agent could reschedule an immediate counter-bid (not yet implemented).

User Interface

The user interface design reflects three key phases in the auction buying process: product location, bid selection, and monitoring. In the setup phase of the FAAAB prototype (Figure 1), users find and evaluate potential auctions of interest, and then create auction agents to monitor and bid on these auctions. The Setup tab of the user interface retrieves auction information from the auction site crawlers. Users may then choose to view the original web page featuring the auction or to search for the best online market price for the product offered by that specific auction. Note that multiple "tabs" can be created, each representing a group of agents (currently limited to 10 per group) acting upon auctions in a given "domain" (e.g., Pentium computers, cameras, sunglasses, etc.)

SRI Confidential

SRI INTERNATIONAL

INTELLECTUAL PROPERTY OFFICE

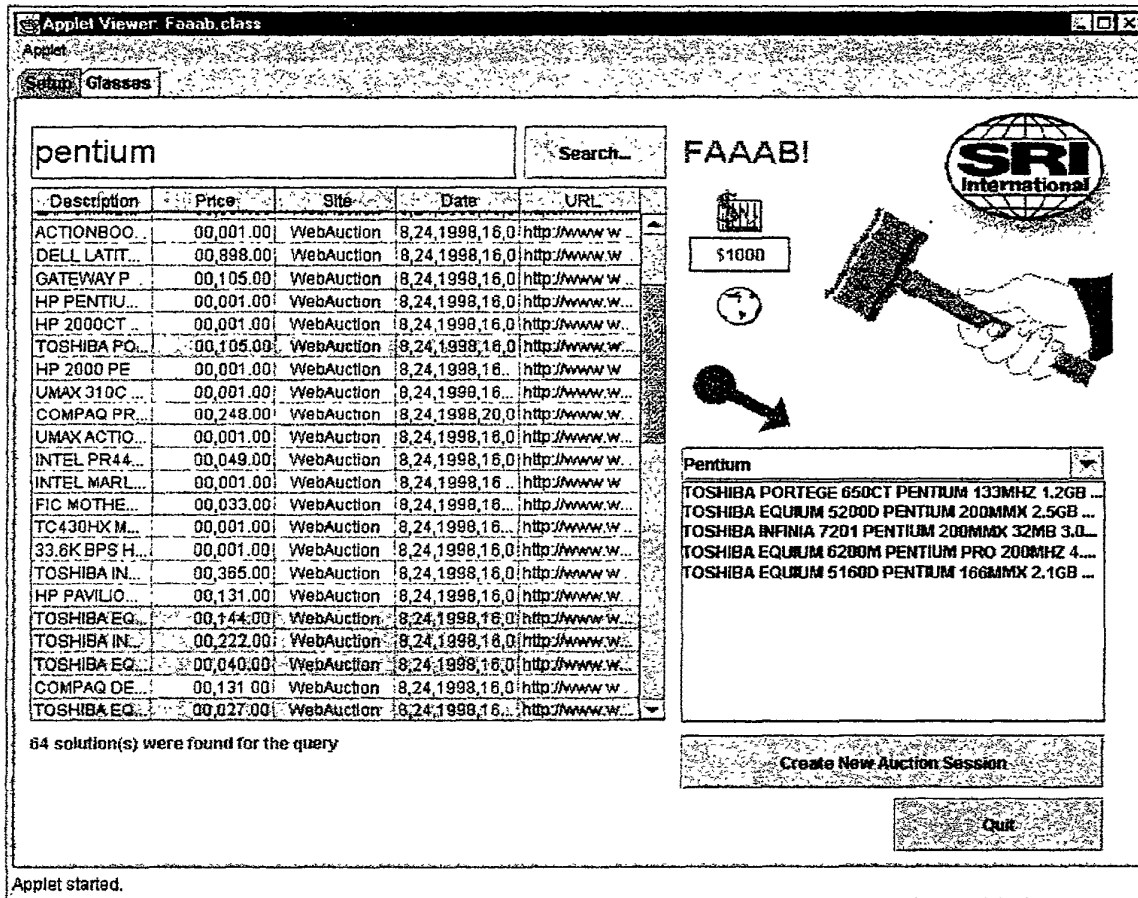


Figure 1. Creating auction agents for "Pentium" computers

For each domain tab created, a user can graphically view progress and results of auction agents (Figure 2). Agents are classified either as monitor agents, who simply record progress of a particular auction, or bidding agents, who autonomously make bids according to user instructions. The market price (actually, the highest market price for all auction products) and the (highest) max bid are displayed on a gauge. As the auctions unfold, the agents graphically move up the meter, displaying their current prices. Agents who have surpassed their max bid are colored with a red background.

An agent editor enables the user to tailor various properties of the auction agent, such as max bid and aggressiveness. Additional information is also available, such as the bidding history to the current moment, market price for the product, etc.

SRI INTERNATIONAL

SRI Confidential

INTELLECTUAL PROPERTY OFFICE

BOULEVARD "ST. ALBERT"

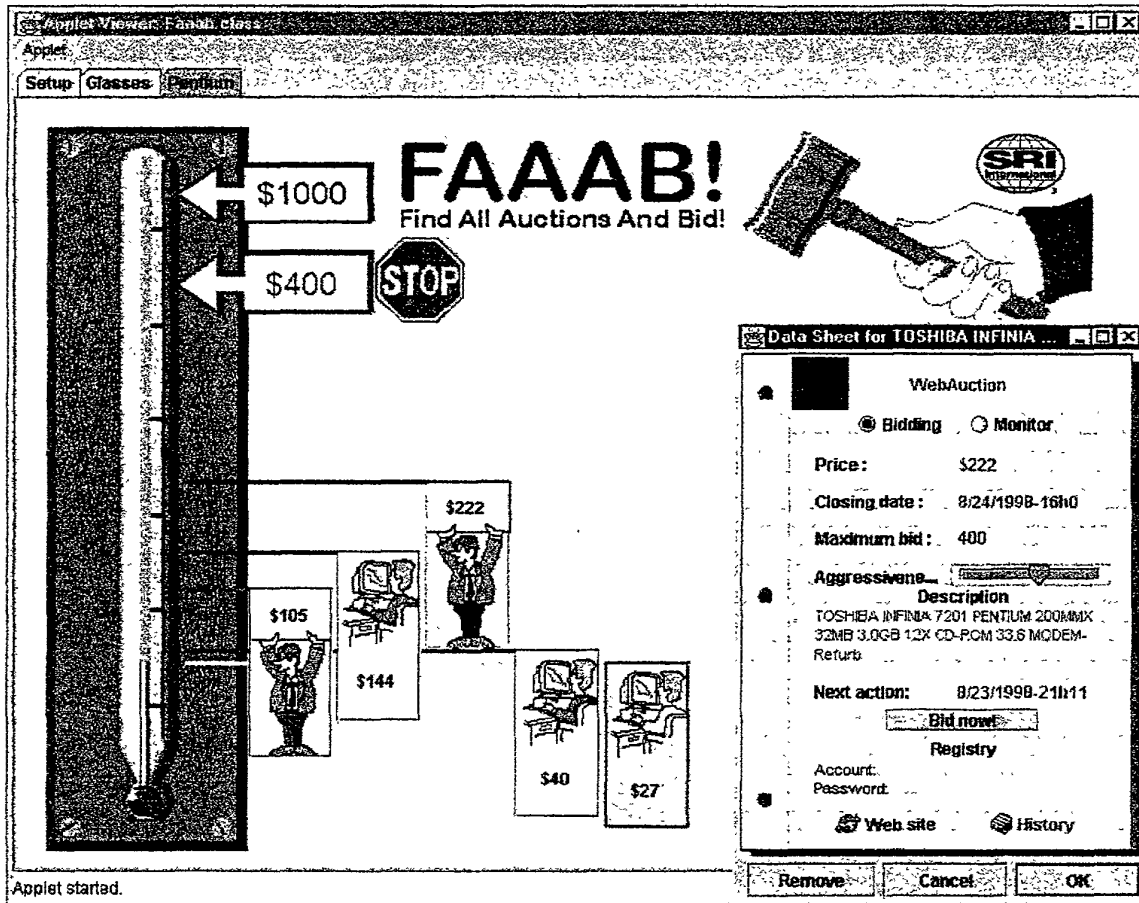


Figure 2. Bidding and monitoring agents for "pentium" auctions

Alternate interfaces are also possible. Figure 3 displays a lightweight "banner" interface which unobtrusively keeps the user informed as to updates by his or her auction agents.

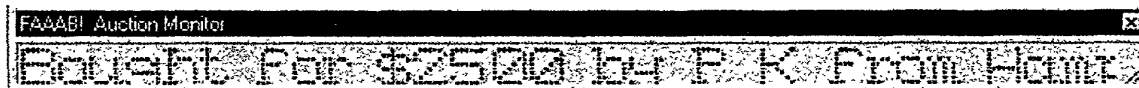


Figure 3. A lightweight banner interface displaying updates

664760 674209

Prototype Architecture

Figure 3 illustrates the architectural layout of the FAAAB components within the OAA. This section details a few notes on information flow and data storage choices implemented in the prototype.

FAAAB!

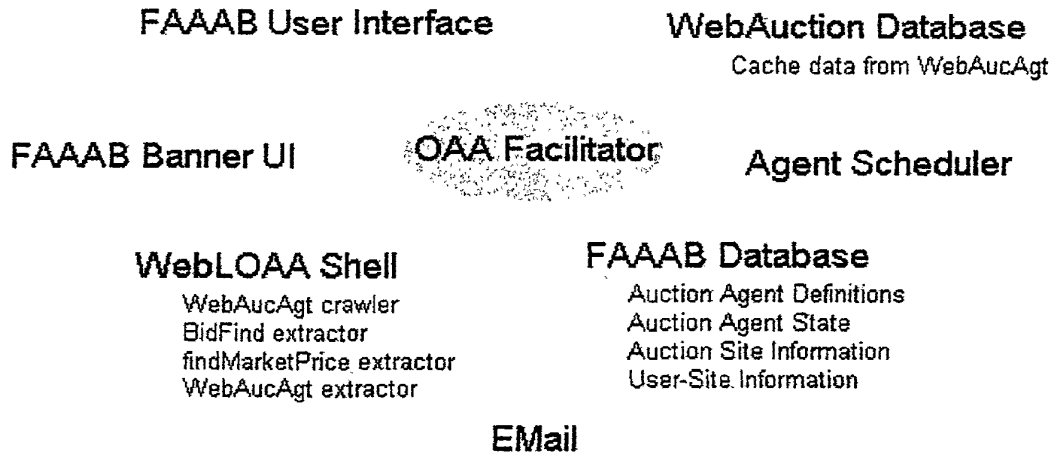


Figure 3. Architecture of FAAAB Prototype

The main FAAAB user interface is accessed from a web browser. According to the FAAAB operational concept, a user will begin by issuing searches for auctions selling interesting products. The results of these searches come from cached data stored in the WebAuction database, and recalculated-on-demand data retrieved by the BidFind extractor agent. WebCrawler caches updates are managed by the Agent Scheduler.

For a given auction found by the above process, the user can query market price information about its products using the findMarketPrice extractor, and view full information about the auction using the UI browser.

A user then selects a subset of auctions to monitor using FAAAB agents -- information about each auction agent is stored in the FAAAB database. A user can edit and tailor agent specific information using the editor provided by the UI.

The Agent Scheduler is notified through OAA's trigger mechanisms of new or modified auction agent definitions. For each auction agent, the scheduler generates a monitoring plan based on auction closing date, importance of the auction site, user-tailorable aggressiveness parameters, etc.

60124719-03109
60270" 6742109

When an auction agent “checkup” time arrives, the scheduler sends a request for an extractor to read the auction site and retrieve all information about it. If a bid should be made according to the optimal bidding strategies for the user’s portfolio, a request is made to available bidding wrapper agents. The results of monitoring and bidding are written to the AucAgtState predicates in the FAAAB database.

The FAAAB user interface and banner user interface both receive update notifications about change in agent state, and display the results accordingly.

An email agent can be used for sending final reports about history and results when an auction closes, and for detecting real-world notifications that another user has outbid you.

Note: the system is extensible and can operate in disconnected mode. As new bidding and monitoring extractor and crawler agents are dynamically added to the system, they will automatically be integrated into the FAAAB process. Disconnected operation is available because both the user interface agents and agent scheduler store all state information in persistent databases and reload this information upon connection at a later time.

Conclusions

The FAAAB prototype illustrates that the construction of an automated meta-auction site management service is a feasible endeavor. The key contributions of the effort are:

- Design and implementation of multiple user interfaces that enable ubiquitous, disconnected access and control to the auction agents.
- Design and implementation of bidding and monitoring strategies and scheduling.
- Integration within a flexible architecture that facilitates light client UIs and complex, distributed, extensible server implementations.
- Selection of a representation language (WebL) for encoding navigational and extraction knowledge for web sites.

EXHIBIT 10-014709

SRI Confidential

SRI INTERNATIONAL

INTELLECTUAL PROPERTY OFFICE

Related Work & Resources

1. MIT Media Lab's KASBAH experiment: multi-agent implementation of a commercial marketplace, where both buyers and sellers are represented by agents. What we can learn: parameters and algorithms for automated buying agents.
<http://ecommerce.media.mit.edu/Kasbah/>
2. AgentSoft's LiveAgent Pro: A scripting language for automating the web. Semi-automatic generation of scripts through construction through example. Cumbersome to use...
<http://www.agentsoft.com/>
3. Digital/Compaq's WebL language: A scripting language implemented in Java which contains powerful "Markup Algebra" and exception handling features. Free!
<http://www.research.digital.com/SRC/WebL>
4. BidFinder: An auction meta-site search engine, allowing users to find current auctions for products (from keywords).
<http://www.bidfinder.com/>
5. WebAuction: One of the most popular and large online auction sites.
<http://www.webauction.com/>
6. Jango: Bought by Excite, the premier product finder on the web. Spinoff from University of Washington (Etzioni & Weld).
<http://www.jango.com/>
7. Junglee: Similar to Jango but currently only for resume selling/buying.
<http://www.junglee.com/>

SRI Confidential

SRI INTERNATIONAL

INTELLECTUAL PROPERTY OFFICE

EXCERPT FROM

66/11/20



010 U.S. PTO

PROVISIONAL APPLICATION COVER SHEET

A/Prov

This transmittal and the documents and/or fees itemized hereon and attached hereto have been deposited as "Express Mail Post Office to Addressee" in accordance with 37 C.F.R. §1.10 with Express Mail Mailing Label Number EL285395885US

Attorney Docket No.: SRIIP024+

First Named Inventor: CHEYER, Adam J.

Assistant Commissioner for Patents
Box Patent Application
Washington, DC 20231

[] Duplicate for fee processing



Sir: This is a request for filing a PROVISIONAL APPLICATION under 37 CFR 1.53(c).

INVENTOR(S)/APPLICANT(S)

Table with 4 columns: LAST NAME, FIRST NAME, MIDDLE INITIAL, RESIDENCE (CITY AND EITHER STATE OR FOREIGN COUNTRY). Row 1: CHEYER, Adam, J., Menlo Park, CA

TITLE OF INVENTION (280 characters max)

AN "INVISIBLE" USER INTERFACE PROVIDING A HIGH DEGREE OF INTEGRATION ACROSS MULTIPLE APPLICATIONS IN A PERSONAL COMPUTER ENVIRONMENT

CORRESPONDENCE ADDRESS

HICKMAN STEPHENS & COLEMAN, LLP
P.O. Box 52037
Palo Alto, CA 94303-0746
(650) 470-7430

ENCLOSED APPLICATION PARTS (check all that apply)

[] Specification Number of Pages [] Small Entity Statement
[] Drawing(s) Number of Sheets [X] Other: Title, Abstract and Supp. Info. (8 Pages)

[X] A check or money order is enclosed to cover the Provisional filing fees. Provisional Filing Fee Amount \$150
[X] The commissioner is hereby authorized to charge any additional fees which may be required or credit any overpayment to Deposit Account No. 50-0384 (Order No. SRIIP024+).

The inventions made by an agency of the United States Government or under a contract with an agency of the United States Government.
[] No [] Yes, the name of the U.S. Government agency and the contract number are:

Respectfully Submitted,

SIGNATURE [Signature]

DATE 3/17/99

TYPED NAME Brian R. Coleman

REGISTRATION NO. 39,145

PROVISIONAL APPLICATION FILING ONLY

TITLE OF THE INVENTION

An "Invisible" User Interface Providing a High Degree of Integration Across Multiple Applications in a Personal Computer Environment

ABSTRACT

Let's say a user is doing some interactive document authoring, and wants to insert some information that exists somewhere in electronic form but is currently external to the document. User wants this to be a highly integrated process, i.e., doesn't want to shift attention away from the working document and temporarily adjust to a different interaction mode, and shouldn't have to waste time on menial, repetitive steps simply to move information around that is already inside the computer.

In accordance with the present invention, an "OAA-style" architecture (facilitated collaboration among distributed agents with declared capabilities in a high-level interagent communication language) can be used to seamlessly ("invisibly") integrate the document authoring application and other auxiliary applications, such as information gathering applications. For example, here is an outline for one possible interactive scenario:

- 1) User is running an interactive document authoring application
- 2) User signals for OAA attention (e.g., function key, like invoking a "help" coach)
- 3) User requests insertion of desired information, e.g. (using natural language) "Insert the directory listings for Adam Cheyer").
- 4) Request is parsed as needed by NL agents.
- 5) Appropriate auxiliary agent(s) is(are) selected, and the Request is automatically dispatched in appropriate form to the selected agent(s) (e.g., local address book, Internet search engine, local file manager)
- 6) Designated agent(s) retrieves/processes the desired data.
- 7) The retrieved data is presented to the document authoring application and inserted into the working document.

Authoring applications featuring "invisible"-style integration of selected auxiliary applications -- that are pre-selected and "hardwired" for anticipated patterns of likely use -- are becoming quite popular, e.g., spelling and grammar checkers, more specialized citation checkers, and even the merger/integration of Web/desktop file management. But OAA-style architecture provides great potential benefit over such alternatives, in part because it is not a hardwired architecture. With OAA-style architecture, many different possible auxiliary application-agents can each declare their capabilities, and user requests are intelligently processed and delegated dynamically -- so that new agents can be plugged in over time. Such a structure, in accordance with the present invention, can effectively provide the primary UI to an entire personal computing environment rather than just for a single specialized application. In addition, with the likely future growth of speech-driven user interfaces, the value added by a UI-integration technology based on OAA-style architecture in accordance with the present invention is even more pronounced.

```
unit main;
```

```
(*****
* Unit: Main
* -----
* Purpose: Program body for "KeyCap Agent", which provides an "invisible"
* or "ubiquitous" user interface to a community of OAA agents.
* From any Windows program, a natural language query (e.g. "phone
* number of bill smith's manager") can be typed and copied to the
* Windows clipboard. When a key (e.g. F12) is hit, this agent takes
* the English request, requests translation and execution from the
* OAA community, and then posts the result back on the clipboard with
* an audial confirmation ("ding"). If the result couldn't be
executed,
* the user hears a failure sound ("bong"). In this way, requests
* involving many agents and programs can be accessed transparently
* from ANY Windows application at any time, without having to call up
* separate user interfaces for each of the involved apps.
* Authors: Adam Cheyer
* Version: 1.0
* Copyright 1998 by SRI International, all rights reserved.
*****)
```

```
(* ===== *)
(* Interface: exported declarations *)
(* ===== *)
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
Menus, OleCtrls, isp3, OAAAgent, ComCtrls, FngKbdNtfy, StdCtrls, MPlayer;
```

```
type
```

```
TfrmMain = class(TForm)
  MainMenu1: TMainMenu;
  oaa: TAgent;
  tcp: TTCP;
  mnuFile: TMenuItem;
  mnuExit: TMenuItem;
  mnuConnect: TMenuItem;
  Status: TStatusBar;
  keyCap: TFnugryKeyboardNotify;
  txtTest: TEdit;
  MediaPlayer1: TMediaPlayer;
  Label1: TLabel;
  procedure mnuConnectClick(Sender: TObject);
  procedure tcpClose(Sender: TObject);
  procedure tcpDataArrival(Sender: TObject; bytesTotal: Integer);
  procedure tcpConnect(Sender: TObject);
  procedure oaaSendData(Sender: TObject; data: String);
  procedure mnuExitClick(Sender: TObject);
  function oaaOAAEvent(Sender: TObject; ks, func, args: PChar;
    var ans: PChar): Boolean;
  procedure keyCapKeyboardMessage(Sender: TObject; Key, lParam: Integer;
    var fDiscard: Boolean);
  procedure FormCreate(Sender: TObject);
  procedure Play(fname: String);
  procedure Button1Click(Sender: TObject);
```

```

private
  { Private declarations }
  function ResolveArg(s: PChar) : PChar;
public
  busy: Boolean;
  { Public declarations }
  procedure ReadCmdLine;
end;

var
  frmMain: TfrmMain;

implementation

uses connect, clipbrd;

{$R *.DFM}

procedure TfrmMain.ReadCmdLine;
var
  i: Integer;
  StartOAA: Boolean;
  host: String;
  port: integer;
begin
  i := 1;
  StartOAA := False;
  oaa.ReadSetupFile(host, port);
  tcp.RemoteHost := host;
  tcp.RemotePort := port;
  While i <= ParamCount do begin
    if ParamStr(I) = '-oaa_host' then begin
      i := i + 1;
      StartOAA := True;
      tcp.RemoteHost := ParamStr(I)
    end else
      if ParamStr(I) = '-oaa_port' then begin
        I := I + 1;
        StartOAA := True;
        tcp.RemotePort := StrToInt(ParamStr(I))
      end else
        if ParamStr(I) = '-oaa' then
          StartOAA := True
        else
          if ParamStr(I) = '-oaa_name' then begin
            I := I + 1;
            oaa.AgentName := ParamStr(I)
          end;
          I := I + 1
        end;
  end;
  If StartOAA then begin
    frmConnect.txtHost.Text := tcp.RemoteHost;
    frmConnect.txtPort.Text := IntToStr(tcp.RemotePort);
    tcp.Connect(frmConnect.txtHost.Text,
      StrToInt(frmConnect.txtPort.Text));
  end;
end;

procedure TfrmMain.mnuConnectClick(Sender: TObject);
begin

```

```

if not oaa.Connected then begin
    frmConnect.txtHost.Text := tcp.RemoteHost;
    frmConnect.txtPort.Text := IntToStr(tcp.RemotePort);
    if frmConnect.ShowModal = mrOK Then begin
        tcp.RemoteHost := frmConnect.txtHost.Text;
        tcp.RemotePort := StrToInt(frmConnect.txtPort.Text);
        mnuConnect.Caption := 'Disconnect from OAA';
        Status.SimpleText := 'Connecting to ' + tcp.RemoteHost + ', ' +
IntToStr(tcp.RemotePort) + '...';
        tcp.Connect(tcp.RemoteHost, tcp.RemotePort)
    end
end
else begin
    mnuConnect.Caption := 'OAA &Connect';
    tcp.Close;
    Status.SimpleText := 'Disconnected'
end
end;

```

```

procedure TfrmMain.oaaSendData(Sender: TObject; data: String);
begin
    tcp.SendData(data)
end;

```

```

procedure TfrmMain.tcpConnect(Sender: TObject);
begin
    Status.SimpleText := 'Connected: ' + tcp.Remotehost + ', ' +
IntToStr(tcp.RemotePort) + '.';
    oaa.Connect;
end;

```

```

procedure TfrmMain.tcpDataArrival(Sender: TObject; bytesTotal: Integer);
var
    data: OleVariant;
begin
    tcp.GetData(data, VarOleStr, bytesTotal);
    oaa.OnDataRead(data)
end;

```

```

procedure TfrmMain.tcpClose(Sender: TObject);
begin
    Status.SimpleText := 'Disconnected.';
    oaa.Disconnect
end;

```

```

procedure TfrmMain.mnuExitClick(Sender: TObject);
begin
    Halt
end;

```

```

procedure TfrmMain.Play(fname: String);
begin
    try
        MediaPlayer1.FileName := fname;
        MediaPlayer1.Open;
        MediaPlayer1.Play;
    except
    end
end;

```

```

(* Returns true if this callback handles incoming event, false otherwise *)
(* ans should be a list of solutions, with the empty list "[]" *)
(* representing failure. *)
function TfrmMain.oaaOAAEvent(Sender: TObject; ks, func, args: PChar;
  var ans: PChar): Boolean;
var goal, a, a2: PChar;
begin
  Result := True;
  a := nil;
  a2 := nil;
  goal := nil;
  (* Default answer: return success *)
  Ans := FStr(['s(%s)'], [func, args], [func, args]);

  (* Results are returned asynchronously from the Facilitator in
  a "solved" message. If there are results to a query, paste
  them to the clipboard and succeed, otherwise fail. *)
  if (StrComp(func, 'solved') = 0) and (ListLen(args) = 4) then begin
    NthElt(args, 2, goal);
    NthElt(args, 4, a);
    ListToTerms(a);
    if StrComp(a, '') <> 0 then begin
      Argument(a, 2, a2);
      RemoveQuotes(a2);
      Clipboard.SetTextBuf(UndoubleQuotes(a2));
      Play('good.wav');
    end
    else begin
      Play('bad.wav');
    end;
    StrFree(a);
    StrFree(a2);
    StrFree(goal);
    busy := False;
    keyCap.Enabled := true;
    Result := false
  end
  else Result := false
end;

function addVariable(s: PChar) : PChar;
var f, args, p: PChar;
begin
  Functor(s, f, args);
  if ListLen(args) = 1 then begin
    p := FStr(['s(%s, X)'], [f, args], [f, args])
  end
  else p := StrNew(s);
  StrFree(f);
  StrFree(args);
  addVariable := p
end;

function TfrmMain.ResolveArg(s: PChar) : PChar;
var f, args, p, a: PChar;
begin
  s := StrNew(s);
  Functor(s, f, args);

```



```

if ListLen(args) = 1 then begin
  if (StrPos(args, '(') <> nil) then begin
    p := AddVariable(args);
    oaa.Solve(p, [nil],[nil],'[[]', a);
    StrFree(p);
    ListToTerms(a);
    if StrComp(a, '') <> 0 then begin
      StrFree(s);
      Term(a, p, s);
      StrFree(a);
      Argument(p, 2, a);
      s := FStr('%s(%s)', [f,a],[f,a]);
      StrFree(a)
    end
  end
end;
StrFree(f);
StrFree(args);
ResolveArg := s;
end;

```

```

(* Procedure executed when the Target key is pressed.
  Finds a query on the clipboard, tries to translate
  the English request to an ICL task, and then send
  the ICL to the Facilitator agent for execution.
  Fail if an ICL translation can't be found. *)

```

```

procedure TfrmMain.keyCapKeyboardMessage(Sender: TObject; Key,
  lParam: Integer; var fDiscard: Boolean);
var
  s: array[0..300] of char;
  answers, p, p2: PChar;
begin
  answers := nil;
  p := nil;
  p2 := nil;
  fDiscard := False;
  if (Key = $7B) and // F5 key
    not busy
    and (Clipboard.GetTextBuf(s, sizeof(s)) > 0)
  then begin
    keyCap.Enabled := False;
    busy := True;
    answers := nil;
    if oaa.Connected then begin
      OAAgent.DoubleQuotes(s, p2);
      oaa.solve('convert_to_LF('%s',LF)', [p2],[p2],'[[]', answers);
      StrFree(p2);
      if StrComp(answers, '[[]') = 0 then begin
        Play('bad.wav');
        busy := False;
        keyCap.Enabled := true;
      end
    else begin
      Play('step.wav');
      ListToTerms(answers);
      Argument(answers, 2, p);
      StrFree(answers);
      if (StrPos(p, 'send(') = p) then begin
        Argument(p, 1, answers);
        txtTest.text := Strpas(answers);
      end
    end
  end
end;

```

```

        if (StrPos(answers, 'post_query(show(') = answers) then begin
            StrFree(p);
            Argument(answers, 1, p); StrFree(answers); // show(email())
            Argument(p, 1, answers); StrFree(p); // email('a');
            p := ResolveArg(answers); StrFree(answers);
            p2 := AddVariable(p); StrFree(p);
            answers := FStr('post_query(%s, [])', [p2], [p2]);
            StrFree(p2)
        end;
        oaa.PostEvent(answers, [nil], [nil])
    end
else begin
    Status.Simpletext := 'Strange: LF not wrapped in send()';
    busy := False;
    keyCap.Enabled := true;
end;
StrFree(p)
end;
StrFree(answers)
end
end
end;

procedure TfrmMain.FormCreate(Sender: TObject);
begin
    busy := False;
end;

procedure TfrmMain.Button1Click(Sender: TObject);
var d: Boolean;
begin
    keyCapKeyboardMessage(Sender, $7B, 0, d);
end;

end.

```

Multimodal Maps: An Agent-based Approach

Adam CHEYER and Luc JULIA

SRI International

333 Ravenswood Ave

Menlo Park, CA 94025 - USA

June 9, 1995

Abstract

In this paper, we discuss how multiple input modalities may be combined to produce more natural user interfaces. To illustrate this technique, we present a prototype map-based application for a travel planning domain. The application is distinguished by a synergistic combination of hand-writing, gesture and speech modalities; access to existing data sources including the World Wide Web; and a mobile handheld interface. To implement the described application, a distributed network of heterogeneous software agents was augmented by appropriate functionality for developing synergistic multimodal applications.

1 Introduction

As computer systems become more powerful and complex, efforts to make computer interfaces more simple and natural become increasingly important. Natural interfaces should be designed to facilitate communication in ways people are already accustomed to using. Such interfaces should allow users to concentrate on the tasks they are trying to accomplish, not worry about what they must do to control the interface.

In this paper, we begin by discussing what input modalities humans are comfortable using when interacting with computers, and how these modalities should best be combined in order to produce natural interfaces. In section three, we present a prototype map-based application for the travel planning domain which uses a synergistic combination of several input modalities. Section four describes the agent-based approach we used to implement

the application and the work on which it is based. In section five, we summarize our conclusions and future directions.

2 Natural Input

2.1 Input Modalities

Direct manipulation interface technologies are currently the most widely used techniques for creating user interfaces. Through the use of menus and a graphical user interface, users are presented with sets of discrete actions and the objects on which to perform them. Pointing devices such as a mouse facilitate selection of an object or action, and drag and drop techniques allow items to be moved or combined with other entities or actions.

With the addition of electronic pen devices, gestural drawings add a new dimension direct manipulation interfaces. Gestures allow users to communicate a surprisingly wide range of meaningful requests with a few simple strokes. Research has shown that multiple gestures can be combined to form dialog, with rules of temporal grouping overriding temporal sequencing [23]. Gestural commands are particularly applicable to graphical or editing type tasks.

Direct manipulation interactions possess many desirable qualities: communication is generally fast and concise; input techniques are easy to learn and remember; the user has a good idea about what can be accomplished, as the visual presentation of the available actions is generally easily accessible. However, direct manipulation suffers from limitations when trying to access or describe entities which are not or can not be visualized by the user.

Limitations of direct manipulation style interfaces can be addressed by another interface technology, that of natural language interfaces. Natural language interfaces excel in describing entities that are not currently displayed on the monitor, in specifying temporal relations between entities or actions, and in identifying members of sets. These strengths are exactly the weaknesses of direct manipulation interfaces, and concurrently, the weaknesses of natural language interfaces (ambiguity, conceptual coverage, etc.) can be overcome by the strengths of direct manipulation [6].

Natural language content can be entered through different input modalities, including typing, handwriting, and speech. It is important to note that, while the same textual con-

tent can be provided by the three modalities, each modality has widely varying properties.

- Spoken language is the modality used first and foremost in human-human interactive problem solving [4]. Speech is an extremely fast medium, several times faster than typing or handwriting. In addition, speech input contains content that is not present in other forms of natural language input, such as prosody, tone and characteristics of the speaker (age, sex, accent).
- Typing is the most common way of entering information into a computer, because it is reasonably fast, very accurate, and requires no computational resources.
- Handwriting has been shown to be useful for certain types of tasks, such as performing numerical calculations and manipulating names which are difficult to pronounce [18, 20]. Because of its relatively slow production rate, handwriting may induce users to produce different types of input than is generated by spoken language; abbreviations, symbols and non-grammatical patterns may be expected to be more prevalent amid written input.

2.2 Combination of Modalities

As noted in the previous section, direct manipulation and natural language seem to be very complementary modalities. It is therefore not surprising that a number of multimodal systems combine the two.

Notable among such systems is the Cohen's Shoptalk system [6], a prototype manufacturing and decision-support system that aids in tasks such as quality assurance monitoring, and production scheduling. The natural language module of Shoptalk is based on the Chat-85 natural language system [26] and is particularly good at handling time, tense, and temporal reasoning.

A number of systems have focused on combining the speed of speech with the reference provided by direct manipulation of a mouse pointer. Such systems include the XTRA system [1], CUBRICON [15], the PAC-Amodeus model [16], and TAPAGE [9, 12].

XTRA and CUBRICON are both systems that combine complex spoken input with mouse clicks, using several knowledge sources for reference identification. CUBRICON's domain is a map-based task, making it similar to the application developed in this paper. However, the two are different in that CUBRICON can only use direct manipulation to

indicate a specific item, whereas our system produces a richer mixing of modalities by adding both gestural and written language as input modalities.

PAC-Amodeus systems such as VoicePaint and Notebook allow the user to synergistically combine vocal or mouse-click commands when interacting with notes or graphical objects. However, due in part to the selected domains, the natural language input is very simple, generally of the style “Insert a note here.”

TAPAGE is another system that allows true synergistic combination of spoken input with direct manipulation. Like PAC-Amodeus, TAPAGE’s domain provides only simple linguistic input. However, TAPAGE uses a pen-based interface instead of a mouse, allowing gestural commands. TAPAGE, selected as one of the “building blocks” for our map application, will be described more in detail in section 4.2.

Other pertinent work regarding the simultaneous combination of handgestures and gaze can be found in [2, 13].

3 A Multimodal Map Application

In this section, we will describe a prototype map-based application for a travel planning domain. In order to provide the most natural user interface possible, the system permits the user to simultaneously combine direct manipulation, gestural drawings, handwritten, typed and spoken natural language. When designing the architecture for the system, other criteria were considered as well:

- The user interface must be light and fast enough to run on a handheld PDA while able to access applications and data that may require a more powerful machine.
- Existing commercial or research natural language and speech recognition systems should be used.
- Through the multimodal interface, a user must be able to transparently access a wide variety of data sources, including information stored in HTML form on the World Wide Web.

The map functionality, interface design, and classes of input data of the system presented here is based on a design by Oviatt and Cohen, used by them in a wizard-of-oz simulation system designed to explore complex interactions of modalities [19]. The agent-based architecture used to realize Oviatt and Cohen’s design is new, as is its application



Figure 1: Multimodal Application for Travel Planning

to travel planning.

As illustrated in Figure 1, the user is presented with a pen sensitive map display on which drawn gestures and written natural language statements may be combined with spoken input. As opposed to a static paper map, the location, resolution, and content presented by the map change, according to the requests of the user. Objects of interest, such as restaurants, movie theaters, hotels, tourist sites, municipal buildings, etc. are displayed as icons. The user may ask the map to perform various actions. For example :

- *distance calculation* : e.g. “How far is the hotel from Fisherman’s Wharf?”
- *object location* : e.g. “Where is the nearest post office?”
- *filtering* : e.g. “Display the French restaurants within 1 mile of this hotel.”
- *information retrieval* : e.g. “Show me all available information about Alcatraz.”

The application also makes use of multimodal (multimedia) output as well as input: video, text, sound and voice can all be combined when presenting an answer to a query.

During input, requests can be entered using gestures (Figure 2), handwriting, voice, or a combination of pen and voice. For instance, in order to calculate the distance between two points on the map, a command may be issued using the following:

- *gesture*, by simply drawing a line between the two points of interest.
- *voice*, by speaking “What is the distance from the post office to the hotel?”.

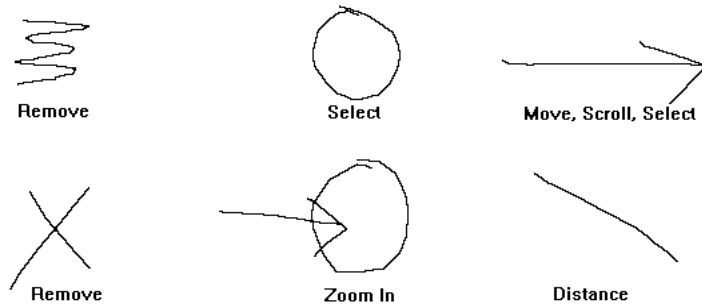


Figure 2: Sample gestures

- *handwriting*, by writing “dist p.o. to hotel?”
- *synergistic combination of pen and voice*, by speaking “What is the distance from here to this hotel?” while simultaneously indicating the specified locations by pointing or circling.

Notice that in our example of synergistic combination of pen and voice, the arguments to the verb “distance” can be specified before, at the same time, or shortly after the vocalization of the request to calculate the distance. If a user’s request is ambiguous or underspecified, the system will wait several seconds and then issue a prompt requesting additional information.

The user interface runs on pen-equipped PC’s or a Dauphin handheld PDA ([7]) using either a microphone or a telephone for voice input. The interface is connected either by modem or ethernet to a server machine which will manage database access, natural language processing and speech recognition for the application. The result is a mobile system that provides a synergistic pen/voice interface to remote databases.

In general, the speed of the system is quite acceptable. For gestural commands, which are handled locally on the user interface machine, a response is produced in less than one second. For handwritten commands, the time to recognize the handwriting, process the English query, access a database and begin to display the results on the user interface is less than three seconds (assuming an ethernet connection, and good network and database response). Solutions to verbal commands are displayed in three to five seconds after the end of speech has been detected; partial feedback indicating the current status of the speech recognition is provided earlier.

4 Approach

In order to implement the application described in the previous section, we chose to augment a proven agent-based architecture with functionalities developed for a synergistically multimodal application. The result is a flexible methodology for designing and implementing distributed multimodal applications.

4.1 Building Blocks

4.1.1 Open Agent Architecture

The Open Agent Architecture (OAA) [5] provides a framework for coordinating a society of agents which interact to solve problems for the user. Through the use of agents, the OAA provides distributed access to commercial applications, such as mail systems, calendar programs, databases, etc.

The Open Agent Architecture possesses several properties which make it a good candidate for our needs:

- An Interagent Communication Language (ICL) and Query Protocol have been developed, allowing agents to communicate among themselves. Agents can run on different platforms and be implemented in a variety of programming languages.
- Several natural language systems have been integrated into the OAA which convert English into the Interagent Communication Language. In addition, a speech recognition agent has been developed to provide transparent access to the Corona speech recognition system.
- The agent architecture has been used to provide natural language and agent access to various heterogeneous data and knowledge sources.
- Agent interaction is very fine-grained. The architecture was designed so that a number of agents can work together, when appropriate in parallel, to produce fast responses to queries.

The architecture for the OAA, based loosely on Schwartz's FLiPSiDE system[24], uses a hierarchical configuration where client agents connect to a "facilitator" server. Facilitators provide content-based message routing, global data management, and process coordination for their set of connected agents. Facilitators can, in turn, be connected as clients of other

facilitators. Each facilitator records the published functionality of their sub-agents, and when queries arrive in Interagent Communication Language form, they are responsible for breaking apart any complex queries and for distributing goals to the appropriate agents. An agent solving a goal may require supporting information and the agent architecture provides numerous means of requesting data from other agents or from the user.

Among the assortment of agent architectures, the Open Agent Architecture can be most closely compared to work by the ARPA knowledge sharing community [10]. The OAA's query protocol, Interagent Communication Language and Facilitator mechanisms have similar instantiations in the SHADE project, in the form of KQML, KIF and various independent capability matchmakers. Other agent architectures, such as General Magic's Telescript [11], MASCOS [21], or the CORBA distributed object approach [17] do not provide as fully developed mechanisms for interagent communication and delegation.

The Open Agent Architecture provides capability for accessing distributed knowledge sources through natural language and voice, but it is lacking integration with a synergistic multimodal interface.

4.1.2 TAPAGE

TAPAGE (edition de Tableaux par la Parole et la Geste) is a synergistic pen/voice system for designing and correcting tables.

To capture signals emitted during a user's interaction, TAPAGE integrates a set of modality agents, each responsible for a very specialized kind of signal [9]. The modality agents are connected to an "interpret agent" which is responsible for combining the inputs across all modalities to form a valid command for the application. The interpret agent receives filtered results from the modality agents, sorts the information into the correct fields, performs type-checking on the arguments, and prompts the user for any missing information, according to the model of the interaction. The interpret agent is also responsible for merging the data streams sent by the modality agents, and for resolving ambiguities among them, based on its knowledge of the application's internal state. Another function of the interpret agent is to produce reflexes: reflexes are actions output at the interface level without involving the functional core of the application.

The TAPAGE system can accept multimodal input, but it is not a distributed system; its functional core is fixed. In TAPAGE, the set of linguistic input is limited to a *verb*

object argument format.

4.2 Synthesis

In the Open Agent Architecture, agents are distributed entities that can run on different machines, and communicate together to solve a task for the user. In TAPAGE, agents are used to provide streams of input to a central interpret process, responsible for merging incoming data. A generalization of these two types of agents could be :

Macro Agents: contain some knowledge and ability to reason about a domain, and can answer or make queries to other macro agents using the Interagent Communication Language.

Micro Agents: are responsible for handling a single input or output data stream, either filtering the signal to or from a hierarchically superior “interpret” agent.

The network architecture that we used was hierarchical at two resolutions – micro agents are connected to a superior macro agent, and macro agents are connected in turn to a facilitator agent. In both cases, a server is responsible for the supervision of its client sub-agents.

In order to describe our implementation, we will first give a description of each agent used in our application and then illustrate the flow of communication among agents produced by a user’s request.

Speech Recognition (SR) Agent: The SR agent provides a mapping from the Interagent Communication Language to the API for the Decipher (Corona) speech recognition system [4], a continuous speech speaker independent recognizer based on Hidden Markov Model technology. This macro agent is also responsible for supervising a child micro agent whose task is to control the speech data stream. The SR agent can provide feedback to an interface agent about the current status and progress of the micro agent (e.g. “listening”, “end of speech detected”, etc.) This agent is written in C.

Natural Language (NL) Parser Agent: translates English expressions into the Interagent Communication Language (ICL). For a more complete description of the ICL, see [5]. The NL agent we selected for our application is the simplest of those integrated into the OAA. It is written in Prolog using Definite Clause Grammars, and supports a distributed vocabulary; each agent dynamically adds word definitions as it connects to the network. A current project is underway to integrate the Gemini natural language system [8], a robust

bottom up parser and semantic interpreter specifically designed for use in Spoken Language Understanding projects.

Database Agents: Database agents can reside at local or remote locations and can be grouped hierarchically according to content. Micro agents can be connected to database agents to monitor relevant positions or events in real time. In our travel planning application, database agents provide maps for each city, as well as icons, vocabulary and information about available hotels, restaurants, movies, theaters, municipal buildings and tourist attractions. Three types of databases were used: Prolog databases, X.500 hierarchical databases, and data loaded automatically by scanning HTML pages from the World Wide Web (WWW). In one instance, a local newspaper provides weekly updates to its Mosaic-accessible list of current movie times and reviews, as well as adding several new restaurant reviews to a growing collection; this information is extracted by an HTML reading database agent and made accessible to the agent architecture. Descriptions and addresses of new restaurants are presented to the user on request, and the user can choose to add them to the permanent database by specifying positional coordinates on the map (eg. “add this new restaurant here”), information lacking in the WWW database.

Reference Resolution Agent: This agent is responsible for merging requests arriving in parallel from different modalities, and for controlling interactions between the user interface agent, database agents and modality agents. In this implementation, the reference resolution agent is domain specific: knowledge is encoded as to what actions must be performed to resolve each possible type of ICL request in its particular domain. For a given ICL logical form, the agent can verify argument types, supply default values, and resolve argument references. Some argument references are descriptive (“How far is it to the hotel on Emerson Street?”); in this case, a domain agent will try to resolve the definite reference by sending database agent requests. Other references, particularly when contextual or deictic, are resolved by the user interface agent (“What are the rates for this hotel?”). Once arguments to a query have been resolved, this agent agent coordinates the actions and calculations necessary to produce the result of the request.

Interface Agent: This macro agent is responsible for managing what is currently being displayed to the user, and for accepting the user’s multimodal input. The Interface Agent also coordinates client modality agents and resolves ambiguities among them : handwriting and gestures are interpreted locally by micro agents and combined with results from the

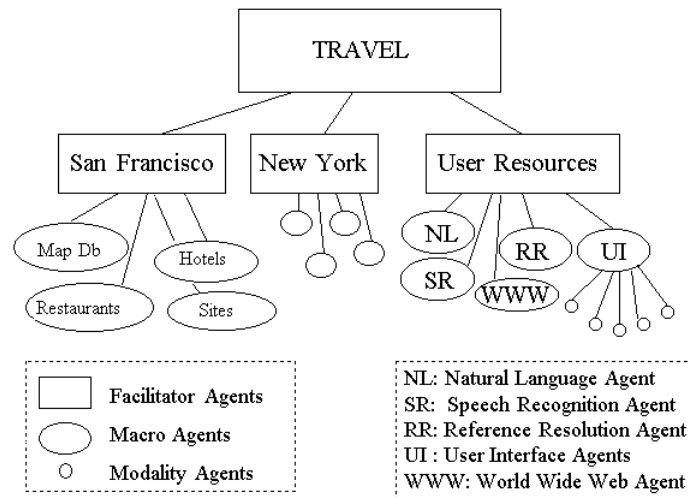


Figure 3: Agent Architecture for Map Application

speech recognition agent, running on a remote speech server. The handwriting micro-agent interfaces with the Microsoft PenWindows API and accesses a handwriting recognizer by CIC Corporation. The gesture micro-agent accesses recognition algorithms developed for TAPAGE.

An important task for the interface agent is to record which objects of each type are currently salient, in order to resolve contextual references such as “the hotel” or “where I was before.” Deictic references are resolved by gestural or direct manipulation commands. If no such indication is currently specified, the user interface agent waits long enough to give the user an opportunity to supply the value, and then prompts the user for it.

We shall now give an example of the distributed interaction of agents for a specific query. In the following example, all communication among agents passes transparently through a facilitator agent in an undirected fashion; this process is left out of the description for brevity.

1. A user speaks: “How far is the restaurant from this hotel?”
2. The speech recognition agent monitors the status and results from its micro agent, sending feedback received by the user interface agent. When the string is recognized, a translation is requested.
3. The English request is received by the NL agent and translated into ICL form.

4. The reference resolution agent (RR) receives the ICL distance request containing one definite and one deictic reference and asks for resolution of these references.
5. The interface agent uses contextual structures to find what “the restaurant” refers to, and waits for the user to make a gesture indicating “the hotel”, issuing prompts if necessary.
6. When the references have been resolved, the domain agent (RR) sends database requests asking for the coordinates of the items in question. It then calculates the distance according to the scale of the currently displayed map, and requests the user interface to produce output displaying the result of the calculation.

5 CONCLUSIONS

By augmenting an existing agent-based architecture with concepts necessary for synergistic multimodal input, we were able to rapidly develop a map-based application for a travel planning task. The resulting application has met our initial requirements: a mobile, synergistic pen/voice interface providing good natural language access to heterogeneous distributed knowledge sources. The approach used was general and should provide a means for developing synergistic multimodal applications for other domains.

The system described here is one of the first that accepts commands made of synergistic combinations of spoken language, handwriting and gestural input. This fusion of modalities can produce more complex interactions than in many systems and the prototype application will serve as a testbed for acquiring a better understanding of multimodal input.

In the near future, we will continue to verify and extend our approach by building other multimodal applications. We are interested in generalizing the methodology further; work has already begun on an agent-building tool which will simplify and automate many of the details of developing new agents and domains.

6 Acknowledgements

The work reported here would not have been possible without the inspiration of Sharon Oviatt and Phil Cohen under whose direction we worked for a year on a project (NSF Grant No. IRI-9213472) in which the combination of modalities contained in the interface presented here was crystallized and studied via simulations. Neither they nor their

sponsors, of course, are responsible for the work presented here.

References

- [1] Allegayer, J, Jansen-Winkel, R., Reddig, C. and Reithinger, N. "Bidirectional use of knowledge in the multi-modal NL access system XTRA". In Proceedings of IJCAI-89, Detroit, pp. 1492-1497.
- [2] Bolt, R. "Put that there: Voice and Gesture at the Graphic Interface". Computer Graphics, 14(3), 1980, pp. 262-270.
- [3] Bellik, Y. and Teil, D. "Les types de multimodalites", In Proc. IIM'92 (Paris), pp. 22-28.
- [4] Cohen, M., Murveit, H., Bernstein, J., Price, P., Weintraub, M., "The DECIPHER Speech Recognition System". 1990 IEEE ICASSP, pp. 77-80.
- [5] Cohen, P.R., Cheyer, A., Wang, M. and Baeg, S.C. "An Open Agent Architecture". In Proc. AAAI'94 - SA (Stanford), pp. 1-8.
- [6] Cohen, P. "The role of natural language in a multimodal interface". Proceedings of UIST'92, 143-149.
- [7] Dauphin DTR-1 User's Manual, Dauphin Technology, Inc. 337 E. Butterfield Rd., Suite 900, Lombard, Ill 60148.
- [8] Dowding, J., Gawron, J.M., Appelt, D., Bear, J., Cherny, L., Moore, B. and Moran D., "Gemini: A natural language system for spoken-language understanding", Technical Note 527, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, April 1993.
- [9] Faure, C. and Julia, L. "An Agent-Based Architecture for a Multimodal Interface". In Proc. AAAI'94 - IM4S (Stanford), pp. 82-86.
- [10] Genesereth, M. and Singh, N.P. "A knowledge sharing approach to software interoperation". Computer Science Department, Stanford University, unpublished ms., 1994.
- [11] General Magic, Inc., "Telescript Product Documentation", 1995.
- [12] Julia, L. and Faure, C. "A Multimodal Interface for Incremental Graphic Document Design". HCI International '93, Orlando.

- [13] Koons, D.B., Sparrell, C.J., and Thorisson, K.R. "Integrating Simultaneous Input from Speech, Gaze and Hand Gestures". In *Intelligent Multimedia Interfaces*, Edited by Mark Maybury, Menlo Park, CA, AAAI Press, 1993.
- [14] Maybury, M.T. (ed.), *Intelligent Multimedia Interfaces*, AAAI Press/MIT Press: Menlo Park, Ca, 1993.
- [15] Neal, J.G., and Shapiro, S.C. "Intelligent Multi-media Interface Technology". In *Intelligent User Interfaces*, Edited by J. Sullivan and S. Tyler, Addison-Wesley Pub. Co., Reading, MA, 1991.
- [16] Nigay, L. and Coutaz, J. "A Design Space for Multimodal Systems: Concurrent Processing and Data Fusion". In Proc. InterCHI'93 (Amsterdam), ACM Press, pp. 172-178.
- [17] Object Management Group, "The Common Object Request Broker: Architecture and Specification", OMG Document Number 91.12.1, December 1991.
- [18] Oviatt, S. "Toward Empirically-Based Design of Multimodal Dialogue Systems". In Proc. AAAI'94 - IM4S (Stanford), pp. 30-36.
- [19] Oviatt, S. "Multimodal Interactive Maps: Designing for Human Performance". Forthcoming journal publication.
- [20] Oviatt, S. and Olsen, E. "Integration Themes in Multimodal Human-Computer Interaction". Proceedings of ICSLP'94, Yokohama, pp. 551-554.
- [21] Park, S.K., Choi J.M., Myeong-Wuk J., Lee G.L., and Lim Y.H. "MASCOS : A Multi-Agent System as the Computer Secretary". Submitted for publication.
- [22] Pfaff, G. and Ten Hagen, P.J.W. *Seeheim workshop on User Interface Management Systems* (Berlin), Springer- Verlag.
- [23] Rhyne J. "Dialogue Management for Gestural Interfaces". Computer Graphics, 21(2), 1987, pp. 137-142.
- [24] Schwartz, D.G. "Cooperating heterogeneous systems: A blackboard-based meta approach". Technical Report 93-112, Center for Automation and Intelligent Systems Research, Case Western Reserve University, Cleveland Ohio, April 1993. Unpublished PhD. thesis.
- [25] Sullivan, J. and Tyler, S. (eds.), *Intelligent User Interfaces*, Addison-Wesley Pub. Co., Reading, MA, 1991.

- [26] Warren, D. and Pereira, F., "An Efficient Easily Adaptable System for Interpreting Natural Language Queries", in *American Journal of Computational Linguistics*, 8(3), 1982, pp. 110-123.
- [27] Wauchope, K., "Eucalyptus: Integrating Natural Language with a Graphical User Interface." Naval Research Laboratory Technical Report NRL/FR/5510-94-9711, in press, 1994.



US005197005A

United States Patent [19]

[11] Patent Number: **5,197,005**

Shwartz et al.

[45] Date of Patent: **Mar. 23, 1993**

[54] DATABASE RETRIEVAL SYSTEM HAVING A NATURAL LANGUAGE INTERFACE

[75] Inventors: **Steven Shwartz**, Orange; **Claudio Fratarcangeli**, Trumbull; **Richard E. Cullingford**, Monroe; **Gregory S. Aimi**, North Haven; **Donald P. Strasburger**, Stratford, all of Conn.

[73] Assignee: **Intelligent Business Systems**, Milford, Conn.

[21] Appl. No.: **345,966**

[22] Filed: **May 1, 1989**

[51] Int. Cl.⁵ **G06F 15/40**

[52] U.S. Cl. **364/419**; 395/600; 364/DIG. 1; 364/274; 364/274.2; 364/274.3; 364/274.8

[58] Field of Search 364/513, 419, 200 MS File, 364/900 MS File; 395/600, 12

[56] References Cited

U.S. PATENT DOCUMENTS

4,736,296	4/1988	Katayama et al.	364/419
4,811,207	3/1989	Hikita et al.	364/900
4,839,853	6/1989	Deerwester et al.	364/900
4,914,590	4/1990	Loatman et al.	364/419
4,930,071	5/1990	Tou et al.	314/300
4,931,935	6/1990	Ohira et al.	364/419
4,943,933	7/1990	Miyamoto et al.	364/513
4,974,191	11/1990	Amirghodsi et al.	395/700
4,994,967	2/1991	Asakawa	364/419

FOREIGN PATENT DOCUMENTS

63-219034 9/1988 Japan .

OTHER PUBLICATIONS

Winston, "Natural Language Understanding", *Artificial Intelligence*, CH. 9, pp. 291-334.

Rich, "Natural Language Interfaces", *IEEE Computer*, Sep. 1984, pp. 39-47.

Kao et al., Providing Quality Responses with Natural Language Interfaces: the Null Value Problem, *IEEE Trans. Software Eng.*, vol. 14, No. 7, 1988.

"Natural Language Interfaces: Benefits, Requirements, State of the Art and Applications", by John L. Manferdelli, *A. I. East*, Oct. 1987.

"Inside Computer Understanding", Schank and Riesbeck, Erlbaum Press, 1981, Chapter 14.

"The LIFER Manual: A Guide to Building Practical Natural Language Interfaces", by Gary G. Hendrix, SRI International, Technical Note 138, Feb. 1977.

"Human Engineering for Applied Natural Language Processing", by Gary G. Hendrix, *SRI International*, Technical Note 139, Feb. '77.

"Applied Natural Language Processing", Shwartz, Steven C., Petrocelli Books, Princeton, N.J., 1987.

Primary Examiner—Michael R. Fleming

Assistant Examiner—Debra A. Chun

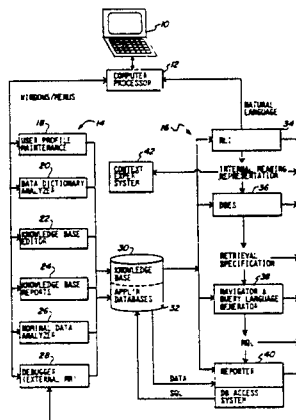
Attorney, Agent, or Firm—Barry R. Lipsitz

[57] ABSTRACT

A database retrieval system having a natural language interface is provided. A database developer creates a knowledge base containing a structural description and semantic description of an application database from which data is to be retrieved. A database independent, canonical internal meaning representation of a natural language query is produced. An expert system accesses structural and semantic description information in the knowledge base and, in accordance with predefined rules, identifies database elements from said information that are necessary to satisfy the query represented by the internal meaning representation. A database query is generated among the database elements, enabling the retrieval and aggregation of data from the database to satisfy the natural language query. A debugging facility derives an external meaning representation from the internal meaning representation. The external meaning representation is database-independent, canonical, and easily understandable to the database developer. The external meaning representation enables the database developer to comprehend the internal meaning representation and verify that a natural language query is properly interpreted by the system to effect the accurate retrieval and aggregation of data from the database. The external meaning representation comprises entities and constraints relating to the entities, without reference to factual or linguistic relationships between entities that would prevent the external meaning representation from being easily understood.

41 Claims, 11 Drawing Sheets

Microfiche Appendix Included
(11,603 Microfiche, 47 Pages)



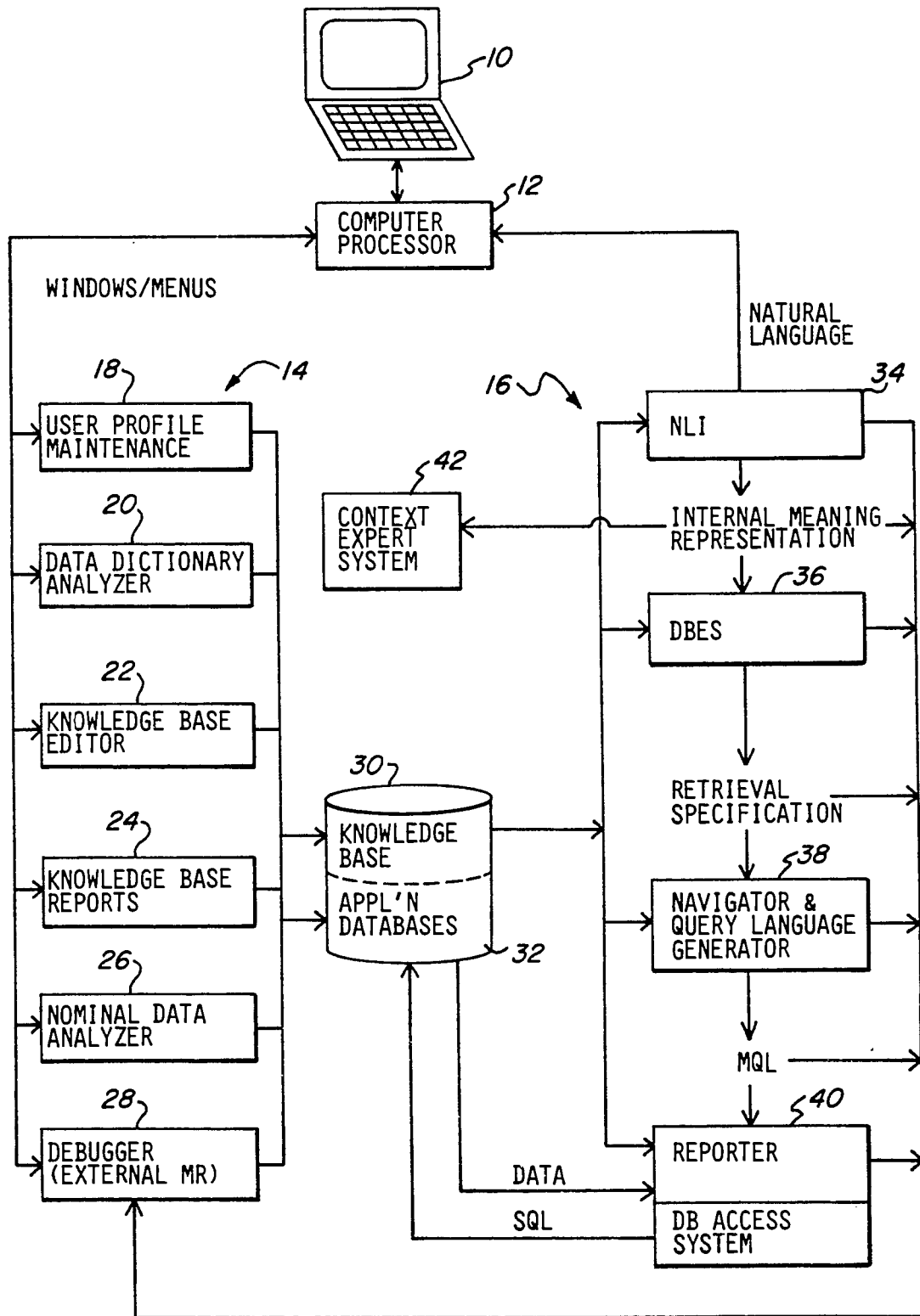


FIG. 1

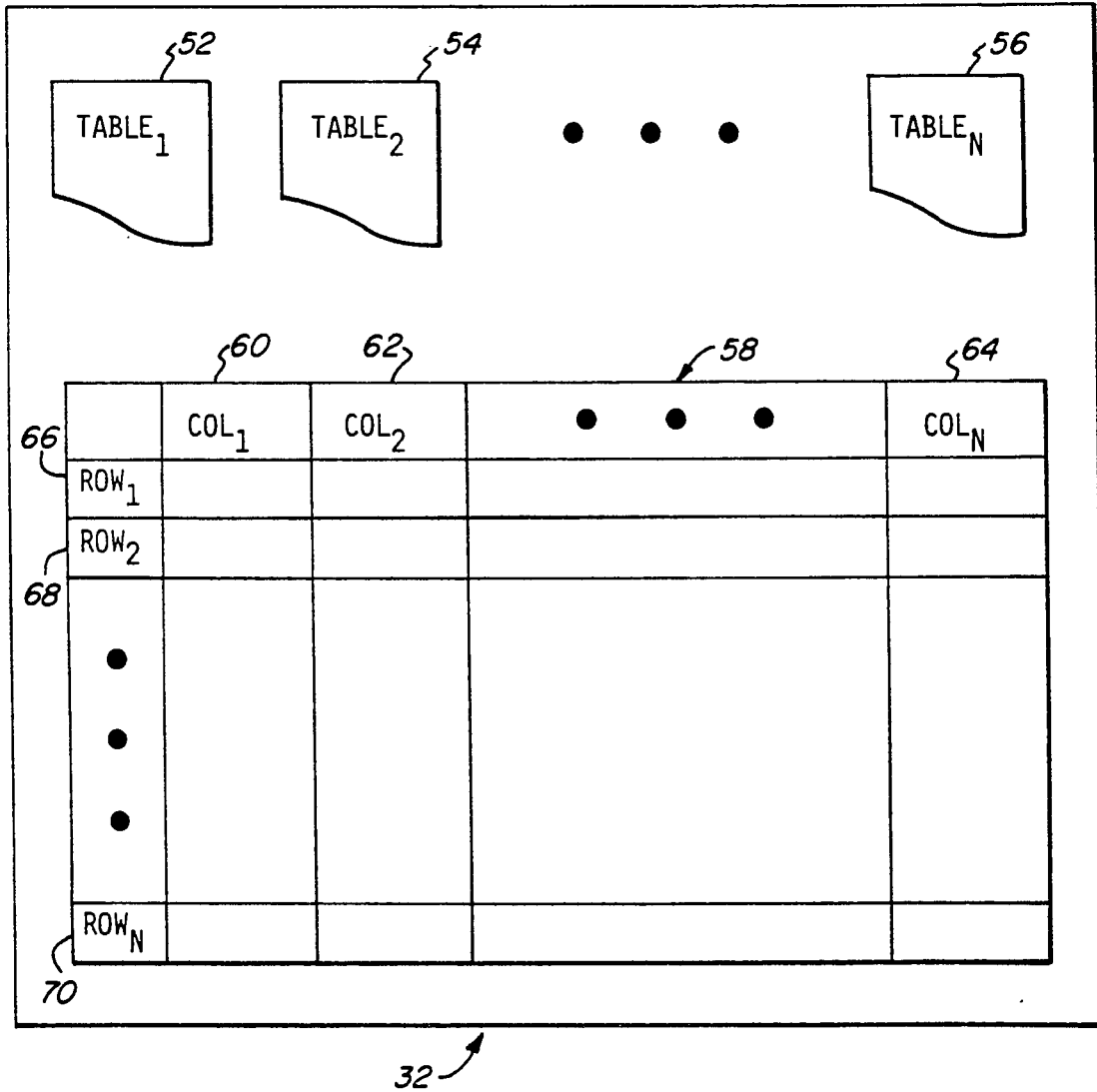


FIG. 2

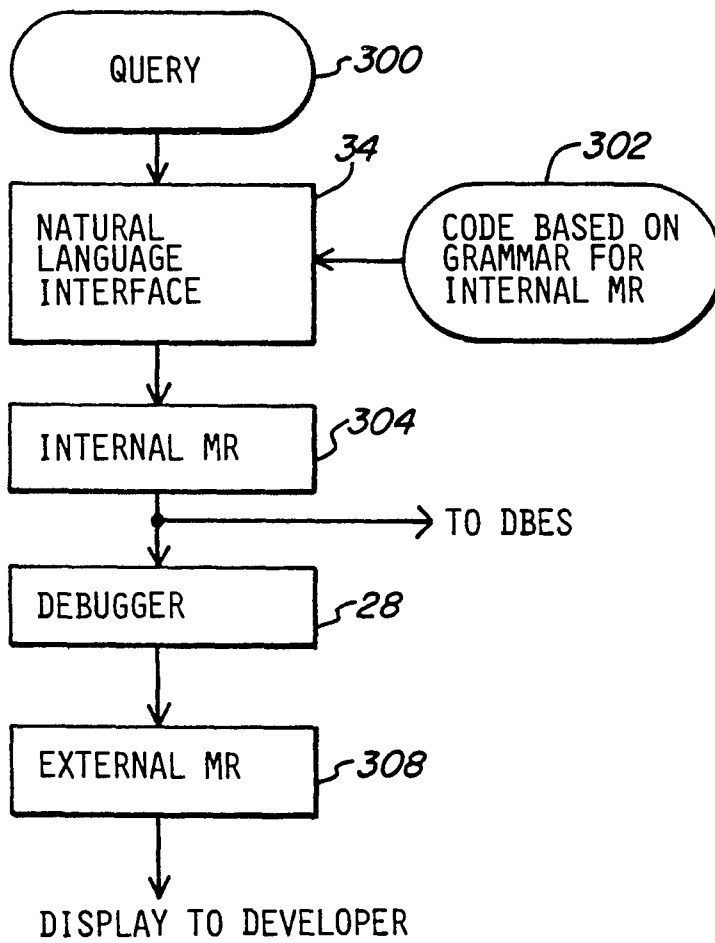


FIG. 3

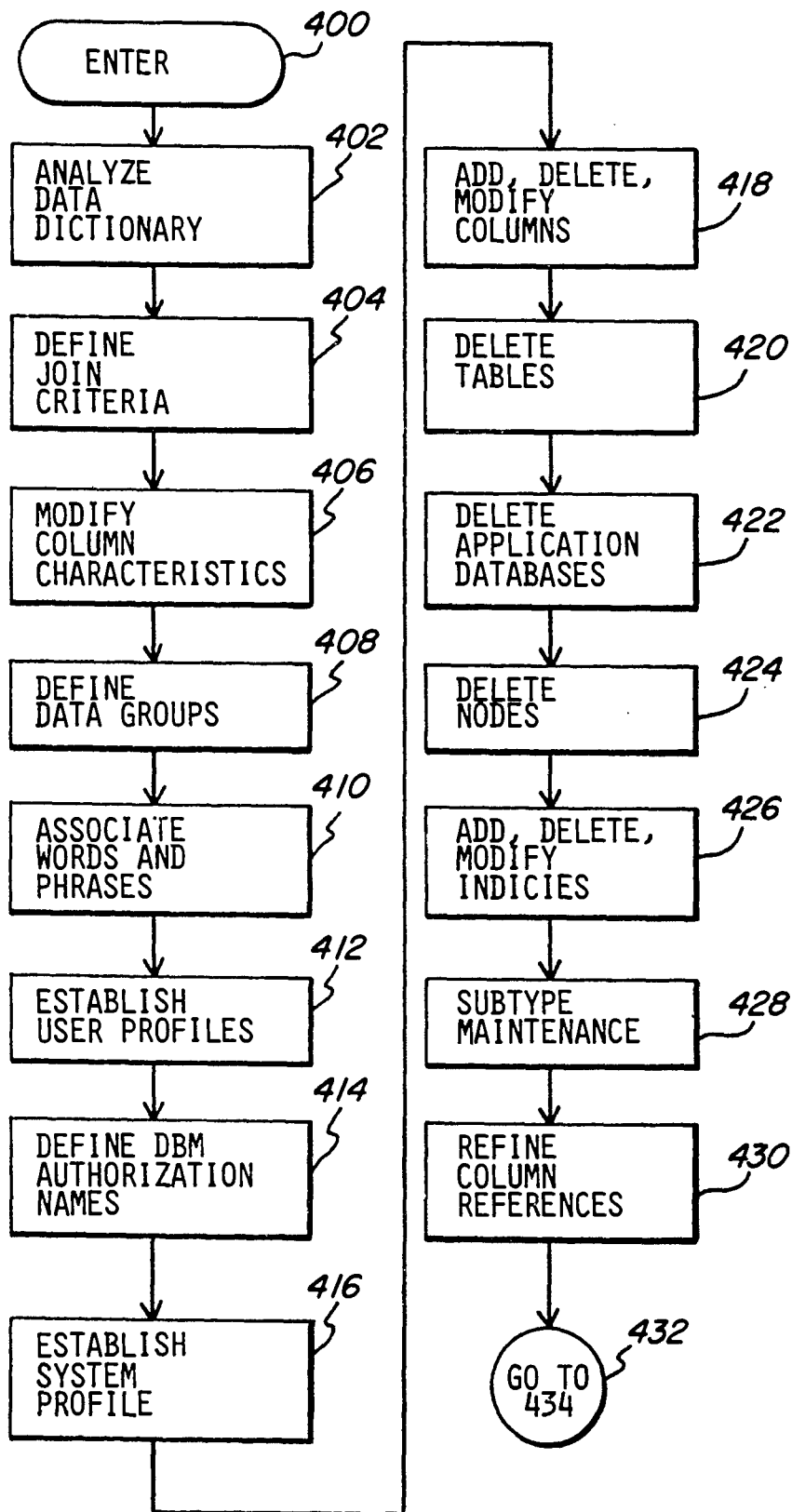


FIG. 4a

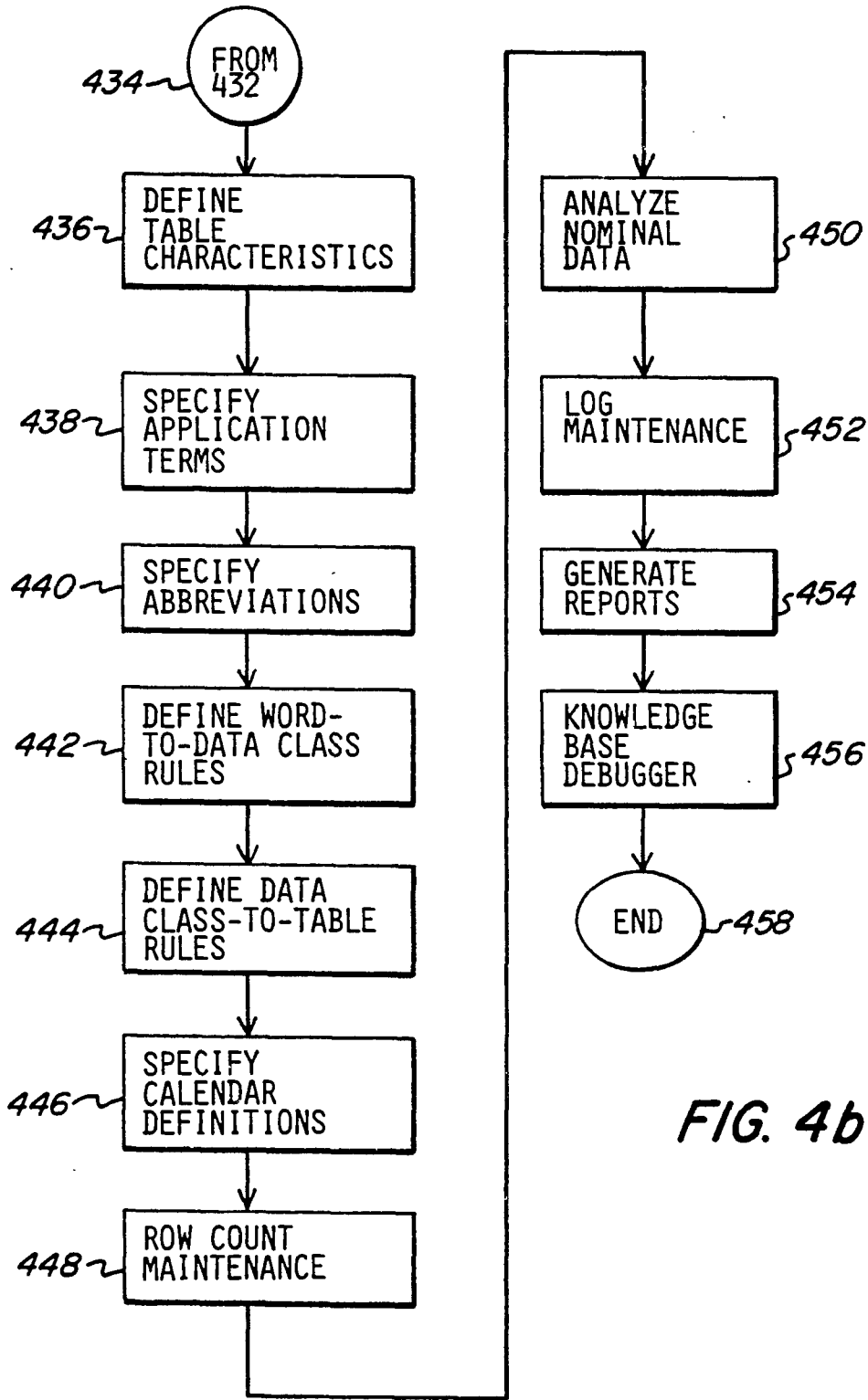


FIG. 4b

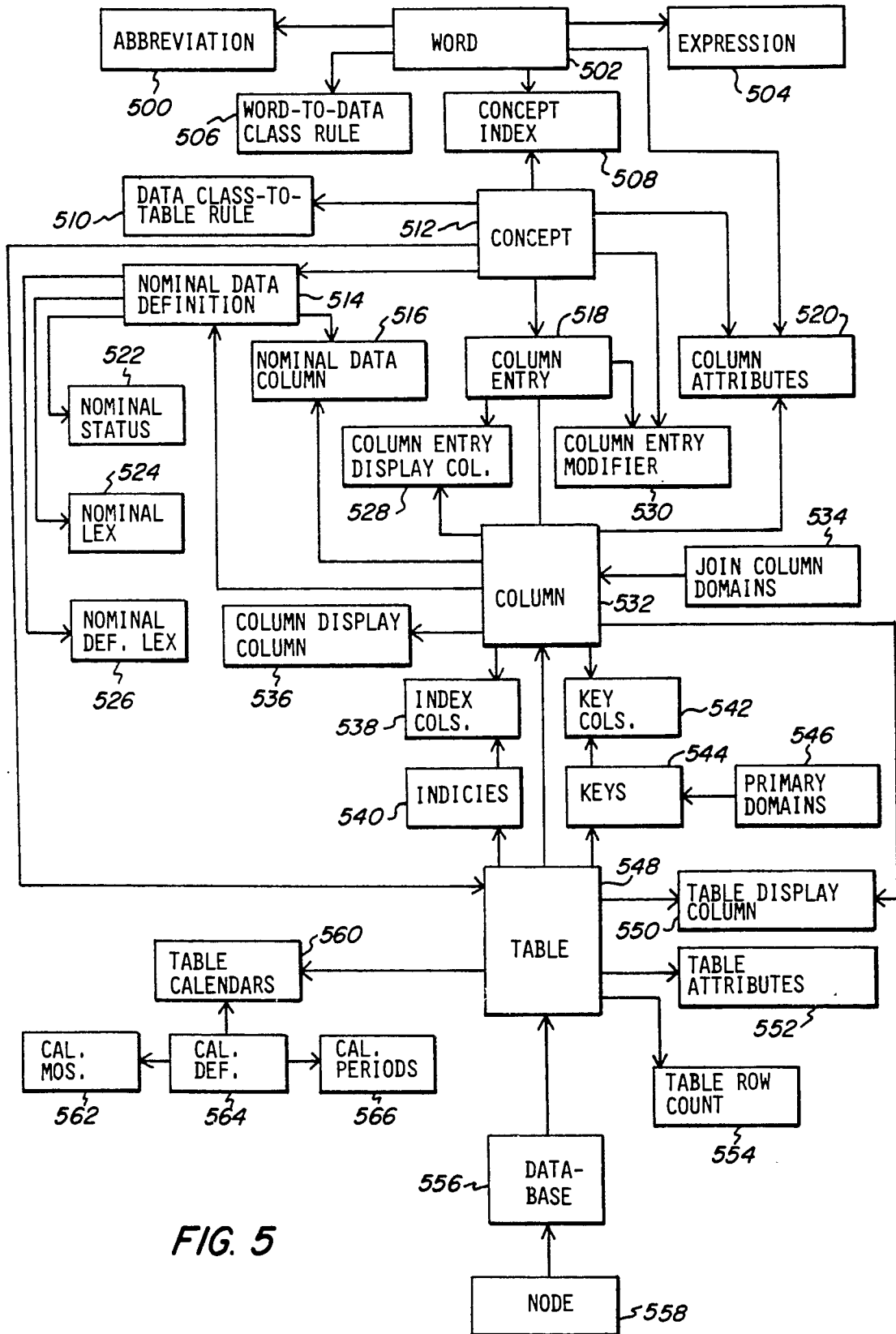


FIG. 5

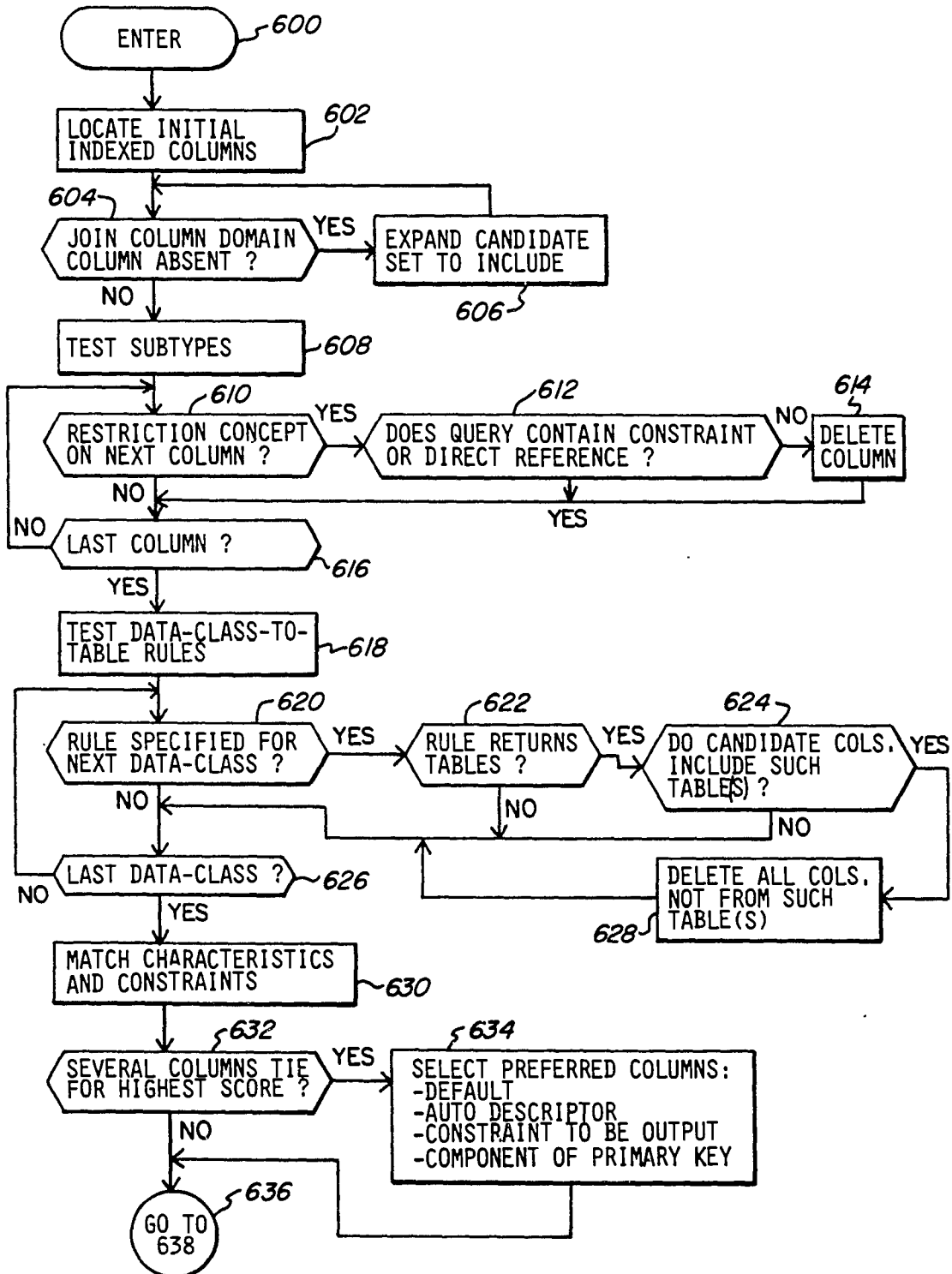


FIG. 6a

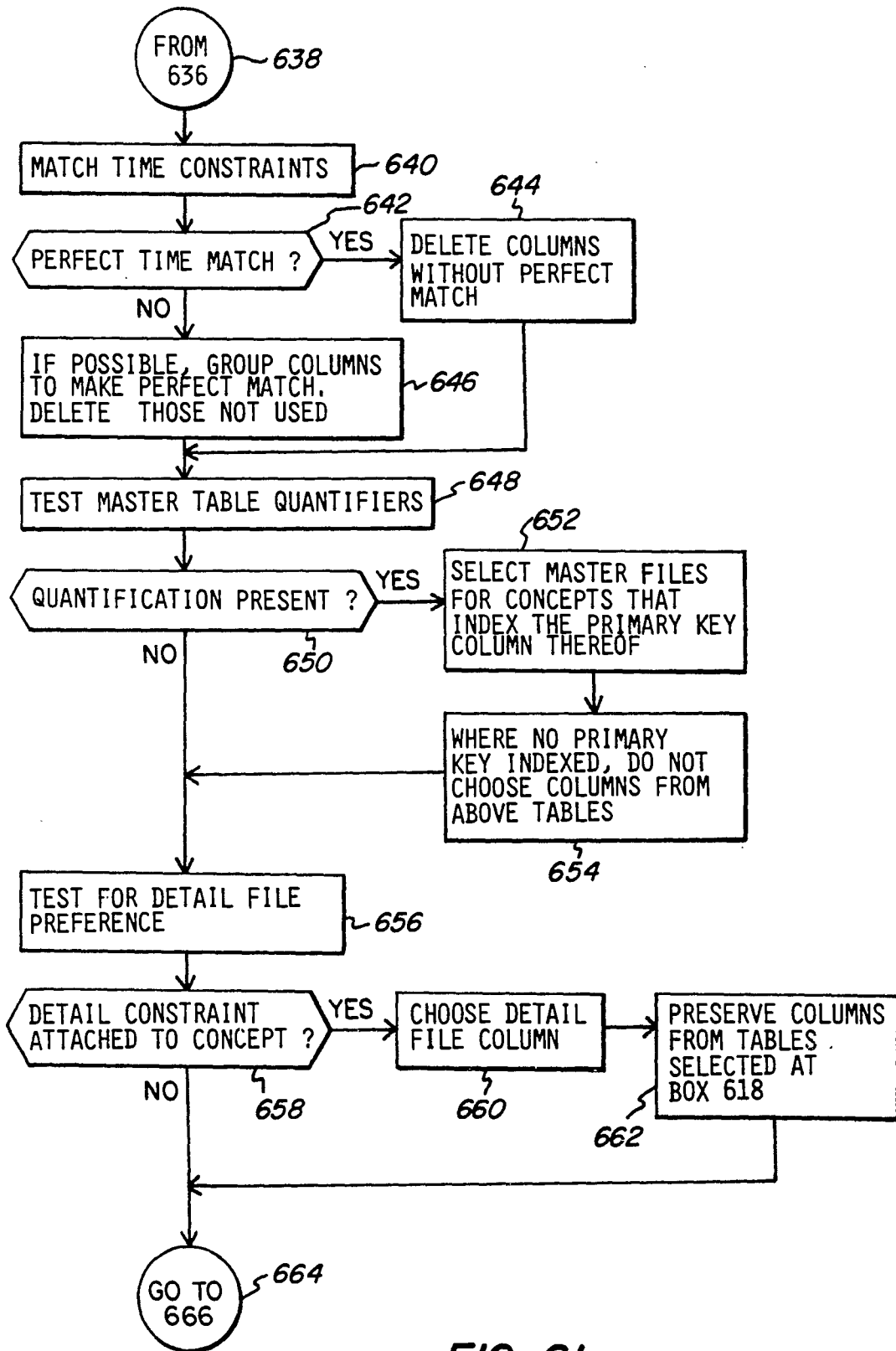


FIG. 6b

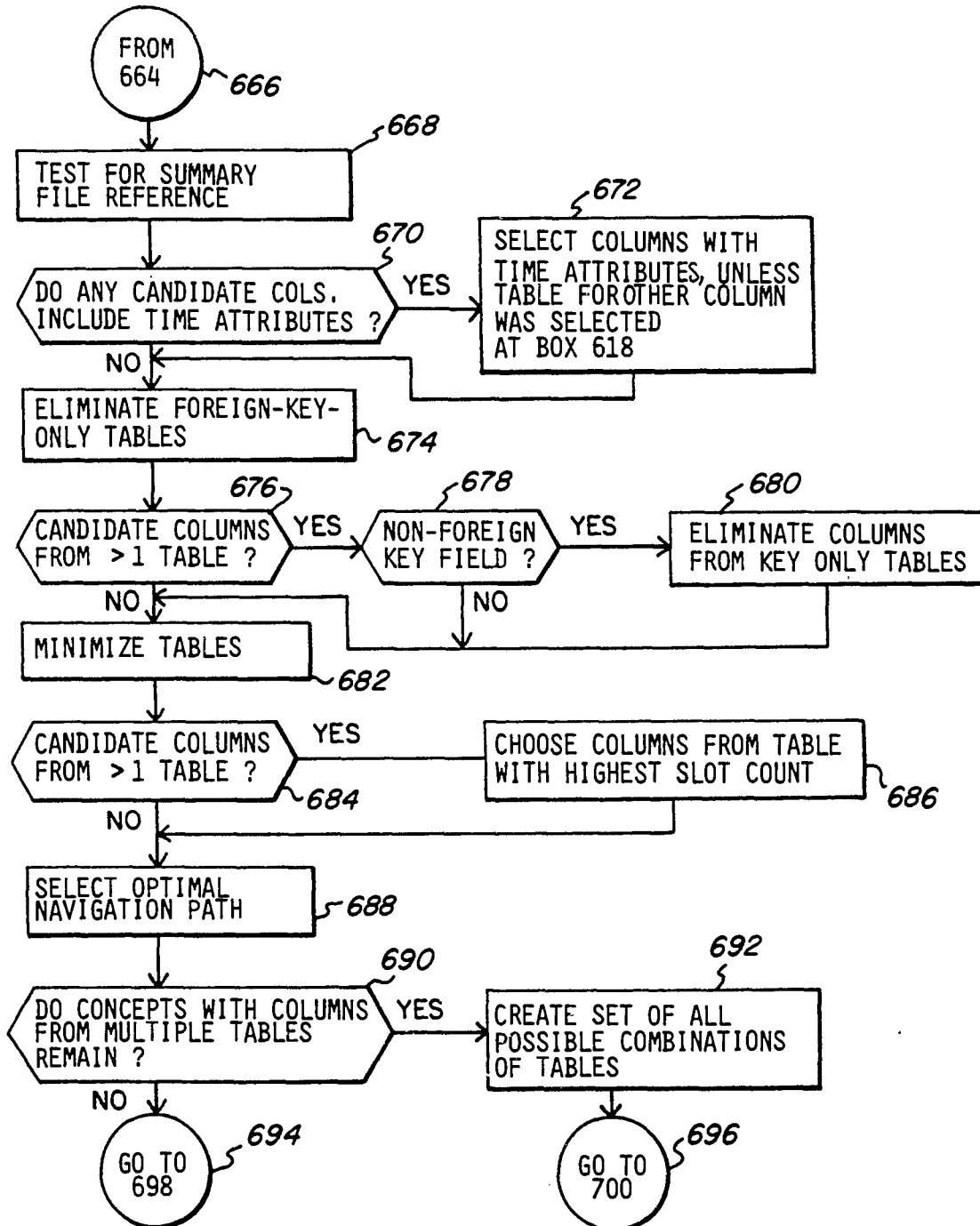


FIG. 6c

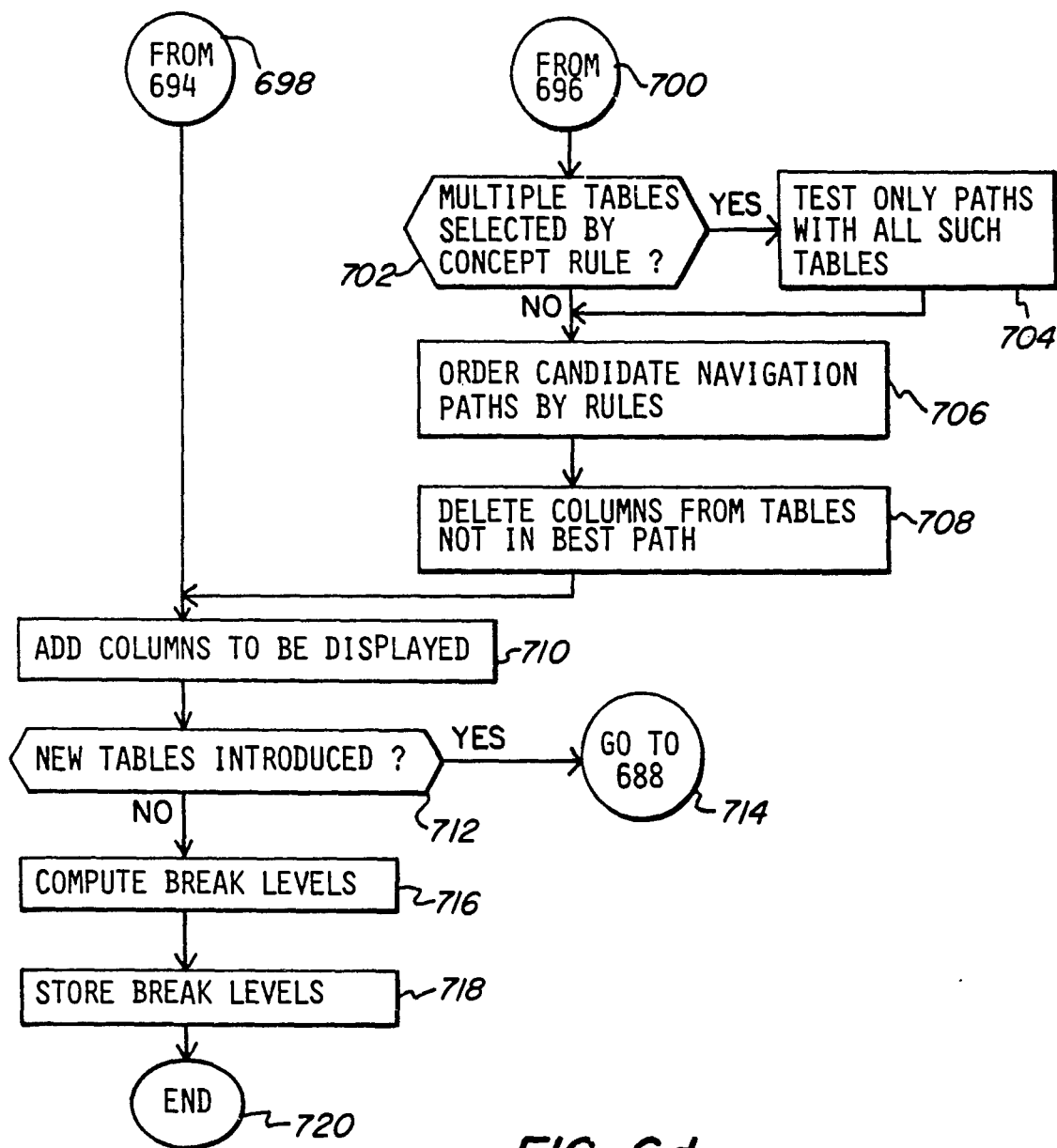


FIG. 6d

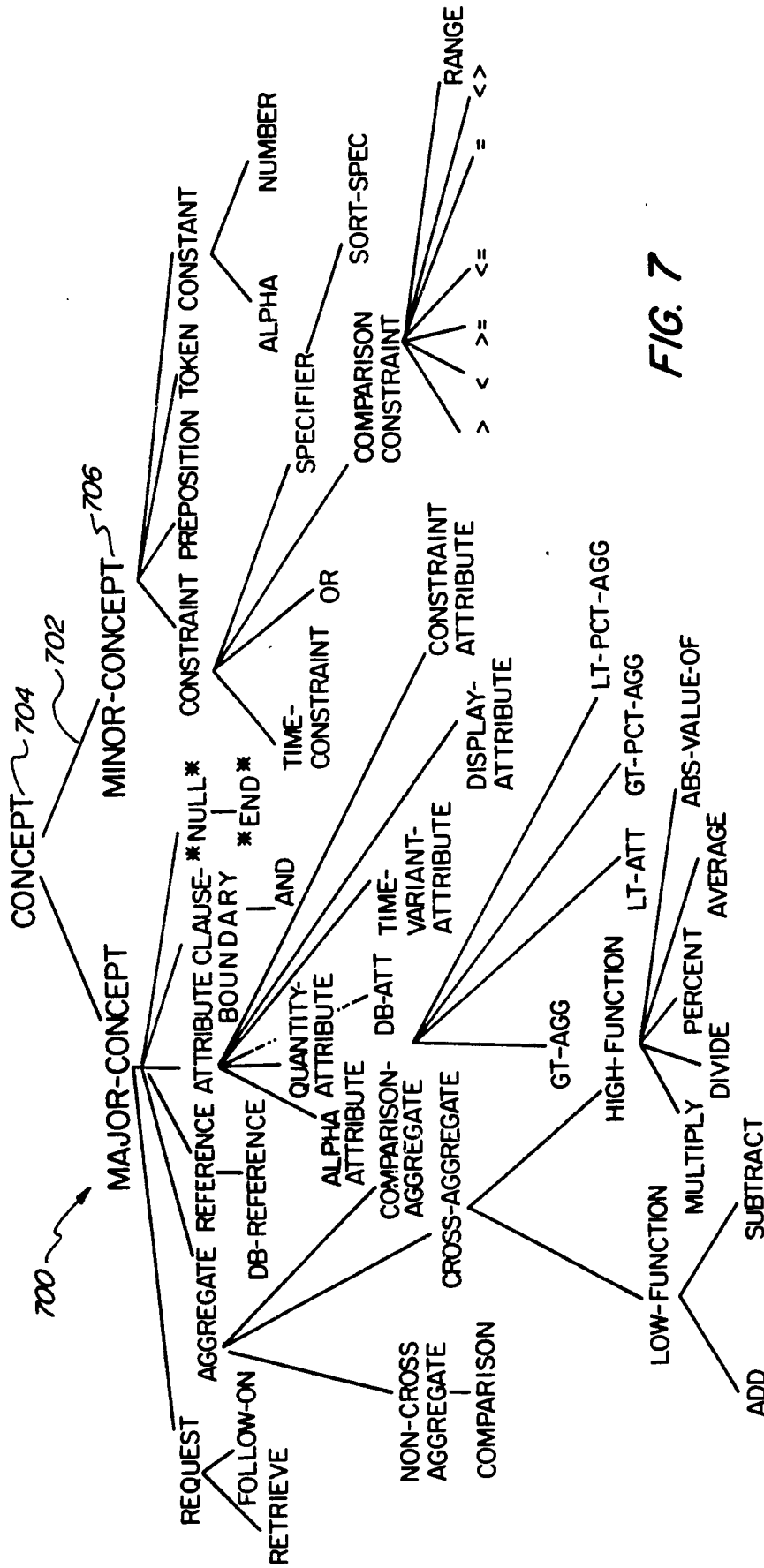


FIG. 7

DATABASE RETRIEVAL SYSTEM HAVING A NATURAL LANGUAGE INTERFACE

This application includes a microfiche appendix, having 47 fiche with a total of 11,603 frames.

BACKGROUND OF THE INVENTION

The present invention relates to a database retrieval system, and more particularly to such a system having a natural language interface.

Business managers and staff require information to run their companies. Data processing departments of companies have been attempting to meet this information need since the early 1950's. The record keeping of most organizations is now computerized, and an abundance of data of all kinds, often describing transactions in minute detail, resides on the central computers of these organizations. In theory, all this data is available for review by employees of such companies. In practice, however, users of such information have faced serious obstacles in retrieving the information they need.

A frequent response to a user's request for data from a database is that the data is not stored in a way that enables it to be used to meet a user's need. Additionally, the complexity of current database systems requires a trained specialist to figure out how the data requested by a user can be retrieved from the database. This specialist must interpret the user's request or "query", determine exactly what it is the user is looking for, and figure out how to get that information from the database. Then, once the data is retrieved, it must be formatted into a report that the user can use and understand.

In recent years, a type of database known as a "relational database" has come into widespread use within the business community. An "entity-relationship" model is often used when mapping a real world system to a relational database management system. The entity-relationship model characterizes all elements of a system as either an entity (e.g., a person, place, or thing) or a relationship between entities. Both constructs are represented by the same structure, referred to as a "table".

A table is a collection of data organized into rows and columns, and represents a unit of a relational database. In an order-entry system, for example, entities will include parts and orders. Such information may be represented in two different tables. The relationship of which parts are requested by an order may be represented by a third table.

Thus, in applying the entity-relationship model, the entities of a system are identified and tables are constructed to represent entities. Then, relationships between the entities are identified and the current tables are extended (or new tables created) to represent these relationships. Finally, the attributes of each entity are identified and the tables are extended to include such attributes. Those skilled in the art are well familiar with the application of the entity-relationship model to relational database management systems.

In recent years, there have been proposals for providing a natural language interface to relational databases. An English language interface, for example, would enable unskilled users of a database to query the database for desired information, and receive such information without the need to rely on a trained specialist to interpret the query, access the database, generate a

report, and communicate the report to the end user. Thus, a natural language interface would save enormous time and money for companies using relational databases, and would enable users with little or no computer experience to use a sophisticated database system by merely inputting (e.g., via a keyboard) a natural language (e.g., English) question.

An example of a natural language interface proposed in the past can be found in the article entitled *Natural Language Interfaces: Benefits, Requirements, State of the Art and Applications*, by John L. Manferdelli, *A.I. East*, October, 1987. This article describes a system in which an English sentence is converted into a grammatical structure ("parsed"), much like a sentence diagram. The diagrammed sentence is then translated into a "representation language" that is a hybrid of a semantic network and first order predicate logic. The representation represents time dependent facts, quantified statements, tense information and general sets, and is based on concepts contained in the original English sentence.

The representation language provided by the prior art system referenced above is complex, and not easily understandable even to a skilled user of the system. Thus, it is difficult for such a system to be implemented as a general purpose interface for any application database that might be desired. Customization of the interface to specific application databases was difficult and time consuming, and no means were provided for enabling a skilled user to easily comprehend the representation language produced by the natural language interface for a given query. Without such means, the building and testing of an interface for a particular application is extremely difficult and costly.

Various other articles have been published concerning software that is currently available to enable a natural language, such as English, to be translated into a representation language that can be used by a computer system to respond to a natural language query. For example, a program known as "McELI" is available for this purpose and discussed in *Inside Computer Understanding*. Schank and Riesbeck, Erlbaum Press, 1981. Another program known as "LIFER" is described in the article *LIFER: A Natural Language Interface Facility*, by Gary G. Hendrix, *SIGART Newsletter*. Issue 61, 1977, pp. 25-26. Each of these programs will translate a natural language into another formal syntax, such as a representation language. However, to date the representation language syntaxes have been complicated and difficult to understand. Therefore, no means have been available to enable anyone but the most sophisticated computer programmers to utilize such languages in providing a natural language interface capability to desired applications, such as the retrieval of information from a database.

A particular problem in providing a natural language interface for a database resides in enabling the system to locate data responsive to a natural language query regardless of the words used in the original query. A primary objection of end users of most prior art database retrieval systems is that they have to learn the names of the database elements, i.e., if the term "salary" is used in the database, the end user would have to use the same term in order to retrieve salary information, and could not use synonyms such as "wage", "earns", "makes", or "pay". This problem is referred to as the "synonym problem".

Some products have attempted to solve this problem by having the system programmers define all of the

synonyms that can be thought of for each database element, and to program these synonyms into the system. Such a requirement makes the setup procedure of a natural language interface extremely cumbersome, and often impractical.

Another problem with providing a natural language interface for database retrieval stems from the fact that the end user does not know where desired information resides in the database. For example, some information would have to be retrieved from detail-level columns in the database, whereas other data would have to come from summary-level columns. The choice of which column(s) to use must be made by the system, since the end user is unable to specify the data location. This problem is referred to as the "data location problem".

The assignee of the present application has marketed a product in the past which attempted to resolve the data location and synonym problems. That product included a built-in database expert system containing rules to resolve a many-to-one relationship between words/phrases and concepts, and also to resolve one-to-many relationships between concepts represented in a natural language query and database columns. For example, words such as sales, sell, bought, purchases, and revenues contained in a query would be mapped to a concept known as "sales". Then, the concept "sales" would be mapped to the various columns of a specific database containing sales information. The specific product involved was a turnkey wholesale distribution application that provided a natural language interface to a specific database. The natural language interface was custom designed for the specific database, and was not database independent. The system did not provide means to enable a skilled user thereof to tailor the interface for any other database. The representation language provided by the natural language interface was not easily understandable to a skilled user. Thus, it will be appreciated that the prior system was not a general purpose database retrieval system.

It would be advantageous to provide a truly general purpose natural language interface for database retrieval, allowing skilled users (who are not experts in artificial intelligence computer theory and application) to easily custom tailor the interface to a specific application database. Such a system should solve both the data location problem and the synonym problem inherent in prior art natural language interfaces.

It would be further advantageous for such a system to generate a representation language, or "meaning representation" that is easily understandable, database independent, and canonical (i.e., two different queries having the same meaning must have the same final meaning representation, and two queries having different meanings must have different final meaning representations). Such a meaning representation should capture, at a conceptual level, the information requirement expressed in the natural language query.

It would be further advantageous to provide such a system in which a skilled user or "developer" builds a knowledge base, pertaining specifically to an application database, that enables the system to efficiently and economically retrieve and report data that is a proper response to a natural language query entered by an unskilled user. Such a system should interpret the query, use the knowledge captured in its database expert system to locate the relevant data tables and columns from a database, and then transparently generate the most efficient code (e.g., structured query lan-

guage—"SQL") to produce a report instantly. No knowledge of SQL, database field names, or other technical jargon should be required of the end user.

The present invention provides such a database retrieval system and method for retrieving data from a database.

SUMMARY OF THE INVENTION

In accordance with the present invention, a database retrieval system having a natural language interface is provided. The system comprises a computer processor, and a natural language interface coupled to the computer processor. Tool kit means are also provided to enable a database developer to create a knowledge base containing a structural description and a semantic description of a database from which data is to be retrieved. First means operatively associated with the computer processor, produces a database-independent, canonical, internal meaning representation of a natural language query entered into the natural language interface. Second means, operatively associated with the computer processor, identifies database elements that are necessary to satisfy the query represented by the meaning representation. Third means, operatively associated with the computer processor, generates a database query among database elements identified by the second means, to enable the retrieval and aggregation of data from a database to satisfy the natural language query. Debugging means derive an easily understandable representation from the internal meaning representation. The external meaning representation enables the database developer to comprehend the internal meaning representation, and verify that a natural language query entered into the natural language interface is properly interpreted to effect the correct retrieval and aggregation of data from the database.

The meaning representation comprises entities and constraints relating to the entities, without reference to factual or linguistic relationships between entities that would prevent the meaning representation from being easily understood to a system developer.

The second means of the database retrieval system comprises an expert system coupled to access structural and semantic description information in the knowledge base, and identifies the database elements from the structural and semantic description information in accordance with predefined rules. The rules comprise steps for identifying an optimal set of database elements to satisfy the query represented by the meaning representation.

The structure of a database used in connection with the system of the present invention may be columnar, and the semantic description information can comprise a concept index of database columns. The semantic description can further comprise the time frame, value unit of measure, and aggregation level of database columns.

Means, operatively associated with the computer processor, can be provided for generating a formatted report containing data responsive to a natural language query. The debugging means can be used by a database developer to view the external meaning representation. A developer can also view a representation of the database elements identified by the second means. The debugging means can further enable the database developer to view the database query generated by the third means.

In order to identify an optimal set of database elements, the system can locate initial indexed columns, test subtypes, test data class-to-table rules, match characteristics and constraints, match time constraints, choose master table quantifiers, test for detail file columns, test for summary file preference, eliminate foreign-key-only tables, minimize tables, and then select the optimal navigation path through the database for satisfying a query. Data retrieved from a database in response to a natural language query can be displayed on a user's workstation, or printed for later reference.

In building the knowledge base, the database developer is able to enter join criteria, column semantics, data group definitions, and word and phrase associations into the knowledge base. The database developer can also build and modify the knowledge base by adding, deleting, and modifying subtypes; refining column references; adding, deleting, and modifying word-to-data class rules; adding, deleting, and modifying data class-to-table rules; and adding, deleting, and modifying nominal data definitions. Much other information can also be entered into the knowledge base and manipulated by the database developer.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of the system of the present invention;

FIG. 2 is a diagrammatic illustration of a relational database;

FIG. 3 is a flowchart depicting the translation of a natural language query to an internal meaning representation and an external meaning representation which is database independent, canonical, and easily understandable to a system developer;

FIGS. 4a and 4b comprise a flowchart of the steps a system developer takes to create and maintain a knowledge base in accordance with the present invention;

FIG. 5 is an entity-relationship diagram for a knowledge base created in accordance with the present invention;

FIGS. 6a, 6b, 6c, and 6d comprise a flowchart of the column selection process used by the system of the present invention to identify an optimal set of database elements necessary to satisfy a query; and

FIG. 7 is a semantic network diagram of the various concepts which can be included in an internal meaning representation.

DETAILED DESCRIPTION OF THE INVENTION

The present invention provides a database independent natural language interface for information retrieval. Unlike prior art systems, the present system can deal with large databases having complex semantics. For example, an item such as year-to-date dollars may be labelled in several tables of a complex database. The system of the present invention determines exactly where to get data in response to a specific request. The data may come, for example, from a table that contains summary-level or detail-level values. The system includes two key components; namely, a developer tool kit and a query system. The developer tool kit enables a system developer to build, test, and maintain a knowledge base containing information about an application database that the system will be used to query. The developer does not need to have any expertise in the computer science field of artificial intelligence, since the system produces external meaning representations that

are easily understandable to the developer without such knowledge.

The query system allows users with little or no computer experience to enter a conversational English (or other natural language) query. A natural language interface interprets the query and reduces it into an internal meaning representation used by the system, and the external meaning representation that is easily understandable to the developer.

The system also includes a context expert system that fills in the implicit meanings of a query. Then, the data responsive to the query is located using a database expert system that enables retrieval of the data from proper tables and columns in the database. The database expert system is essentially an artificial intelligence engine that understands the database through the knowledge base set up by the developer, and through this understanding is able to find things in the database.

Turning now to FIG. 1, the system of the present invention is depicted in block diagram form. A user, which can be either a developer (a high level user with some computer experience) or an end user (typically a manager or administrator with little or no computer experience) accesses the system through a workstation 10. Any number of workstations 10 can be provided to enable various system developers and end users to interact with the system.

A computer processor 12, coupled to workstation 10, controls the overall operation of the system. The elements generally designated 14 in FIG. 1 comprise the developer tool kit, which is used by system developers to communicate their knowledge of application databases to the system. It is important to recognize that the overall system of the present invention is database independent, and can be used with any application database once a developer builds a knowledge base containing information about the application database.

The elements generally designated 16 in FIG. 1 comprise the query system which is used to process a natural language query input by an end user, and to extract relevant information in response to the query.

A system developer accesses the developer tool kit 14 through a series of windows and menus displayed on workstation 10. In building a knowledge base, the system developer goes through a series of steps which are described in detail below in connection with FIGS. 4a and 4b. Generally, the steps taken by a developer include setting up a profile of the end user(s) as indicated at box 18 in FIG. 1, running a data dictionary analyzer 20, editing the knowledge base through editor 22, running and formatting reports through knowledge base reporter 24, running a nominal data analyzer 26, and debugging the knowledge base using debugger 28.

Data dictionary analyzer 20 automatically reads the relational database management system ("DBMS") catalog for the application database to learn all about the database structures such as tables, fields (i.e., "columns"), and data formats. Knowledge base editor 22 is used by the developer to give the system an understanding of the semantics, or meaning, of the data. For example, knowledge base editor 22 is used to provide the knowledge base with information as to how the DBMS tables are related to one another, and to define whether columns in the tables contain summary-level or detail-level data. Time and other attributes (i.e., information) for the columns is also entered through the knowledge base editor.

The knowledge base reports function 24 provides the developer with detailed information in printed form about the structural and conceptual data contained in the knowledge base. Nominal data analyzer 26 automatically reads nominal data values and sets up necessary definitions to enable queries to reference these data values by name. The term "nominal data" refers to columns in the application database that contain data whose values are names of things, such as products or customers.

The debugger 28 provided in accordance with the present invention is particularly powerful, due to the fact that it provides an external meaning representation for a query in an easy to understand form, and enables the developer, who typically has no knowledge of artificial intelligence principles, to follow the processing of the query by the query system at any point from the generation of the internal meaning representation, through the generation of the structured query language ("SQL") or other code that is ultimately produced by the query system to retrieve information from the database. The external meaning representation is provided to enable the developer to comprehend the interpretation of a query by the natural language interface.

Once a developer has entered the information necessary to build a knowledge base, the information is compiled to provide a run-time knowledge base 30 used by the query system to determine what data to access in response to a natural language query. One or more developers can build knowledge bases for different application databases 32 that can be accessed by the system.

The query system elements 16 depicted in FIG. 1 include a natural language interface 34, a database expert system ("DBES") 36, a navigator and query language generator 38, a reporter and database access system 40, and a context expert system 42. Natural language interface 34 comprises a natural language parser of a type known in the art, such as that available from Cognitive Systems, Inc. of New Haven, Conn., U.S.A.

The output of natural language interface 34 is used to produce an internal meaning representation that is used by the database expert system, but is not easily understandable. Thus, the internal meaning representation would be of little or no use to a developer.

Subsequently, and in accordance with the present invention, the internal meaning representation is transformed into an external meaning representation which is easily understandable to a developer. This process is depicted in the block diagram of FIG. 3. The external meaning representation enables the developer to debug and validate that portion of the knowledge base that is used by the natural language interface in connection with its query interpretation function.

As shown in FIG. 3, an end user inputs a natural language query 300 to natural language interface 34. Software ("code") 302 is provided for use by natural language interface 34 to enable the production of the internal meaning representation 304. Code 302 is based on the grammar for the internal meaning representation. The grammar for the internal meaning representation of a preferred embodiment of the present invention is set forth in Appendix A hereto. The notation of the grammar in Appendix A is in a standard form well known to those skilled in the art, and will enable a skilled programmer to generate code 302. A semantic network diagram 700 of various concepts that can be included in

the internal meaning representation is provided in FIG. 7. Each line in the network diagram represents an "ISA" link. For example ISA link 702 specifies that minor-concept 706 is an attribute of concept 704.

The internal meaning representation 304 is converted, in accordance with the present invention, to an easily understood external meaning representation 308 by debugger 28 in the developer tool kit. The external meaning representation is database independent, canonical, and easily understandable. The feature of understandability enables a developer to comprehend how the natural language interface 34 has interpreted the query. With this comprehension, the developer can refine the knowledge base, as necessary, to ensure that the interpretation of a query by the natural language interface will be proper and that as a result, information retrieved by a query will properly reflect the intent of the end user.

Debugger 28 derives the external meaning representation from the internal meaning representation by finding all the entities in the internal meaning representation and placing them to the left of a colon. All of the constraints associated with the entities in the internal meaning representation are then located, and placed to the right of the colon, to form the external meaning representation. Thus, in a preferred embodiment the external meaning representation takes the form:

ENTITY: CONSTRAINT

In the external meaning representation, the hierarchy of the constraints in the internal meaning representation is ignored, as this information is not pertinent to the developer, and its inclusion would defeat the desired characteristic of easy understandability. Factual (i.e., "real world") and linguistic relationships between entities in the internal meaning representation, that would prevent the external meaning representation from being easily understood, are also ignored.

An example of an internal meaning representation, which would be produced by natural language interface 34 in response to the query "show total year-to-date sales dollars by customer", is as follows:

```
(Retrieve Reference (DB-REFERENCE
Constrained-Slots
((SALES st ((YTD
(DOLLAR)
(TOTAL)))
(CUSTOMER st ((SORT Number (1))))))
```

The same query will produce the following external meaning representation:

```
SALES:      YTD, DOLLAR, TOTAL
CUSTOMER:   SORT (1)
```

As will be appreciated, the internal meaning representation is not easily understandable, and would not enable a developer to comprehend how the natural language interface has interpreted a query. The external meaning representation, however, is quite clear, and will enable a developer to assess the accuracy of the natural language interface's query interpretation.

In accordance with the present invention, an external meaning representation can comprise several components; namely, display concepts, selection criteria, formatting directions, semantic modifiers, and time concepts. Examples of each of these, along with the vocab-

ulary that produces them, are set forth in Appendix B hereto.

Turning again to FIG. 1, the internal meaning representation output from natural language interface 34 is an explicit meaning representation of the query. In the explicit meaning representation, ambiguities have been eliminated according to disambiguation rules, but inferences that fill in implicit meaning (e.g., ellipses) have not been made. Therefore, a context expert system 42 is provided to fill in ellipses and produce a final, implicit meaning representation that is then passed to database expert system 36. The context expert system 42 would be used, for example, for a query that concerns customers (e.g., "show customers") and then states "with address" without explicitly indicating that the address desired is that of each customer. The context expert system will refer back to the customer concept, and infer that the addresses desired are those of the customers.

By accessing semantic and structural information pertaining to an application database and residing in knowledge base 30, DBES 36 provides a retrieval specification that lists the tables and columns chosen, in accordance with column selection rules, for the retrieval of information from the application database in response to a query represented by the meaning representation. A navigator and query language generator 38 is used to define optimal navigation paths through the database tables and columns to respond to the query, and to generate a meta-query language ("MQL"). The meta-query language is used by a reporter and database access system 40 to generate the code (e.g., structured query language ("SQL") code) to actually retrieve the information from the application database. Reporter 40 also generates the reports which are displayed on the end user's workstation 10, or printed out for future reference.

FIG. 2 depicts the structure of a typical application database 32. Such a database includes various tables 52, 54, 56, each of which is a data structure holding data in the database. An example of such a table in greater detail is generally designated by reference numeral 58. As shown, table 58 includes a plurality of rows 66, 68, 70 and columns 60, 62, 64. In a typical sales database, a table designated "SALES" might contain columns for customer numbers, product numbers, year-to-date dollar sales, etc. Another table, designated "SALESMAN" might contain columns for salesman numbers, salesman names, and year-to-date dollar sales for each salesman. Similarly, there may be additional tables for "products", "customers", and more detailed information including "line items" and "orders". Each row in the table defines data for a different entity, such as different customers, different salesmen, different products, etc. By accessing a specific row and column, a particular piece of data can be retrieved.

Indexes can be provided in a relational database to provide quick access to rows in a table, and to enforce the uniqueness of rows within a table. Unique indexes enforce the requirement that rows not be duplicated within a table.

A "primary key" of a table is used to uniquely identify each row in the table and is comprised of one or more columns of a table. The value of a primary key uniquely identifies one row of a table. Therefore, for every primary key value there is exactly one row, and for every row there is exactly one primary key value.

Relationships between entities are often represented through primary keys.

"Foreign keys" represent relationships between tables. Such keys consist of a column or group of columns whose values are derived from the primary key of a table, which can be another table or the same table in which the foreign key resides. The existence of a foreign key implies that the table with the foreign key is related to the primary key table from which the foreign key is derived.

Other characteristics of relational databases are well known to those skilled in the art.

In order to enable the system of the present invention to retrieve data from an application database, the database must first be defined by a developer, and the definition stored in the knowledge base. A knowledge base built by a developer is subdivided into related groups of tables, and each of these groups is also called a knowledge base. These knowledge bases include a structural knowledge base, a semantic knowledge base, a report knowledge base, a system knowledge base, and a utility knowledge base. The structural knowledge base contains structural role information about the application databases. Such information defines database nodes, databases, tables, columns, keys and domains, and indexes.

The semantic knowledge base contains dictionary information used by the language processing facility, and conceptual information about application databases. This knowledge base includes dictionary entries, concepts (i.e., "data classes"), specifically "data groups" and "data categories"), calendars, column entries, structural element attributes (column "attributes" or "information", specifically report information, column descriptions, and characteristics including semantic data type, aggregation, and time specification), rules (including data class-to-table rules and word-to-data class rules) and nominal data information.

The report knowledge base contains information that facilitates report generation, such as saved requests for reports.

The system knowledge base includes security and configuration information, as well as logs of system operations. The utility knowledge base includes utility structures used by the system's internal knowledge base access system.

Various types of integrity rules apply to the knowledge base, including relation integrity rules that are referential integrity rules and special integrity rules. Referential integrity deals with the prevention of inconsistencies that may occur in the knowledge base because of foreign keys. Special integrity involves rules and procedures that must be observed to maintain consistencies between columns of the same or different tables when these consistencies cannot be expressed by referential integrity rules. Integrity rules that fall outside the category of relational integrity include domain integrity rules, rules specifying that the values in a set of columns are unique, and rules regarding the number of rows in a table.

The steps taken by a developer in building a knowledge base using the developer tool kit are shown in the flowchart of FIGS. 4a and 4b. It is noted that although the developer tool kit is depicted in terms of a flowchart for purposes of explanation, the preferred embodiment provides the developer with access to the tool kit through the use of windows and menus. Therefore, the flowchart is not meant to imply that use of the tool kit

is in any way dictated by the order of functions as they appear in the flowchart.

After entering the developer tool kit at box 400, a developer can run the data dictionary analyzer as indicated at box 402. The data dictionary analyzer analyzes the DBMS data dictionary to determine the structure of the data in the database by extracting table, column, and index information. After the system has completed this analysis, and stored the relevant database structural information, the developer is given an opportunity to define join criteria at box 404.

A join is the mechanism by which the relationship that exists between two or more tables is defined. A join is defined when a key in one table and a key in another table are given the same join name. As noted above, a key is a column or set of columns in a table by which the table is related to one or more other tables. The data contained in the key column(s) in one of the related tables is the same as the data contained in the key column(s) in each of the other related tables. One or more keys can be defined in any table, and can be composed of one or more columns.

To join two tables, the developer defines a primary key in one of the tables and one or more foreign keys in the other table, and specifies the same join name for each key. To join more than two tables, a primary key is defined in one of the tables, and one or more foreign keys are defined in each of the other tables. Then, the same join name is specified for each key. The primary key and all of the foreign keys that have the same join name belong to the same join. The relationship between each foreign key and its related primary key must also be defined by the developer. The relationship can be either one-to-one or one-to-many. If a foreign key can have the same value in two or more rows, the relationship is one-to-many. Otherwise, the relationship is one-to-one.

In defining join criteria, the developer can add a new join, delete a join, specify additional foreign keys, or remove existing foreign keys. After all the join criteria are specified, the developer is given the opportunity at box 406 to modify column characteristics.

The data dictionary analyzer (box 402) creates a column characteristics definition for each column that it analyzes. This definition becomes column reference entry number one for the column. The data dictionary analyzer also assigns the fully qualified column name to the column description field of the column characteristics definition. The column description becomes a direct column reference, such that if the column description is used in a query, the system retrieves data from that column for the report it displays in response to the query. A developer can delete from the knowledge base information about those columns that will never be selected when a query is being processed and will never be displayed on a report.

Column descriptions are used by the present system in two ways. First, if the word or phrase specified in the column description is used in a query, the column is considered for selection. Second, the column description specifies what is included in the query paraphrase if the column is selected. A query paraphrase is a reformulation of the query, which may be displayed to the system user, in order to verify that the system properly understands the query as originally input.

When a query is submitted, it is analyzed and reduced to an internal meaning representation and an external meaning representation. In the first step of the column

selection process, each column that is referenced by an entity in the external meaning representation is identified. In this way, one or more columns can be identified as candidates for selection. In the second step of the column selection process, the constraints present in the query are used by matching them with the column characteristics defined by a developer, in order to eliminate some of the columns that have been identified. This refinement or optimization process utilizes semantic data type, aggregation, and time information input to the knowledge base by the developer. The columns that are not eliminated are then selected to compose the report that the system ultimately produces in response to a query. At step 406, the developer is permitted to modify the column characteristics definition, but may not delete it.

After the modification of column characteristics, the developer can define data groups as indicated at box 408. In simple applications, column descriptions specified in column characteristics definitions can be used to find the column or columns that contain the data that must be retrieved in response to a query. For complex applications, further definition is required to enable the system to locate the data. This is accomplished by grouping the columns of an application into data groups. After data groups have been defined, words and phrases can be associated with the data groups, as indicated at box 410. When an end user includes one of the associated words or phrases in a query, the system considers the columns in the data group for selection. A developer can add or delete data groups, and add or delete one or more columns to a previously defined data group. The developer can associate words or phrases with data groups and data categories.

At box 412, the developer establishes user profiles for each end user. Such profiles are used when an end user logs into the system to determine which values have been assigned to certain parameters and whether certain operations are authorized for the user.

At box 414, the developer defines database manager ("DBM") authorization names to make it possible for a user for whom a user profile is added to access the proper tables in the application database.

At box 416, the developer can establish the system profile to specify information such as the type of automatic logging that is to take place and the type of information that is to be logged, whether error logging and/or query trapping is to be enabled, and to define certain defaults such as time type, current fiscal year, and current period. The concept of query trapping is discussed below in connection with box 448 of FIG. 4b.

At box 418, the developer can add, delete, and modify columns without re-running the data dictionary analyzer. Similarly, tables may be deleted at box 420 without re-running the data dictionary analyzer. Deleting a table from the knowledge base automatically deletes all keys, indexes, table characteristics definitions, columns, column characteristics definitions, and column reference definitions that are associated with the table.

At box 422, application databases may be deleted without running the data dictionary analyzer. At box 424, nodes can be deleted from the knowledge base without running the data dictionary analyzer. Deleting a node automatically deletes all of the databases, tables, keys, indexes, table characteristics definitions, columns, column characteristics definitions, and column reference definitions associated with the node.

At box 426, the developer can add, delete, and modify indices without running the data dictionary analyzer.

Subtype maintenance is provided at box 428. A subtype (sometimes referred to as a "restriction concept" or "modifier") limits or qualifies the meaning of a data group or data category. Subtypes are used in column reference definitions and the data class-to-table rules to control the column selection process. Before a subtype can be used in a column reference or a data class-to-table rule, it must be defined. A subtype definition contains an arbitrary name for the subtype and the words or phrases that are associated with it.

At box 430, the developer is provided with an opportunity to refine column references. Column references (or "column-choice entries") give the system additional information it needs to determine which columns of data to retrieve in response to a query. A column reference provides a developer with a means of refining the column selection criteria. For example, a developer could specify that one column is to be preferred if the end user uses words or phrases that refer to the subtype "open", and another column is to be preferred if the end user uses words or phrases that refer to the subtype "closed".

In analyzing a query, the system of the present invention separates the query into three components; namely, data classes (data groups and data categories), column descriptions, and direct column references, each of which appear as entities in the external meaning representation. Identification of the three components listed is accomplished by analyzing the words and phrases in the query. A data group or data category is identified if either a term that has been associated with the data group or data category is used in the query, or a word or phrase that is used to define an application term is used in the query. A column description is identified if either the column description specified in a column characteristics definition is present in the query, or an application term that refers to the column description is used in the query. A direct column reference is identified by a column description, or a word or phrase (not a data class) that has been specified in the reference field of a column reference. In accordance with the present invention, the data groups, data categories, column descriptions, and direct column references can be viewed by the developer by requesting a final conceptual query representation when in the debug mode of the developer tool kit.

When column descriptions are found in a query, no additional information is necessary for the system to determine which columns to select. However, for data classes, additional information is required, and is entered at box 430 through the column reference refinement procedure. It will be apparent that a data group or data category refers to a column if either the column can be referred to by combining a word that is associated with the data group or data category with a data specification, a unit of measure, a level of computation, or a time-related value; or, the column is one in a set of columns that should be retrieved in response to a query such as "show customers" (in which "customers" is a data group or data category). Multiple column references can be defined for a column, and each reference provides a different means by which the column can be selected. For example, a column can be selected by a column description, direct column reference, data group, or data category. Thus, at step 430, a developer

can add, modify, or delete a reference to a column by a data group; add, modify, or delete a reference to a column by a data category; and add, modify, or delete a reference to a column by a word or phrase.

After refining column references, the flowchart proceeds to box 436 of FIG. 4b via boxes 432 and 434. At box 436, a developer defines table characteristics for each of the tables contained in the application database. Table characteristics definitions are used to specify the columns in the table that are always displayed in reports, whenever any column in the table is chosen by the system during the processing of a query submitted by an end user. Table characteristics definitions also specify calendar information and time information. Table characteristics definitions can be added, modified, and deleted by the developer.

At box 438, the developer specifies application terms which are words or phrases that are defined in terms of a word or phrase already existing in the application dictionary. When an application term is used alone, it is exactly equivalent to the word or phrase that defines it. However, if an application term is used as part of a phrase (or, if the term is a phrase, as part of a larger phrase), it is not necessarily equivalent to the word or phrase that defines it. For example, if a developer adds the term CUST and gives it the definition CUSTOMER, then the term CUST can be substituted for the word CUSTOMER in a query and the result will be the same. However, if the term BEST CUSTOMER has also been defined, the phrase BEST CUST may not be substituted for the term BEST CUSTOMER with the same result. To define two terms that have the same definition, the same full definition can be specified for both terms, or the full definition may be specified for one term and then that term may be specified as the definition of the other term.

Abbreviations are specified by the developer at box 440. An abbreviation is a word or phrase that is equivalent to another word or phrase. The equivalence exists whether the abbreviation is used alone or as part of a phrase. Thus, in the example above, if CUST is defined as an abbreviation for CUSTOMER, then BEST CUST can be substituted for BEST CUSTOMER with the same result.

At box 442, the developer may define word-to-data class rules (sometimes referred to as "word-to-concept rules"). These rules are used by the system when a word or phrase that has been used in a query points to two or more data classes. Only one word-to-data class rule can be defined for any word or phrase.

An example of an instance where a word-to-data class rule would be necessary is where end users may formulate several different queries that include the word "owe". Some may be concerned about how much money their company owes one of its vendors, others may be concerned about how much money their customers owe. Examples of such queries might include the following:

1. Does anyone owe us money?
2. Who do we owe money?
3. What is owed?

In the first query, which is concerned with money owed by customers, the word "owe" is followed by the word "us". In the second query, which is concerned with money owed to vendors, the word "owe" is preceded by the word "we". In the third query, it is not clear whether the word "owed" relates to "owing" or to "being owed".

In this situation, the developer will create a word-to-data class rule for the word "owe" that takes into account all of the above possible queries. The following rule is an example:

```
IF WORD-FOLLOWS (ME, US)
  THEN CUST_BALANCE
ELSE IF WORD-PRECEDES (I, WE)
  THEN VEND_BALANCE
ELSE QUERY-USER ("OWED BY CUSTOMERS",
  "OWED TO VENDORS", CUST_BALANCE,
  VEND_BALANCE)
```

Using this rule, the system uses the data class CUST_BALANCE for the first query and uses the data class VEND_BALANCE for the second query. For the third query, the user is asked to specify either "owed by customers" or "owed to vendors". If the user specifies "owed by customers", the data class CUST_BALANCE is used. If the user specifies "owed to vendors", the data class VEND_BALANCE is used.

In addition to defining word-to-data class rules as described above, a developer can also define data class-to-table rules (sometimes referred to as "concept-to-table rules") as indicated at box 444 of FIG. 4b. A data class-to-table rule is necessary when a data class refers to columns in more than one table. Such a situation is created by specifying the same data class when column references for columns are defined in two different tables. However, it is not necessary to define a data class-to-table rule if the two tables have been joined. Only one data class-to-table rule can be defined for any data class.

For example, if a data class SALES has been defined that points to both the CUSTOMER table (for month-to-date sales) and the SALES_HIST table (for year-to-date sales), the system needs a data class-to-table rule to be able to handle a query such as: "show sales". In this case, the developer must decide whether the user is more likely to want month-to-date or year-to-date information in response to such a query. If the most likely situation (the default) requires year-to-date information, the following data class-to-table rule permits the query to be handled:

```
DATA CLASS=SALES;
IF WORD-PRESENT (MTD)
  THEN CUSTOMER
ELSE SALES_HIST;
```

At box 446, the developer may specify calendar definitions. Generally, the time related data in an application database relates to a normal annual calendar with quarters that start on January 1, April 1, July 1, and October 1. If a fiscal calendar is desired, or a calendar with a number of periods peculiar to a particular application is necessary, the appropriate information is specified by the developer.

At box 448, a scheme referred to as "row count maintenance" can be implemented. Some of the tables in an application database may contain a very large number of rows. A query submitted by an end user may cause a column or columns from such a table to be selected. Extracting data from such a column or columns to prepare a report in response to the query may tie up the system for quite some time, and be expensive. Thus, the developer can enable expensive query trapping. In this mode, the system estimates the number of rows that must be processed to answer each query and saved request. To do this, it must know approximately how many rows are contained in each table. A developer can

specify the approximate number of rows in each table by using the row count maintenance feature. Generally, this feature should be used before expensive query trapping is enabled and whenever the number of rows in one of the tables changes significantly.

At box 450, the system can be instructed to analyze nominal data, i.e., columns in the application database that contain data whose values are names of things, such as products or customers. The nominal data analyzer automatically reads nominal data values and sets up the necessary definitions so that queries can reference these data values by name. Once the nominal data analyzer has been run, any one of the words used in a description composed of several words will, if used in a query, cause information about the nominal data to be displayed.

At box 452, a query log and error log provided by the system can be maintained by a developer. At box 454, the developer can generate knowledge base reports. Such reports are useful to provide the developer with detailed information in printed form about the structural and conceptual data contained in the knowledge base.

At box 456, the developer can enter the debugger provided in the present system. Although part of the developer tool kit, the debugger is entered when the developer is using the query processor. As noted above, this feature enables the developer to view the processing of a query throughout each stage of the query processor. The debugger converts the internal meaning representation used by the query processor to an easily understandable meaning representation that enables the developer to comprehend the processing of a query. When in the debugging mode of operation, the developer can input test queries to the query processor, explore their interpretation by the query processor (via the external meaning representation, the MQL, and the SQL), and analyze the data reports generated by the system to determine if natural language queries retrieve the correct information.

The developer tool kit routine ends at box 458.

When a developer has completed the task of building a knowledge base, or has completed modifications to an existing knowledge base, the information is compiled. During the compilation, navigation paths and join column domains are generated. At the same time, the application dictionary, structural information, and indexed rules are checked for validity. After compilation, the developer can test the new knowledge base and if unsuccessful, modifications can be made by again entering the developer tool kit, and repeating the process until a satisfactory knowledge base has been completed.

FIG. 5 is an entity relationship diagram for a knowledge base constructed in accordance with the present invention. This diagram illustrates the mappings of words, concepts, columns, and tables within the database and the relationships between these entities and the attributes and rules defined by a developer. For example, word 502 maps onto abbreviation 500, expression 504, concept index 508 (which maps to concept 512), word-to-data class rule 506, and column attributes 520. Concept 512 maps to data class-to-table rule 510, table 548, nominal data definition 514, column entry attribute 518, and column attributes 520. The mappings of each of the other entities and other attributes is apparent from FIG. 5.

As noted in connection with FIG. 1, after the knowledge base 30 has been built for an application database 32, the query processor 16, and particularly database expert system 36, uses the information stored in the knowledge base to locate data from the application database 32 that is responsive to a query. In a preferred embodiment of the present invention, an initial set of candidate columns from the database is identified by the database expert system, and that candidate set is then narrowed down to identify an optimal set of database elements to satisfy the query. The column selection and optimization routine is shown in the flowchart of FIGS. 6a to 6d. The routine is entered at box 600, at which point every attribute referenced in the final meaning representation for the query is gathered and built into a table of the columns that should be used to resolve each attribute. Then, a sequential execution of the heuristic rules designated at boxes 602, 608, 618, 630, 640, 648, 656, 668, 674, 682, 688, 710, and 716 proceeds.

At box 602, an initial candidate set of columns is located. This candidate set has one entry for each basic concept (i.e., data group or data category) or direct column reference inside a query concept. Each entry is composed of the data group or data category, plus all column entries indexed by that concept or direct column reference. For every join column domain column in the candidate set, the candidate set is expanded to include any equivalent join column domain columns that may not have a column entry. This procedure is indicated at boxes 604 and 606. A "join column domain column" is a set of semantically equivalent columns derived from primary and foreign key information about an application database.

After the expansion of the initial candidate set is complete, control passes to box 608 where subtypes are tested. The testing procedure can proceed on a column by column basis throughout the initial candidate set, and as indicated at box 610, a determination is made for each column as to whether a restriction concept is specified. If so, control passes to box 612 where a determination is made as to whether the query contains a constraint on the concept or a direct reference to the column. If not, the column is deleted at box 614. Otherwise, control flows directly to box 616 where a determination is made as to whether the last column in the candidate set has been tested. If not, a loop continues until all columns have been tested. Thus, the rule for testing subtypes eliminates any columns from the candidate set for which (i) a restriction concept is specified for the column, and (ii) the restriction concept is not present as a constraint on the concept or a direct column reference in the concepts identified from the query.

The next rule applied tests data class-to-table rules, as indicated at box 618. These are database specific rules entered by the developer when the knowledge base is built, and they take priority over everything except the subtype rules dealt with at box 608. It is noted that the subtype rules are also defined by the developer in creating the knowledge base. At box 620, a determination is made as to whether a rule has been specified for the next data class present in the query. If so, the rule will return either a list of tables or a NIL. The list of tables are the preferred tables for the data class. If the candidate columns for the data class include one or more of these tables, all columns not from any of the tables are deleted. If a NIL is returned, it means that the rule does not apply to the current query and its result should not effect the candidate set. A concept without a data class-

to-table rule is treated as if its rule returned a NIL. Accordingly, if at box 620 a determination is made that a rule has been specified for the next data class tested, and the rule returns tables at box 622, then control flows to box 624 to determine whether the candidate columns include such tables. If so, all columns not from those tables are deleted at box 628. Box 626 determines if the last data class in the query has been tested, and if not, a loop continues until the testing is complete.

At box 630, characteristics and constraints are matched. The best candidate column(s) is selected for each data class in the candidate set according to the following procedure:

1. Each candidate column is scored by counting one point for each semantic descriptor of the column that matches one of the constraints or specifiers in the query concepts attached to the data class.
2. Then, the column with the highest score (or the set of columns that tie for the highest score) is found. If the concept has no specifiers or constraints, all indexed columns tie with the score of zero.
3. If there are more than one highest score columns, then, within a table, any that have been marked as default columns by the developer are preferred.
4. If any of the candidates have an AUTO descriptor, or a constraint to be output, or are a component of the primary key, then they are selected also.

This procedure is used to find the best semantic matches between a query and candidate columns. For example, if the query asks for year-to-date information, and such information is available directly from a column, then corresponding month-to-date columns for the information are eliminated.

Box 632 designates the determination as to whether several columns tie for the highest score. If so, then the preferred columns are selected at box 634. As noted above, preferred columns include those marked default, those with an AUTO descriptor, those with a constraint to be output, and those which are a component of the primary key. From box 632, control passes to box 640 via boxes 636 and 638. At box 640 columns are selected on the basis of the best time match. A perfect time match is preferred. Thus, for example, if year-to-date information is requested, and available directly from a column in the candidate set (e.g., YTD-SALES), this column is preferred to a partial match such as month-to-date sales. This determination is made at box 642. If a perfect match is found, columns without a perfect match are deleted at box 644.

If no perfect time match is found at box 642, then at box 646 columns are grouped, if possible, to make a perfect match. Those not used are deleted.

At box 648, master table quantifiers are tested. If quantification is present in the query, as determined at box 650, then two preferences are necessary. First, at box 652, master files are selected for concepts that index the primary key column of the master file. For concepts within the chain of quantification, or that are constrained but do not index the primary key column of a master file, it is preferred not to choose columns from the tables chosen at box 652. This is indicated at box 654.

The selection of master table quantifiers depicted at boxes 648 to 654 can be better understood by reference to several examples. If the query is "What customers have no sales this month?", then a master file for CUSTOMER (e.g., CUST.CUST#) is preferred and a table other than CUST is preferred for SALES.

If the query is "show salesmen all of whose sales were in Connecticut", then a master file for SALESMEN (e.g., SALESMEN.SLM#) is preferred, and a table other than SALESMEN is preferred for SALES and for STATE. Note that choosing STATE from the SALESMEN table would be an error (sales of all Connecticut salesmen) that would be difficult for a user to detect.

In another example, the query might be "what customers have bought every product this month?". A master file for CUSTOMER and for PRODUCT would be preferred, and a table other than either CUST or PROD would be preferred for SALES.

After the master table quantifiers are tested, the system tests for detail file preference at box 656. For each concept, detail file columns (i.e., a column from a table that has at least one column with a DATE data type or a data value meaning of DAY, WEEK, MONTH, QUARTER, PERIOD, or YEAR) are preferred if any detail constraint such as COUNT, AVERAGE, TOTAL, MINIMUM, MAXIMUM or EACH is attached to the data class in the query concepts. A determination as to whether a detail constraint is attached to a concept is made at box 658. If so, at box 660 the detail file column is chosen. However, columns from tables selected by any data class-to-table rule are not deleted, as indicated at box 662. Thus, for example, if the query states "show max, min, average invoice for March" and "show each sales for March", the test for detail file preference rule will select the SALES-DATE column from the sales transaction file over the MARCH-SALES column from a similar summary file. After the detail file preference testing, control is passed to box 668 via boxes 664 and 666.

At this point, the remaining candidate columns are tested for summary file preference. Summary file columns with an associated time frame are preferred to transaction file columns with a time data type. As indicated at box 670, a determination is made as to whether any candidate columns include time attributes. The time attribute may be either a relative or absolute time. If so, control passes to box 672 and the columns from tables with such columns are preferred over columns from tables without time attributes, unless the table was selected by a data class rule (i.e., at box 618 of FIG. 6a). The result of a summary file preference rule test is illustrated by the query "show year-to-date sales by product". The summary file preference rule will select the YTD-SALES-\$ column from the sales summary file over the SALES-DATE column from the sales transaction file.

The next heuristic rule results in the elimination of foreign-key-only tables as indicated at box 674. For each concept, if there are candidate columns for more than one table, and at least one of the candidate columns is a non-foreign key field, then any candidate columns from key-only tables are eliminated. These steps are indicated at boxes 676, 678, and 680 of FIG. 6c. This rule helps to minimize the number of selected tables by selecting columns from key-only tables only where there are no candidate columns from non-key-only tables. The rule is necessary to prevent errors during table minimization in the next rule (box 682). The application of the eliminate key-only tables rule is illustrated by the query "count salesmen and customers", where SALES_BY_CUST columns are eliminated from consideration and the preferred candidate set will be:

CUSTOMER - CUST.CUST#, SALES_BY_CUST.CUST#
SALESMAN - SLM.SLM#, SALES_BY_CUST.SLM#

The next rule, indicated at box 682, minimizes the remaining tables. For each table in the candidate set, a number of slots that contain at least one candidate column data structure from the table (among the candidate columns remaining in the candidate set) is counted. For each concept, if there are candidate columns for more than one table, the columns in the table with the highest count are chosen. These steps are indicated at boxes 684 and 686 of FIG. 6c.

After the tables are minimized, the best navigation path is chosen as indicated at box 688. In accordance with this rule, a determination is made at box 690 as to whether concepts with columns from multiple tables remain. If so, at box 692 the set of all possible combinations of tables is created. At box 702 of FIG. 6d, a determination is made as to whether multiple tables were selected by a concept (e.g., data class-to-table rule), and if so only navigation paths with all those tables are tested as indicated at box 704. Then, at box the candidate navigation paths are ordered in accordance with the following rules, which are successively applied until only one final path remains:

1. candidates for which an actual navigation path is found;
2. candidates with fewer tables in the navigation path;
3. candidates with more primary key columns in the navigation path (e.g., select SLM# from SLM Master rather than CUST Master);
4. candidates for which the total number of primary key columns in all tables of the path is fewest;
5. candidate paths for which the sum of the candidate column-join column domain levels (i.e., hierarchy levels) of the candidate columns is lowest; and
6. candidate paths for which the sum of the estimated number of rows in the tables is fewest.

Any candidate columns from tables not satisfying the rule set forth are deleted at each step in the above ordering. This step is indicated at box 708 of FIG. 6d. If, after applying all of these rules, more than one candidate path remains, any one of the remaining paths may be chosen arbitrarily.

At box 710, the columns to be displayed in response to the query are added. This is essentially a formatting routine so that the report produced will be in a form predefined by the database developer when the knowledge base is built. Interestingly, this step may introduce new tables for the navigator, and as a result, the navigator will have to be called once more to produce the final navigation path. Thus, at box 712 a determination is made as to whether new tables have been introduced, and if so, control returns back to box 688 as indicated at box 714.

Finally, the break levels for the columns are computed at box 716, and stored at box 718, to establish the ordering of the output of data responsive to the query. The query language generator 38 (FIG. 1) will make use of the break level data in producing the meta query language.

At box 720, the column selection routine ends. It is noted that throughout the column selection routine, rules are successively applied to columns to narrow down the candidate set to an optimal set, deleting unnecessary, redundant columns along the way.

It will now be appreciated that the present invention provides a versatile database retrieval system having a natural language interface. End users need only describe the information they want as the request would be described to another person. There is no need for a database manager to provide technical support, and the end user does not need technical expertise in query syntax or database structure and nomenclature. Access to data in response to a natural language query is immediate.

Using a concept known as "conceptual dependency", the system interprets questions posed in a natural language, such as conversational English. It infers not only what is asked, but what is needed to answer the query. Appropriate data is located in the database, efficiently retrieved, and an intelligible report is presented immediately. These results are accomplished largely through the meaning representation of the information sought. The meaning representation is provided in both an internal form for use by the system, and in an external form which is easily understandable to a developer. In addition to being easily understandable, the external meaning representation (like the internal meaning representation) is canonical and database independent. The provision of such an external meaning representation is a significant advance in the art, enabling a database developer with no experience in artificial intelligence principles to provide a knowledge base that successfully enables the proper data to be retrieved in response to a natural language query.

The system and method of the present invention also provide a sophisticated developer tool kit enabling a database developer to build an application specific knowledge base, as well as a set of heuristic rules for identifying an optimal set of database elements to satisfy a query.

The information in the knowledge base includes descriptions of all the columns and tables in the database, using the same symbols as the external meaning representation. In addition, rules for aggregating the data elements and matching them to the meaning representation are provided in order to generate an efficient report program. In generating the optimal retrieval program to find the data required to satisfy a query, the system captures the knowledge in the knowledge base to create a conceptual road map of the database which guides the retrieval process.

A software listing implementing the system of the present invention is attached hereto as a microfiche appendix. The program listing is a hexadecimal dump of the object code, and is implemented for use with the ORACLE relational database management system on DEC VAX and Micro VAX computers available from Digital Equipment Corporation of Maynard, Mass., under the VMS operating system.

Although the present invention has been described in connection with a preferred embodiment thereof, many variations and modifications can be made. It is intended to cover all such variations and modifications that fall within the scope of the present invention, as defined in the following claims.

APPENDIX A

Internal Meaning Representation Grammar

```

mr          → request-cd | retrieve-cd

request-cd  → (REQUEST) | (REQUEST request-slot-filler-pair+)

request-slot-filler-pair → request-attribute request-attribute-value

request-attribute → REFERENCE | AGGREGATE

request-attribute-value → {reference-value-cd+} | {aggregate-value-cd}
                        | {comparison-agg-cd} | {}

reference-value-cd → (ch-reference (ref-slot-filler-pair+) | ())

ch-attribute → attributes-in-ch-attribute-defobj | db-att

ch-attribute-value →

ch-reference → DB-REFERENCE

ref-slot-filler-pair → ref-slot (ref-slot-value-cd+)

ref-slot → DISPLAY-SLOTS | CONSTRAINED-SLOTS

display-slots-ref-slot-value-cd → (ch-attribute FILLER ch-attribute)

constrained-slots-ref-slot-value-cd → (ch-attribute FILLER
                                       (ch-attribute ST constraint-def))

```


constraint-def → (constraint+) | (OR ARG (constraint constraint+))

constraint → comparison-constraint | superlative-constraint |
time-constraint | specifier

comparison-constraint → (comparison-constraint-type ARG constraint-
value)
(comparison-constraint-type ARG constraint-
value BY constraint-by-value)

constraint-value → (number) | constant | reference-value-cd

constraint-by-value → (number) | constant

APPENDIX B

External Meaning Representation

DISPLAY CONCEPTS

The external meaning representation indicates what developer-defined concepts are referred to in the query. The following example contains the concepts CUSTOMER and CUST-BALANCE:

<u>Query</u>	<u>Meaning Representation</u>
"Show customer balances"	CUSTOMER: CUST-BALANCE:

SELECTION

Selection criteria are displayed following the concept, e.g.

<u>Query</u>	<u>Meaning Representation</u>
"Show products with quantity on hand > 500"	PRODUCT: "quantity on hand": > 500

FORMATTING

Formatting requests are indicated as follows:

<u>Query</u>	<u>Meaning Representation</u>
"Show sales by customer and product"	SALES: CUSTOMER: SORT(1) PRODUCT: SORT(2)
"List customers by sales in descending order"	SALES: SORT(1), DESCENDING CUSTOMER:
"Rank my best 5 customers in terms of sales"	SALES: SORT(1), DESCENDING, TOP(5) CUSTOMER:

Appendix A

constant → (constant-type = literal-value)

literal-value → (number) | (string)

constant-type → ALPHA | PERCENT | LOC | LENGTH | CURRENCY | NUMBER

comparison-constraint-type → < | <= | = | >= | > | <>

superlative-constraint → QUALITY-AVERAGE |
SORT-SPEC sort-spec-slot-filler-pair+ |
SORT-SPEC

sort-spec-slot-filler-pair → {TO number-constant}? | {DIR {(BOTTOM-UP)
| (TOP-DOWN) | (NORMAL) | (REVERSE) }}?

{SHOWING reference-value-cd}? |
{EXPLICIT ((EXPLICIT) | (RANK))}? |
{SORT-ORDER number constant}?

time-constraint → (TB-CURRENT) | (TB-WTD) | (TB-WEEK) ORIGIN (-? number)
MTD MONTH
YTD YEAR
QTD DAY
LYTD PERIOD
PYTD QUARTER
(TIME (FROM date-value)? (TO date-value)?) |
(TB-RANGE-REL FROM date-value TO date-value UNIT
time-unit
(TB-LENGTH = (number) UNIT (time-unit)) |
(TB-DAY-OF-WEEK DAY (number) ORIGIN (number)))

time-constraint → POINT | RANGE

aggregate-value-cd → (ch-aggregate RESULT-NAME ((string))
OP1 (aggregate | comparison-agg |
reference-value-cd)
OP2 (aggregate | comparison-agg |
reference-value-cd))

comparison-agg-cd → (ch-comparison-agg RESULT-NAME ((string))
OP1 (aggregate | comparison-agg
| reference-value-cd)
OP2 (aggregate | comparison-agg
| reference-value-cd)
BY constant)

ch-aggregate → ADD | SUBTRACT | MULTIPLY | DIVIDE | PERCENT |
AVERAGE | COMPARISON | ABS-VAL-OF

ch-comparison-agg → GT-AGG | LT-AGG | GT-PCT-AGG | LT-PCT-AGG

Appendix B

MODIFIERS

Modifiers are semantic modifiers that are represented similarly to constraints. For example:

<u>Query</u>	<u>Meaning Representation</u>
"List each invoice"	INVOICE: EACH
Examples of modifiers that could be built into the General English Lexicon of the system include:	
<u>Words</u>	<u>Modifier</u>
"detail"	DETAIL
"each", "every"	EACH

"all"	ALL
"location", "where"	LOCATION
"should I", "can I", "will I"	EXPERT
"last"	LAST
"number"	NUMBER
"status", "flag"	STATUS
"frequent", "frequency"	FREQUENT
"rate"	RATE
"total"	TOTAL
"unit"	UNIT
"history"	HISTORY
"type", "class", "kind", "sort of"	CLASS
"code"	CODE
"future"	FUTURE
"miscellaneous"	MISCELLANEOUS

Modifiers can also be developer-defined.

DATATYPES

Certain words are interpreted by the Natural Language Interface as referring to datatypes. Examples include the following:

<u>Words</u>	<u>Datatype</u>
"value", "dollar", "amount", "\$", "buck"	DOLLAR
"amount", "size", "count", "quantity", "number of", "volume"	COUNT
"percent", "%"	PERCENT

TIME

The time period covered by a column is represented as follows:

<u>Query</u>	<u>Meaning Representation</u>
"Show ytd sales"	SALES: TIME(YTD)

Legal time specifications include:

<u>Words</u>	<u>Time</u>
"n days ago"	DAY(-n)
"n days from now"	DAY(+n)
"n weeks ago"	WEEK(-n)
"n weeks from now"	WEEK(+n)
"n months ago"	MONTH(-n)
"n months from now"	MONTH(+n)
"n periods ago"	PERIOD(-n)
"n periods from now"	PERIOD(+n)
"n quarters ago"	QUARTER(-n)
"n quarters from now"	QUARTER(+n)
"n years ago"	YEAR(-n)
"n years from now"	YEAR(+n)
"since ..."	SINCE,<time>
<time> is any of above	
"before ..."	BEFORE,<time>
<time> is any of above	
"from ... to ..."	RANGE,<time1>,<time2>
<time1> and <time2> are any of above values	

"wtd", "week to date" WTD
 "mtd", "month to date" MTD
 "qtd", "quarter to date" QTD
 "ptd", "period to date" PTD
 "ytd", "year to date" YTD
 "lytd", "last year to date" LYTD
 "pytd", "previous year to date" PYTD
 "January 1", "from Sept to Oct", ... (e.g. "Jan. 1, 1985 - Feb. 15" --->
 ABSOLUTE (1/1/85, 2/15/85) ABSOLUTE, dd/mm/yy, dd/mm/yy

What is claimed is:

1. A database retrieval system having a natural language interface, said system comprising:

a computer processor;

a natural language interface coupled to said computer processor;

first means operatively associated with said computer processor for producing a database-independent, canonical, internal meaning representation of a natural language query entered into said natural language interface;

second means operatively associated with said computer processor for identifying database elements that are necessary to satisfy the query represented by said internal meaning representation;

third means operatively associated with said computer processor for generating a database query among database elements identified by said second means, said database query enabling the retrieval and aggregation of data from a database to satisfy said natural language query; and

debugging means for deriving an external meaning representation from said internal meaning representation, wherein said external meaning representation is provided in a form that can be easily understood by a database developer;

said external meaning representation enabling a database developer to comprehend the internal meaning representation and verify that a natural language query entered into the natural language interface is properly interpreted to enable the correct retrieval and aggregation of data from said database.

2. The database retrieval system of claim 1 wherein said external meaning representation comprises entities and constraints relating to the entities, without reference to factual or linguistic relationships between entities that would prevent the external meaning representation from being easily understood by a database developer.

3. The database retrieval system of claim 2 further comprising:

means for displaying the external meaning representation to the database developer.

4. The database retrieval system of claim 1 further comprising:

tool kit means coupled to said computer processor for enabling the database developer to create a knowledge base containing a structural description and a semantic description of a database from which data is to be retrieved; and

wherein said second means comprises an expert system coupled to access structural and semantic description information in said knowledge base and identify said

15 database elements from said information in accordance with predefined rules.

5. The database retrieval system of claim 4 wherein said database structure is columnar, and said semantic description comprises a concept index of database columns.

6. The database retrieval system of claim 5 wherein said semantic description further comprises a time frame, value unit of measure, and aggregation level of database columns.

7. The database retrieval system of claim 4 further comprising:

means for enabling a database developer to view the external meaning representation produced by said debugging means.

8. The database retrieval system of claim 7 wherein said debugging means comprise means for producing a representation of the database elements identified by said second means for viewing by a database developer.

9. The database retrieval system of claim 8 wherein said debugging means also enables a database developer to view the database query generated by said third means.

10. The database retrieval system of claim 4 wherein said external meaning representation comprises entities and constraints relating to the entities, without reference to factual or linguistic relationships between entities that would prevent the meaning representation from being easily understood by a database developer.

11. The database retrieval system of claim 10 further comprising:

means for enabling a database developer to view the external meaning representation produced by said debugging means.

12. The database retrieval system of claim 4 wherein said rules comprise steps for identifying an optimal set of database elements to satisfy the query represented by the internal meaning representation.

13. The database retrieval system of claim 12 further comprising:

means operatively associated with said computer processor for generating a formatted report containing data from said database responsive to said natural language query.

14. The database retrieval system of claim 12 wherein said steps for identifying an optimal set of database elements include:

locating initial indexed columns;
 testing subtypes;
 testing data class-to-table rules;
 matching characteristics and constraints;
 matching time constraints;
 testing master table quantifiers;
 testing for detail file columns;

eliminating foreign-key-only tables;
 minimizing tables; and
 selecting the optimal navigation path for satisfying a query.

15. A database retrieval system having a natural language interface, said system comprising:

a computer processor;
 tool kit means coupled to said computer processor for enabling a database developer to create a knowledge base containing a structural description and a semantic description of a database from which data is to be retrieved;

a natural language interface coupled to said computer processor;

means operatively associated with said computer processor for producing a database-independent, canonical, internal meaning representation of a natural language query entered into said natural language interface;

expert system means for accessing structural and semantic description information in the knowledge base, and using said information to identify database elements that are necessary to satisfy the query represented by said internal meaning representation; and

means operatively associated with said computer processor for generating a database, query among database elements identified by said expert system means, said database query enabling the retrieval and aggregation of data from a database to satisfy said natural language query.

16. The database retrieval system of claim 15 wherein said tool kit means comprises means for enabling a database developer to enter join criteria into said knowledge base.

17. The database retrieval system of claim 15 wherein said tool kit means comprises means for enabling a database developer to enter data group definitions into said knowledge base.

18. The database retrieval system of claim 15 wherein said tool kit means comprises means for enabling a database developer to enter word and phrase associations into said knowledge base.

19. The database retrieval system of claim 15 wherein said tool kit means comprises means for enabling a database developer to add, delete, and modify subtypes in said knowledge base.

20. The database retrieval system of claim 15 wherein said tool kit means comprises means for enabling a database developer to add, modify and delete column references in said knowledge base.

21. The database retrieval system of claim 15 wherein said tool kit means comprises means for enabling a database developer to add, delete, and modify word-to-data class rules in said knowledge base.

22. The database retrieval system of claim 15 wherein said tool kit means comprises means for enabling a database developer to add, delete, and modify data class-to-table rules in said knowledge base.

23. The database retrieval system of claim 15 wherein said tool kit means comprises means for enabling a database developer to add, delete, and modify nominal data definitions in said knowledge base.

24. A method for retrieving data from a database comprising the steps of:
 inputting a natural language query to a computer processor;

processing said query in said processor to produce an internal meaning representation thereof;

identifying database elements that are necessary to satisfy the query represented by said internal meaning representation;

generating a database query among the identified database elements, for use in the retrieval and aggregation of data from an application database to satisfy said natural language query; and

deriving an external meaning representation from said internal meaning representation, to enable a database developer to comprehend the internal meaning representation and verify that a natural language query is properly processed to enable the correct retrieval of data from said database.

25. The method of claim 24 wherein said database elements comprise tables and columns in a relational database, and said identifying step comprises the steps of:

locating an initial set of candidate columns that contain data responsive to the query represented by said internal meaning representation; and
 eliminating candidate columns from said initial set that:

- (a) contain a restriction not specified in said internal meaning representation, and
- (b) are also not directly referenced by a concept in the internal meaning representation.

26. The method of claim 25 wherein said identifying step comprises a plurality of additional steps subsequent to the identification of said database elements, said subsequent steps including:

locating concepts in said internal meaning representation that have associated data class-to-table rules which specify tables applicable to the query represented by said internal meaning representation;

testing candidate columns for each concept located in the preceding step to determine if they are included in at least one table specified by the associated data class-to-table rule; and

eliminating candidate columns tested in said testing step that are not included in at least one table specified by the associated data class-to-table rule.

27. The method of claim 26 wherein said identifying step comprises, after the eliminating step of claim 26, the subsequent step of:

eliminating candidate columns that provide semantic matches which are not as close as semantic matches provided by other candidate columns for each concept in said internal meaning representation.

28. The method of claim 27 wherein said identifying step comprises, after the eliminating step of claim 27, the subsequent steps of:

determining whether any candidate columns contain an exact time match for a time period specified in the query represented by said internal meaning representation; and if so,
 eliminating redundant candidate columns that do not contain an exact time match.

29. The method of claim 28 wherein said identifying step comprises, after the eliminating step of claim 28, the subsequent steps of:

determining if quantification is present in the query represented by the internal meaning representation; and if so:

- (i) locating concepts within a chain of quantification;

33

- (ii) determining if any concepts located in step (i) index a primary key column of a table;
- (iii) selecting any table to which step (ii) pertains as a source of data responsive to the associated concept; and
- (iv) selecting tables other than those selected in step (iii) as sources of data responsive to concepts located in step (i) that do not index a primary key column of a table.

30. The method of claim 29 wherein said identifying step comprises, after the steps of claim 29, the subsequent steps of:

- locating concepts that have a detail constraint associated therewith;
- selecting candidate columns having detail attributes as a source of data responsive to concepts located in the preceding step; and
- eliminating redundant candidate columns as sources of data responsive to concepts referred to in the preceding step unless the candidate columns are in tables previously selected on the basis of said data class-to-table rules.

31. The method of claim 30 wherein said identifying step comprises, after the eliminating step of claim 30, the subsequent steps of:

- locating candidate columns that include time attributes;
- noting tables in which the candidate columns located in the preceding step reside;
- eliminating redundant candidate columns from tables other than those:
 - (i) noted in said noting step, or
 - (ii) which are selected on the basis of said data class-to-table rules.

32. The method of claim 31 wherein said identifying step comprises, after the eliminating step of claim 31, the subsequent steps of:

- locating concepts that have candidate columns from more than one table;
- noting any candidate columns for a concept located in the preceding step that are non-foreign key fields; and
- if any columns are noted in the immediately preceding step, eliminating any candidate columns for the concept referred to in the immediately preceding step that are from key only tables.

33. The method of claim 32 wherein said identifying step comprises, after the steps of claim 32, the subsequent steps of:

- determining, for each concept, if there are candidate columns from more than one table; and if so,
- eliminating redundant candidate columns from tables other than those with the highest count of such candidate columns.

34. The method of claim 33 wherein said identifying step comprises, after the steps of claim 33, the subsequent step of:

- choosing a navigation path among the remaining tables and candidate columns for satisfying the query represented by the internal meaning representation.

35. The method of claim 34 wherein said step of choosing the navigation path comprises the steps of:

- determining if there still exist concepts that have candidate columns from multiple tables; and if so:
 - (a) creating a set of all of the possible combinations of such tables, each combination representing a candidate navigation path,
 - (b) determining whether the multiple tables associated with a concept were selected on the basis of a data class-to-table rule, and if so, deleting can-

34

- didate navigation paths that do not include all of the tables required by the data class-to-table rule,
- (c) ordering the remaining candidate navigation paths in accordance with predefined navigation path rules to obtain said navigation path for satisfying the query represented by the meaning representation.

36. The method of claim 35 wherein said predefined navigation path rules comprise:

- (i) giving priority to paths within tables that have a predefined navigation path,
- (ii) giving priority to paths containing fewer tables;
- (iii) giving priority to paths with more primary key columns;
- (iv) giving priority to paths in which the total number of primary key columns in all tables in the path is fewest;
- (v) giving priority to paths for which the sum of the hierarchy levels of the candidate columns is lowest;
- (vi) giving priority to paths for which the sum of the estimated number of rows in the tables is fewest; and
- eliminating any candidate columns from tables that are not included in the highest priority navigation path resulting from the application of the above navigation path rules.

37. The method of claim 36 comprising, after the eliminating step of claim 36, the subsequent steps of:

- adding table and column level columns to data from said database to be displayed;
- determining if the adding step introduced new tables; and if so:
- applying said navigation path rules again to produce a final navigation path.

38. The method of claim 24 comprising the further steps of:

- building a knowledge base representative of said application database; and
- enabling the developer to modify the knowledge base, to correct an erroneous query interpretation represented by the external meaning representation.

39. A method for deriving, from an internal meaning representation produced by a natural language computer interface, an external meaning representation for use in debugging said interface by allowing a human being to comprehend a query interpretation represented by said internal meaning representation, comprising the steps of:

- identifying entities contained in the internal meaning representation;
- identifying constraints associated with the entities in the internal meaning representation; and
- combining said entities and constraints into an external meaning representation that can be easily comprehended by a human being.

40. The method of claim 39 comprising the further step of:

- deriving said internal meaning representation from a database query;
- wherein said external meaning representation is canonical, indicative of said database query, and is database independent.

41. The method of claim 39 comprising the further step of:

- ignoring factual or linguistic relationships between entities in the internal meaning representation that would prevent the external meaning representation from being easily comprehended by a human being if included therein.

* * * * *



US005748974A

United States Patent [19]

[11] Patent Number: 5,748,974

Johnson

[45] Date of Patent: May 5, 1998

[54] MULTIMODAL NATURAL LANGUAGE INTERFACE FOR CROSS-APPLICATION TASKS

[75] Inventor: David Edward Johnson, Peekskill, N.Y.

[73] Assignee: International Business Machines Corporation, Armonk, N.Y.

[21] Appl. No.: 354,987

[22] Filed: Dec. 13, 1994

[51] Int. Cl.⁶ G06F 9/45

[52] U.S. Cl. 395/759; 395/700

[58] Field of Search 364/419.08, 419.01, 364/419.02, 419.03, 419.04; 395/155, 156, 161, 375, 934, 700, 759

Attorney, Agent, or Firm—Whitham, Curtis, Whitham & McGinn; Stephen J. Kaufman

[57] ABSTRACT

A multimodal natural language interface interprets user requests combining natural language input from the user with information selected from a current application and sends the request in the proper form to an appropriate auxiliary application for processing. The multimodal natural language interface enables users to combine natural language (spoken, typed or handwritten) input selected by any standard means from an application the user is running (the current application) to perform a task in another application (the auxiliary application) without either leaving the current application, opening new windows, etc., or determining in advance of running the current application what actions are to be done in the auxiliary application. The multimodal natural language interface carries out the following functions: (1) parsing of the combined multimodal input; (2) semantic interpretation (i.e., determination of the request implicit in the pars); (3) dialog providing feedback to the user indicating the systems understanding of the input and interacting with the user to clarify the request (e.g., missing information and ambiguities); (4) determination of which application should process the request and application program interface (API) code generation; and (5) presentation of a response as may be applicable. Functions (1) to (3) are carried out by the natural language processor, function (4) is carried out by the application manager, and function (5) is carried out by the response generator.

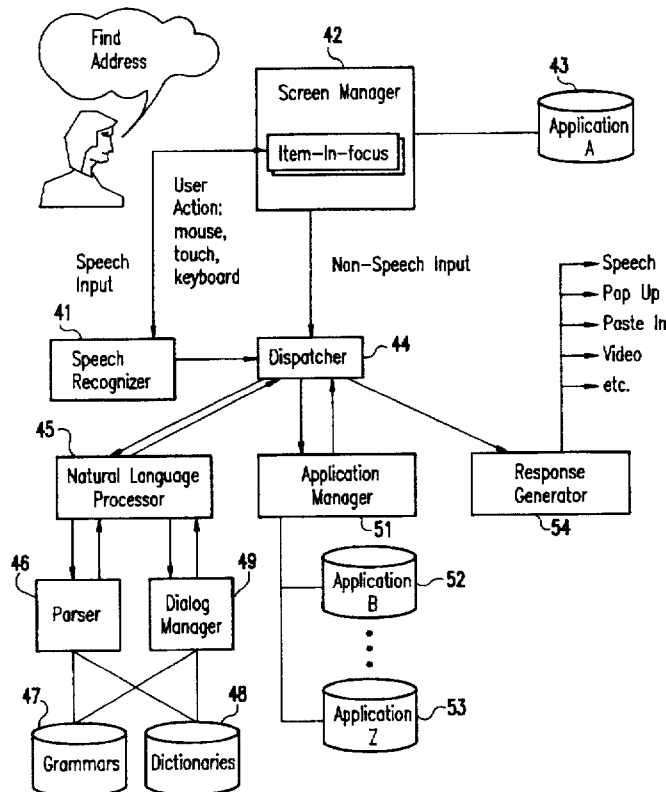
[56] References Cited

U.S. PATENT DOCUMENTS

4,736,296	4/1988	Katayama	364/419
5,252,951	10/1993	Tennenbaum	345/156
5,282,265	1/1994	Rohra Suda	364/419.08
5,301,326	4/1994	Linnett	395/700
5,321,608	6/1994	Namba	364/419.08
5,377,103	12/1994	Lamberti	364/419.08
5,390,281	2/1995	Luciw	364/419.08
5,442,780	8/1995	Tanashi	364/419.08

Primary Examiner—Jeffery Hofsass
Assistant Examiner—Albert K. Wong

8 Claims, 7 Drawing Sheets



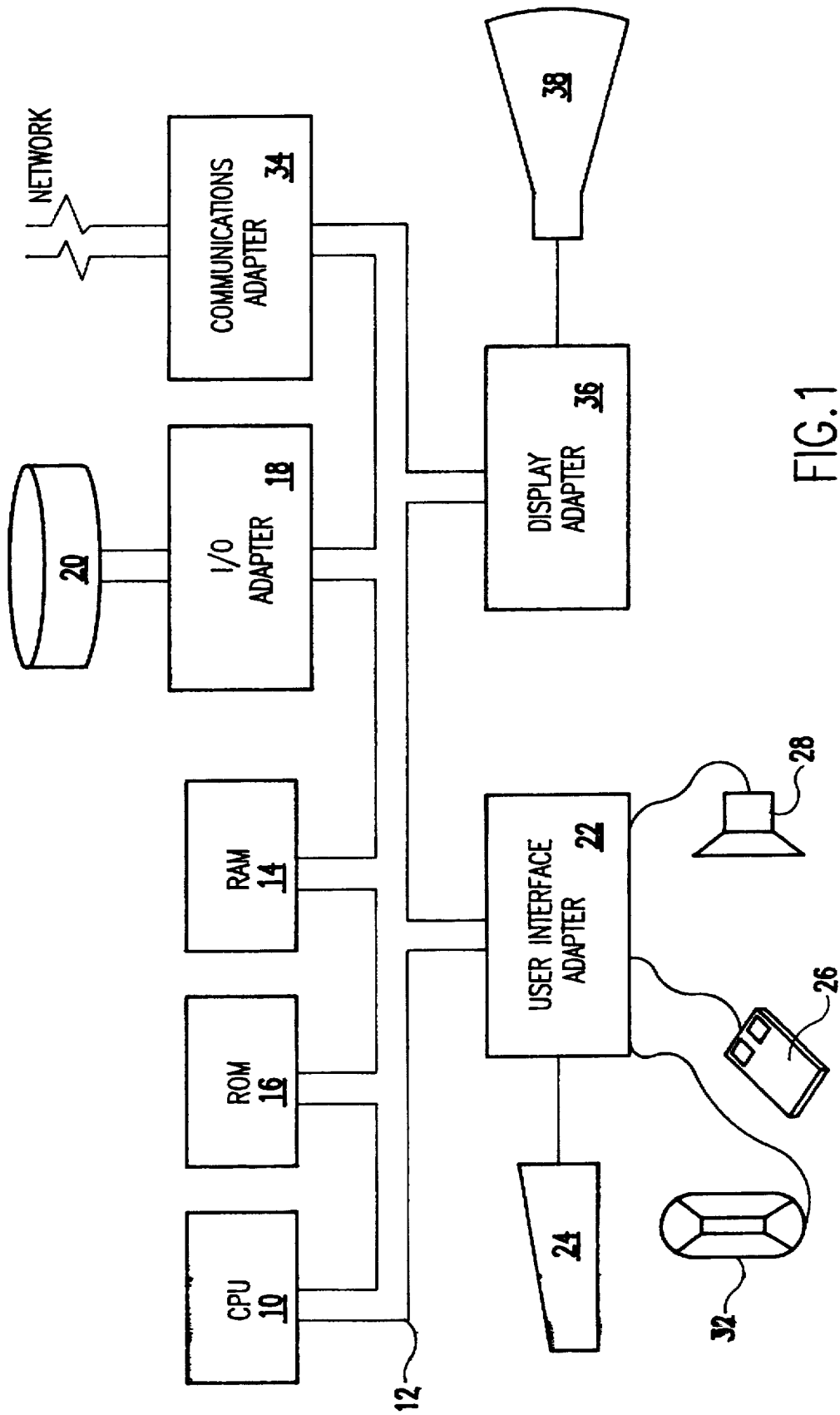


FIG. 1

PRIOR ART

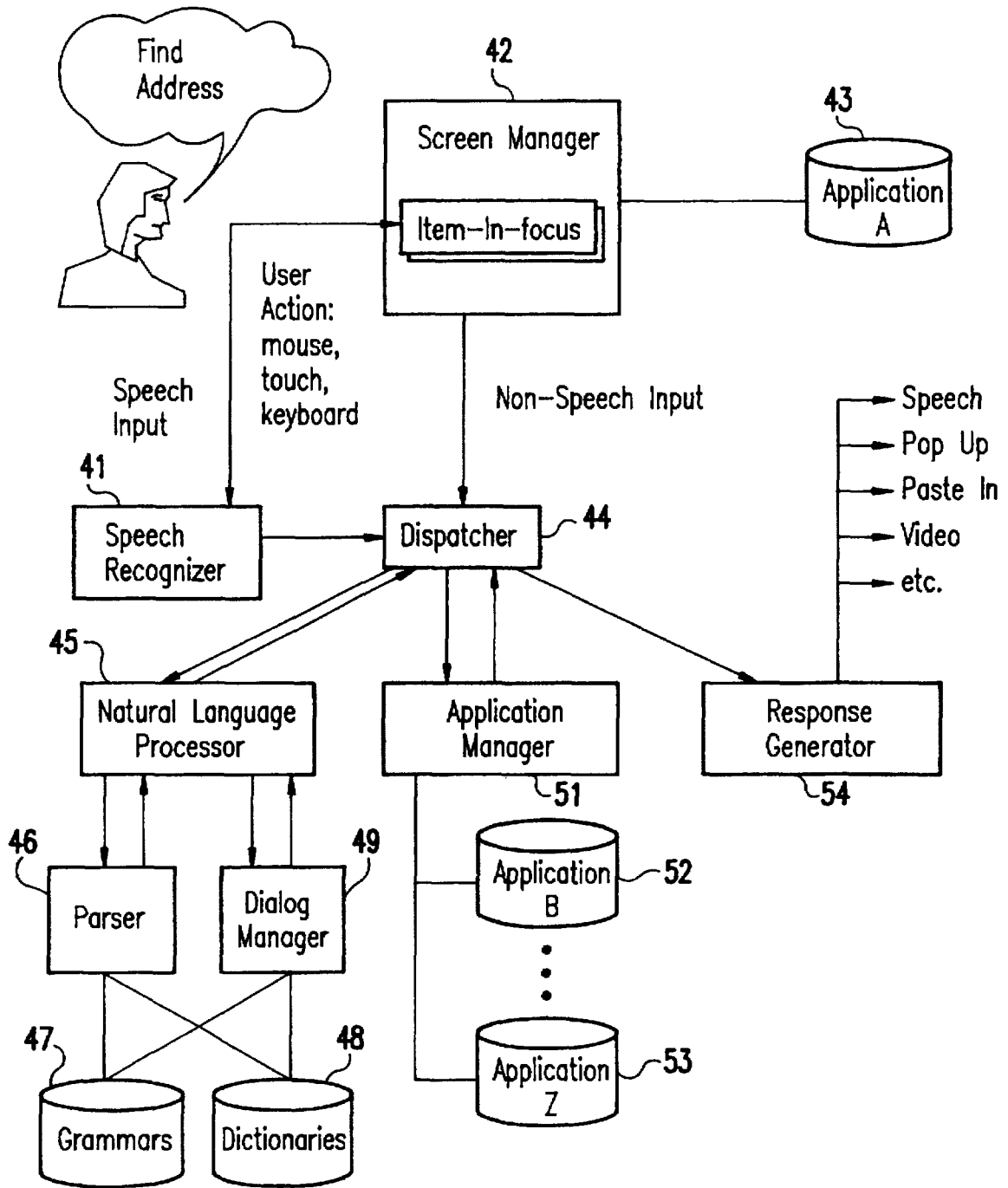


FIG.2

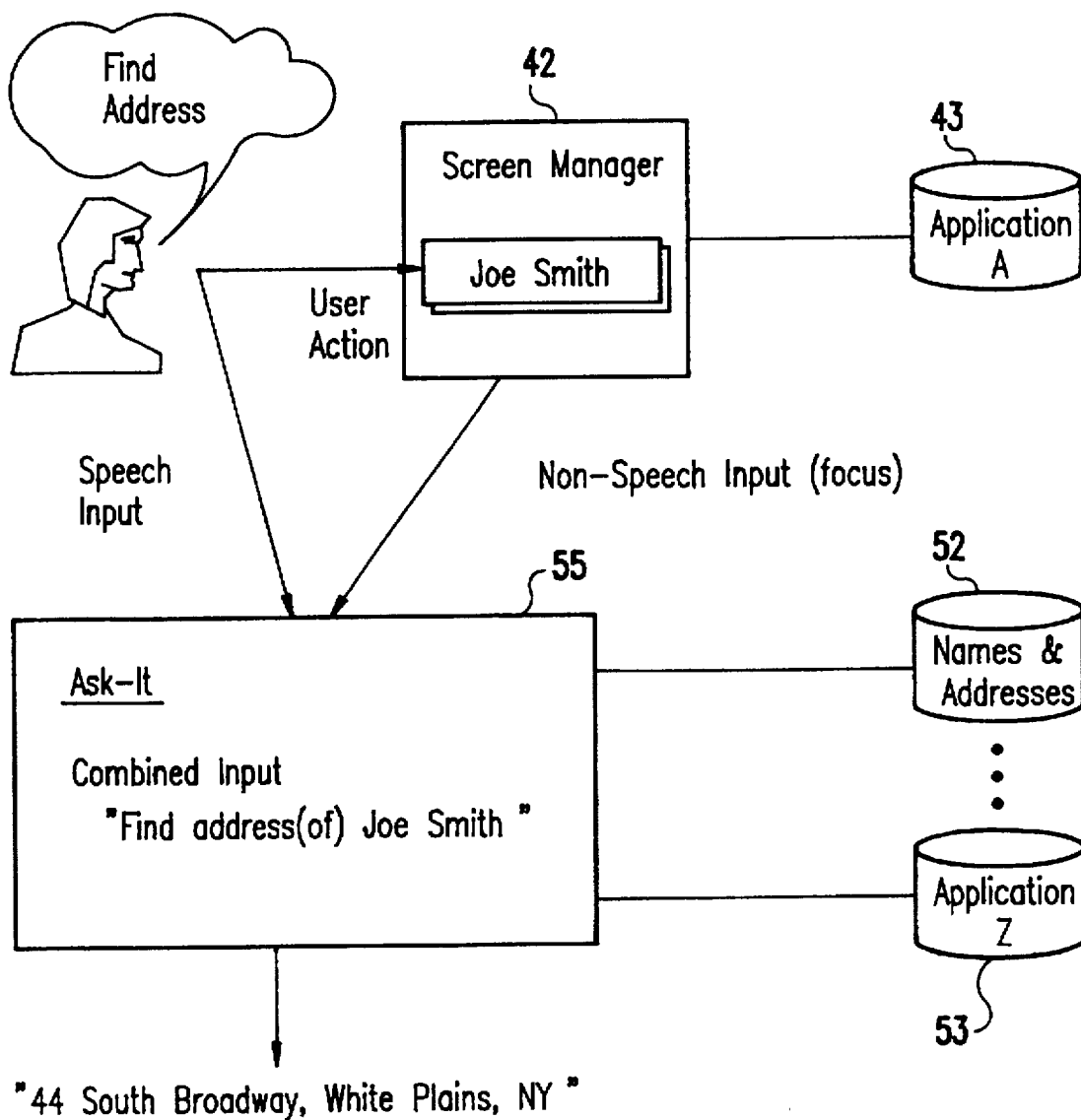
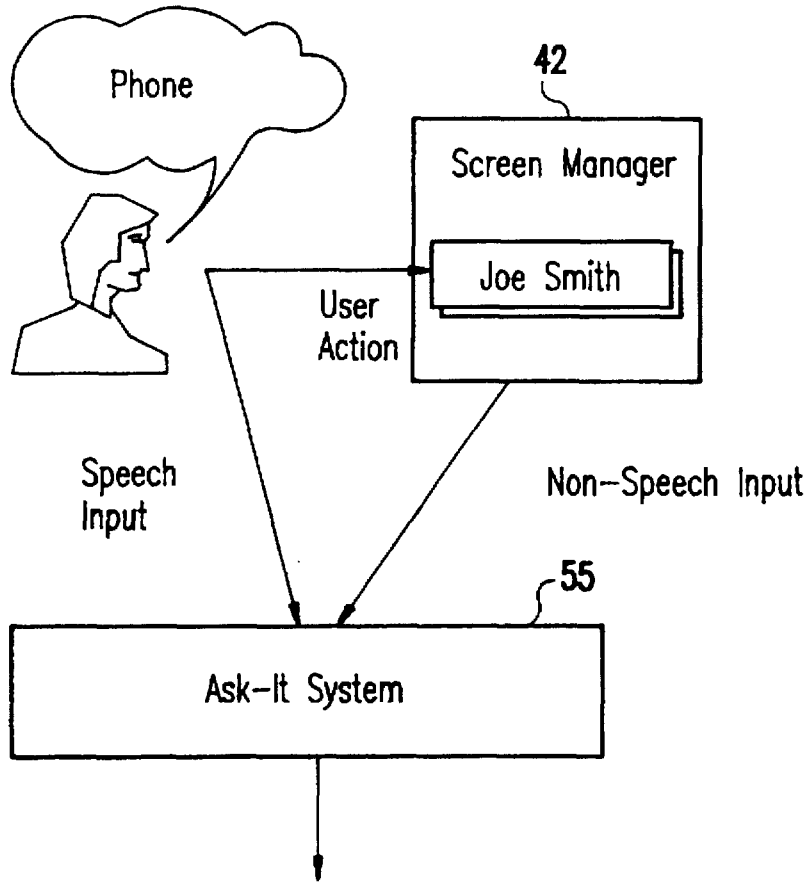


FIG.3



Kind of
Answers:

1. 609-921-9521
2. There are 2 such names. Do you mean:
 1. Joe A. Smith
 2. Joe B. Smith?Please select one.
3. There is no Joe Smith in your phonebook. Should I look elsewhere?

FIG.4

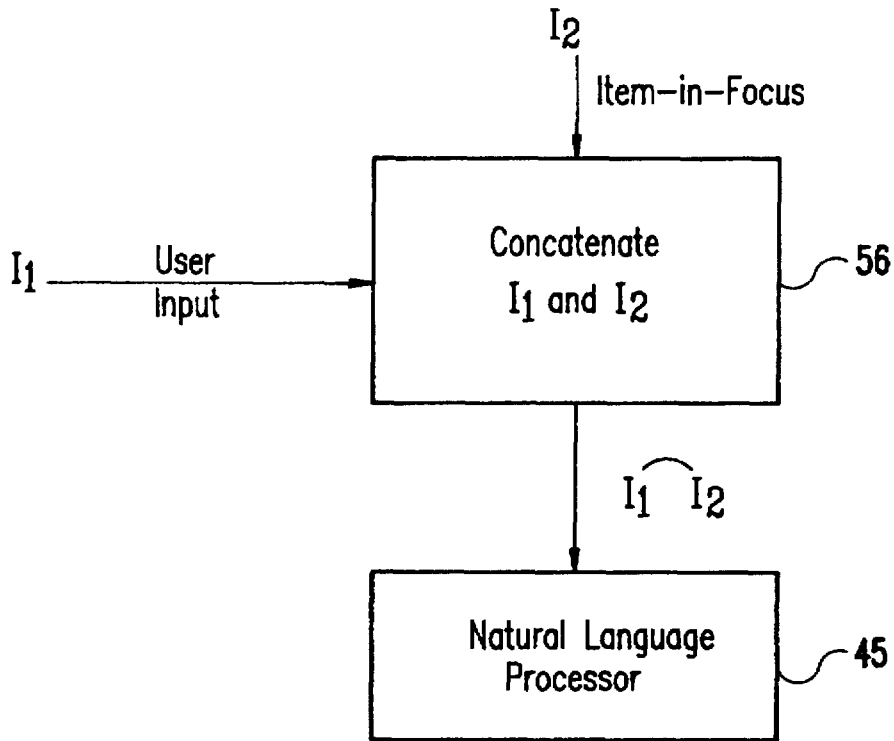


FIG.5

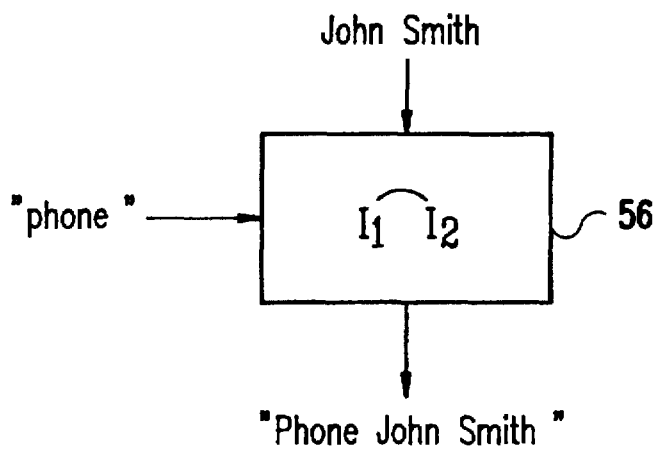


FIG.5A

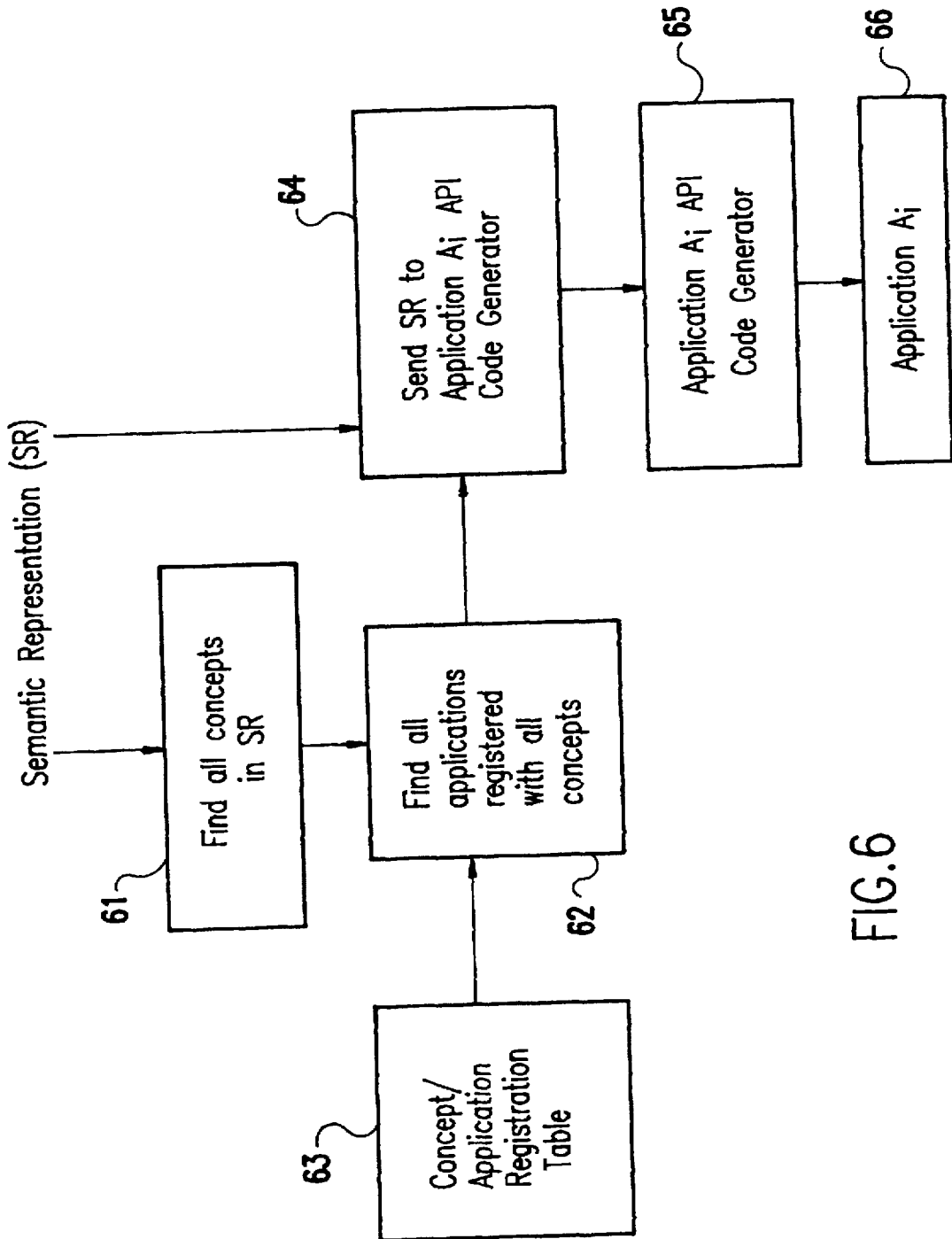


FIG. 6

App : Con	Al	Ai	Ar
⋮			
Person		✓	✓
⋮			
Phone	✓	✓	
Fax ⋮		✓	✓

FIG.6A

MULTIMODAL NATURAL LANGUAGE INTERFACE FOR CROSS-APPLICATION TASKS

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention generally relates to user interfaces for computer systems and, more particularly, to a multimodal natural language interface that allows users of computer systems conversational and intuitive access to multiple applications. The term "multimodal" refers to combining input from various modalities; e.g., combining spoken, typed or handwritten input from the user.

2. Description of the Prior Art

Since the introduction of the personal computer, it has been a goal to make using such a computer easier. This goal recognizes that greater numbers of people are using computers in their daily lives and business and that the majority of the people using computers have little training in their use. The term "user friendly" was coined to describe applications running on computers which required minimal training for a user to be able to effectively use those applications and become productive. In a business context, training employees in the use of a computer can be a very expensive overhead cost to the business.

The graphic user interface (GUI) was introduced by the Xerox Palo Alto Research Center (PARC) and made popular by the Apply Macintosh computers. The GUI is often described as a "point-and-click" interface because a cursor pointing device, such as a mouse, trackball or the like, is used to move a cursor on the display to an icon or command bar where the user simply "clicks" or, in some cases, double "clicks" a mouse button, for example. This is in contrast to typing in carefully composed commands, a process which is anything but intuitive. The GUI is now the de facto standard in such operating systems and International Business Machines (IBM) Corporation's OS/2 operating system and the forthcoming Microsoft Windows 95 operating system.

While the GUI has been a major improvement in computer interfaces, the effective use of applications running under operating systems supporting a GUI still requires a knowledge of procedures to effectively use applications running on those operating systems. For example, users running an application (current application) frequently want to perform some unanticipated task in another application (auxiliary application) based in part on information in the current application. Currently, performing such tasks is time-consuming and cumbersome, requiring the user to determine what auxiliary application needs to be accessed, open a new window, import information from the current application, and other related tasks. Thus, as important as the GUI has been in making computer systems "user friendly", there still remains much improvement to be made to facilitate use of computers by an increasingly large number of people.

SUMMARY OF THE INVENTION

It is therefore an object of the present invention to provide a multimodal natural language interface that interprets requests combining natural language input from the user with information selected from the current application and sends the request in the proper form to the appropriate application for processing.

According to the invention, there is provided a multimodal natural language interface that enables users to combine

natural language (spoken, typed or handwritten) input selected by any standard means from an application the user is running (the current application) to perform a task in another application (the auxiliary application) without either leaving the current application, opening new windows, etc., or determining in advance of running the current application what actions are to be done in the auxiliary application.

The invention carries out the following functions: (1) parsing of the combined multimodal input; (2) semantic interpretation (i.e., determination of the request implicit in the parse); (3) dialog providing feedback to the user indicating the systems understanding of the input and interacting with the user to clarify the request (e.g., missing information and ambiguities); (4) determination of which application should process the request and application program interface (API) code generation; and (5) presentation of a response as may be applicable. Functions (1) to (3) are carried out by the natural language processor, function (4) is carried out by the application manager, and function (5) is carried out by the response generator.

The invention allows the use of multimodal (spoken, typed, handwritten) natural language input supplied by the user combined with information selected from a current application via any standard technique. The invention further provides a unique combination and application of techniques from artificial intelligence and computational linguistics that have been used in other applications, e.g., natural language database query and machine translation, in the area of user interfaces supporting cross-application tasks. Together, these go beyond current state-of-the-art user interfaces supporting cross-application tasks.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other objects, aspects and advantages will be better understood from the following detailed description of a preferred embodiment of the invention with reference to the drawings, in which:

FIG. 1 is a block diagram showing a hardware configuration on which the subject invention may be implemented;

FIG. 2 is a block diagram of the multimodal system architecture according to the present invention;

FIG. 3 is a block diagram of a first example of the operation of the multimodal system shown in FIG. 2;

FIG. 4 is a block diagram of a second example of the operation of the multimodal system shown in FIG. 2;

FIG. 5 is a flow diagram showing the logic of the combining multimodal linguistic input function of the dispatcher;

FIG. 5A is an example of the combining multimodal linguistic input function of the dispatcher;

FIG. 6 is a flow diagram showing the logic of the application manager; and

FIG. 6A is an example of a concept/application registration table used by the application manager.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT OF THE INVENTION

Referring now to the drawings, and more particularly to FIG. 1, there is shown a representative hardware environment on which the subject invention may be implemented. This hardware environment may be a personal computer, such as the IBM's PS/2 family of Personal Computers, running an operating system capable of supporting multitasking, such as IBM's OS/2 operating system. The

hardware includes a central processing unit (CPU) 10, which may conform to Intel's X86 architecture or may be a reduced instruction set computer (RISC) microprocessor such as IBM's PowerPC® microprocessor. The CPU 10 is attached to a system bus 12 to which are attached a read/write or random access memory (RAM) 14, a read only memory (ROM) 16, an input/output (I/O) adapter 18, and a user interface adapter 22. The RAM 14 provides temporary storage for application program code and data, while ROM 16 typically includes the basic input/output system (BIOS) code. The I/O adapter 18 is connected to one or more Direct Access Storage Devices (DASDs), here represented as a disk drive 20. The disk drive 20 typically stores the computer's operating system (OS) and various application programs, each of which are selectively loaded into RAM 14 via the system bus 12. The user interface adapter 22 has attached to it a keyboard 24, a mouse 26, a speaker 28, a microphone 32, and/or other user interface devices (not shown). The personal computer also includes a display 38, here represented as a cathode ray tube (CRT) display but which may be a liquid crystal display (LCD) or other suitable display. The display 38 is connected to the system bus 12 via a display adapter 34. Optionally, a communications adapter 34 is connected to the bus 12 and to a network, for example a local area network (LAN), such as IBM's Token Ring LAN. Alternatively, the communications adapter may be a modem connecting the personal computer or workstation to a telephone line as part of a wide area network (WAN).

The preferred embodiment of the invention is implemented on a hardware platform as generally shown in FIG. 1. The architecture of the multimodal natural language interface according to the invention will now be described followed by specific examples of its operation. The multimodal natural language interface is linked to applications permitting users, from within a current application, to perform actions in an auxiliary application without the necessity of opening new windows or similar procedures. The term "multimodal" refers to the feature of combining input from various modalities; e.g., combining spoken, typed, or handwritten input from the user with input selected from an application the user is running by any standard means, including point-and-click, touch, and keyboard selection.

With reference now to FIG. 2 there is shown the basic architecture of the system. The user input may be spoken, typed, handwritten, mouse controlled cursor, touch, or any other modality. In the illustrated example, speech is input via microphone 32 (FIG. 1). The speech input, "Find address", is supplied to a speech recognizer 41 which generates an output. At the same time, the user may also provide non-speech input; e.g., by keyboard 24, mouse 26, a touch screen (not shown) attached to display 38, or the like. As mentioned the multimodal input contemplates handwritten input as well, and this may be accommodated by means of a stylus and tablet (not shown) or the mouse 26. This non-speech input is received by the screen manager 42, such as the Presentation Manager (PM) of the OS/2 operating system. The screen manager 42 also provides the a display window for application A, the current application, here shown as being accessed from a direct access storage device (DASD) 43, such as the hard disk 20 (FIG. 1). Within the window for application A, there is an "Item-in-Focus", such as text or a graphic.

The output of the speech recognizer 41 and the non-speech input received by the screen manager 42 are sent to a dispatcher 44 which combines the inputs and directs the combined input to first of all a natural language processor 45. The natural language processor 45 directs the combined

multimodal input to a parser/semantic interpreter 46 which accesses grammars and dictionaries on DASDs 47 and 48, which may be the same or different hard disk 20 (FIG. 1) on which application A resides. The parsed input is subjected to further semantic interpretation by the dialog manager 49, again with the aid of the grammars and dictionaries on DASDs 47 and 48. The natural language processor 45 provides feedback to the user via the dispatcher 44 to indicate the system's understanding of the input. If necessary, the natural language processor 45 interacts with the user to clarify any missing information or ambiguities in the request. The techniques employed by the natural language processor 45, parser 46 and dialog manager 49 are common in the area of natural language query database systems. Examples of commercially available natural language query database systems are IBM's "LanguageAccess" and NRI's "Natural Language" products.

Based on the output of the natural language processor 45, the dispatcher 44 invokes the application manager 51 to determine which application should process the request. Note that in the prior art the application manager of the operating system would have to be invoked by the user to first open a window for a selected application and then the application would have to be started and run in that window. The user would then have to access the requested information and then, using a clipboard function, copy and paste the information into the original application window. According to the invention, this is all done automatically without any intervention by the user. For example, the application manager 51 may access any of applications B to Z on DASDs 52 to 53, again which may be the same or different hard disk 20 (FIG. 1) on which application A resides. The application accessed is the auxiliary application. The application manager 51 determines which of applications B to Z has the requested information. The application manager 51 may determine that a database program, say application B, contains an address file where the requested information resides. The application manager 51 sends semantic representation of the request to the API code generator for application B which, in turn, generates the application program interface (API) code required to access the requested information. This is done without opening a window. The auxiliary application (e.g., the database program) is opened in the background and the API code (e.g., query) is generated to retrieve the requested information. Once the information has been accessed by the application manager 51, the requested information is supplied to the dispatcher 44 which then dispatches the information to the response generator 54. The response generator 54 then generates a response appropriate to the nature of the request and the current application. This response can be speech, from a synthesizer (not shown), text in a pop up window, text or a graphic which is pasted into the current application, a video clip, or the like.

Consider now a specific example with reference to FIG. 3. If the current application (application A) is a word processor and the user is writing a letter to Joe Smith, after typing John Smith's name via keyboard 24, the user may provide the speech input, "Find address". The combined multimodal input, the typed name of Joe Smith ("Item-in-Focus" in FIG. 1) and the spoken request "Find address", is processed by the natural language processor 45 and supplied by the dispatcher 44 to the application manager 51, here represented by the "Ask-It" block 55. In the example described, the combined input is "Find address (of) Joe Smith". The function performed is to access a names and addresses file 56 via a database program on DASD 52 and retrieve Joe Smith's address. The appropriate response is to

paste the retrieved address of Joe Smith in the letter being written by the word processor application (application A).

Consider next the example shown in FIG. 4. The user has typed in Joe Smith's name, but now instead of requesting an address, the user provides the speech input "Phone". There are several possible answers illustrated in the example of FIG. 4. The first is to retrieve Joe Smith's telephone number. However, if there are two Joe Smiths in the database, then there is an ambiguity that must be clarified before a final response can be generated. The dialog manager 49 (FIG. 2) will provide a choice to the user, perhaps in a pop-up window, and request the user to select one of the choices. On the other hand, there may be no Joe Smith listed in the phonebook, in which case there is not enough information in the request to process it. The dialog manager 49 would then inform the user that there is no Joe Smith listed and ask for more information, such as "Should I look elsewhere". This response could be a text display in a pop up window, for example, or synthesized speech. Ultimately, when the telephone number is located, the response could be either a listing of the number itself or the number would be dialed via the communications adapter 34 (FIG. 1).

The functions which support the multimodal natural language interface are the dispatcher 44 and the application manager 51 shown in FIG. 2. With reference now to FIG. 5, the dispatcher function is illustrated by way of a flow diagram. The user input, I1, and the item-in-focus input, I2, from the current application are simply concatenated in function block 56 as "user input"+"item-in-focus". The grammar and semantic interpretation rules used in the natural language processor 45 insure the intended meaning is recovered. As mentioned, various state of the art natural language processing systems can be used to perform the function of the natural language processor 45. Even if the concatenated input to the natural language processor 45 does not match the natural order of the natural language processed, the natural language processor will still recover the intended meaning. For example, if the concatenated input were "send to Mary"<filename>, meaning "send to Mary financial data", the natural language processor 45 would understand this by the correct English expression "send <filename> to Mary", meaning "send financial data to Mary" since the natural language processor can analyze unusual word orders by supplying the appropriate grammatical rules. A significant ease of use advantage of this system is that the user input and the input supplied from the current application can be input in either temporal order or even overlap in time.

FIG. 5A provides another example of the operation of the dispatcher function 56. In this case, the user input is "phone" and the application input is "John Smith". The dispatcher concatenation function is to output "phone John Smith" to the natural language processor.

The flow diagram of the application manager 51 is shown in FIG. 6, to which reference is now made. For a given input, the application manager first finds all concepts in the semantic representation provided by the natural language processor 45 in function block 61 and then, in function block 62, determines from the semantic representation each application that is registered with every concept in the semantic representation. This determination is made by referencing a concept/application table 63. Some concepts might be stipulated to be application independent, and those would not need to be considered. Such concepts could be identified by a flag set in a dictionary. Each application-specific concept is listed along with the names of the applications registered with that concept in the concept/application registration

table 63. This is logically just a table where, without loss of generality, the columns are labeled with application names and the rows with concept names. An example is shown in FIG. 6A. Once the set of application-specific concepts is determined, each such concept is looked up in the concept/application registration table, and the associated set of registered application names is returned. Each concept thus results in a set of application names being produced, which may be referred to as a "Concept-Application Set". After each concept has been processed, the result is a collection of Concept-Application Sets, one set of application names for each application-specific concept looked up in the concept/application registration table 63. The name of each application that occurs in every Concept-Application Set derived from the input semantic representation is determined. Logically, this can be done by simple set intersection. The result is a set of application names (Application Set), all of which are registered with each application-specific concept derived from the semantic representation of the input.

Next, in function block 64, the application manager sends the semantic representation to the API code generator 65 of each such application. Typically, there will be only one, but nothing precludes more than one application name occurring in the Application Set. In such a case, the input is truly ambiguous and the system could either report this to the user via the dispatcher or simply submit the semantic representation to each of the named application API code generators or both. Nothing in the architecture hinges on this choice and parameter could be set to determine the actual behavior of the system in particular circumstances. It is also possible that the Application Set is empty, corresponding to an input that was not meaningful with respect to the applications registered with the system in the concept/application registration table 63. This event would be reported back to the dispatcher for further processing, e.g., interaction with the user to determine the next action, if any. Assuming that an application is found and the semantic representation is sent to that application's API code generator in function block 65, the application then acts on the code in function block 66 to retrieve the data requested.

While the invention has been described in terms of a single preferred embodiment, those skilled in the art will recognize that the invention can be practiced with modification within the spirit and scope of the appended claims.

Having thus described my invention, what I claim as new and desire to secure by Letters Patent is as follows:

1. A multimodal natural language interface for a computer system which interprets user requests combining natural language input from the user with information selected from a current application running on the computer system and sends the request in proper form to an appropriate auxiliary application for processing, the multimodal natural language interface comprising:

- a dispatcher receiving a natural language input from the user and combining the natural language input with input information selected from a current application to form a combined multimodal request;
- a parser receiving the combined multimodal request for parsing the combined multimodal request;
- a natural language processor performing semantic interpretation of the parsed combined multimodal request and generating a semantic representation of the combined multimodal request;
- an application manager receiving the semantic representation from the natural language processor for determining which auxiliary application should process the

7

request, said application manager invoking the auxiliary application and generating application program interface (API) code to access requested information via the auxiliary application, the accessed requested information being supplied to said dispatcher; and

a response generator receiving the accessed requested information from the dispatcher for generating a response as may be applicable to the user's request.

2. The multimodal natural language interface recited in claim 1 further comprising a dialog manager providing feedback to the user indicating the system's understanding of the input and interacting with the user to clarify the request, if necessary.

3. The multimodal natural language interface recited in claim 2 wherein said dispatcher forms the combined multimodal request by concatenating the user natural language input with the input information selected from the current application running on the system.

4. The multimodal natural language interface recited in claim 3 wherein the application manager includes a concept/application registration table, said application manager finding all concepts in the semantic representation from the natural language processor and then finding all applications registered in said concept/application registration table for those concepts.

5. A method implemented in a computer system for interpreting user requests by combining natural language input from a user with information selected from a current application running on the computer system comprising the steps of:

receiving a natural language input from the user and combining the natural language input with input infor-

8

mation selected from a current application to for a combined multimodal request;

parsing the combined multimodal request;

performing semantic interpretation of the parsed combined multimodal request to generate a semantic representation of the combined multimodal request;

determining of which auxiliary application should process the request;

invoking the auxiliary application and generating application program interface (API) code to access requested information via the auxiliary application; and receiving the accessed requested information and generating a response as may be applicable to the user's request.

6. The method recited in claim 5 further comprising the step of providing feedback to the user indicating the system's understanding of the input and interacting with the user to clarify the request, if necessary.

7. The method recited in claim 6 wherein the step of combining is performed by concatenating the user natural language input with the input information selected from the current application running on the system.

8. The method recited in claim 7 further comprising the steps of:

generating a concept/application registration table; finding all concepts in the semantic representation; and then finding all applications registered in said concept/application registration table for those concepts.

* * * * *



US006188985B1

(12) **United States Patent**
Thrift et al.

(10) **Patent No.:** **US 6,188,985 B1**
(45) **Date of Patent:** **Feb. 13, 2001**

(54) **WIRELESS VOICE-ACTIVATED DEVICE FOR CONTROL OF A PROCESSOR-BASED HOST SYSTEM**

5,796,394	*	8/1998	Wicks et al.	345/329
5,802,526	*	9/1998	Fawcett et al.	707/104
5,890,122	*	3/1999	Van Kleeck et al.	704/275
5,890,123	*	3/1999	Brown et al.	704/275
6,075,575	*	6/2000	Schein et al.	348/734

(75) Inventors: **Philip R. Thrift**, Dallas; **Charles T. Hemphill**, Allen, both of TX (US)

OTHER PUBLICATIONS

(73) Assignee: **Texas Instruments Incorporated**, Dallas, TX (US)

Holmes "Speech Synthesis and Recognition" Chapman Hill, p. 109, 1988.*

(*) Notice: Under 35 U.S.C. 154(b), the term of this patent shall be extended for 0 days.

Ballou "Handbook for Sound Engineers" Howard Sams, p. 376, 1987.*

(21) Appl. No.: **08/943,795**

Dragon "Dragon Dictate 1.0 for Windows" Dragon systems, pp. 140, 13.*

(22) Filed: **Oct. 3, 1997**

* cited by examiner

Related U.S. Application Data

Primary Examiner—David R. Hudspeth

(60) Provisional application No. 60/034,685, filed on Jan. 6, 1997.

Assistant Examiner—Harold Zintel

(51) **Int. Cl.**⁷ **G10L 15/00**; H04N 5/44

(74) *Attorney, Agent, or Firm*—Robert L. Troike; Frederick J. Telecky, Jr.

(52) **U.S. Cl.** **704/275**; 348/734

(57) **ABSTRACT**

(58) **Field of Search** 704/275, 270; 348/734, 738

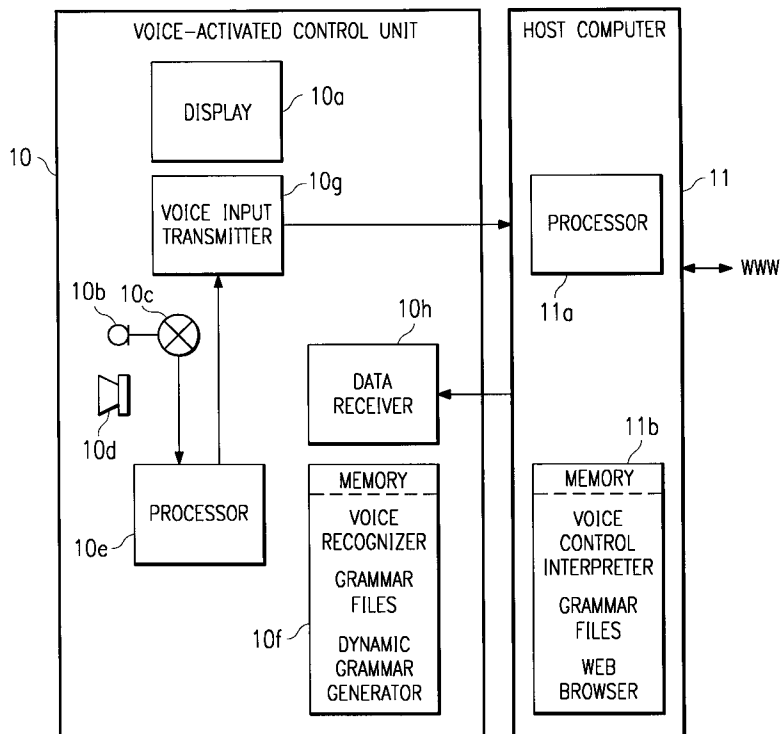
A hand-held wireless voice-activated device (10) for controlling a host system (11), such as a computer connected to the World Wide Web. The device (10) has a display (10a), a microphone (10b), and a wireless transmitter (10g) and receiver (10h). It may also have a processor (10e) and memory (10f) for performing voice recognition. A device (20) can be specifically designed for Web browsing, by having a processor (20e) and memory (20f) that perform both voice recognition and interpretation of results of the voice recognition.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,661,659	*	4/1987	Nishimura	455/462
5,199,080	*	3/1993	Kimura et al.	381/110
5,247,580	*	9/1993	Kimura et al.	704/275
5,636,211	*	6/1997	Newlin et al.	370/465
5,737,491	*	4/1998	Allen et al.	704/270
5,774,628	*	6/1998	Hemphill	704/255

18 Claims, 3 Drawing Sheets



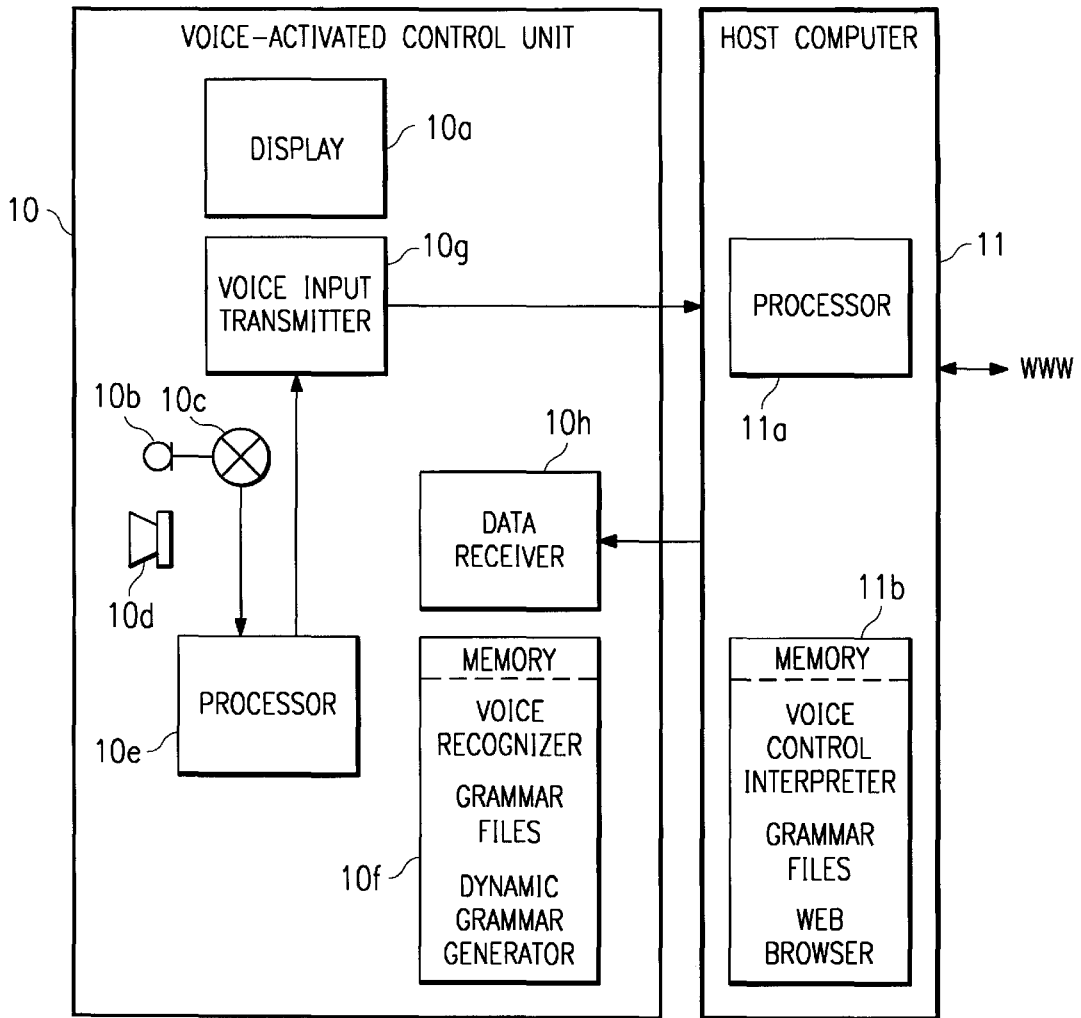


FIG. 1

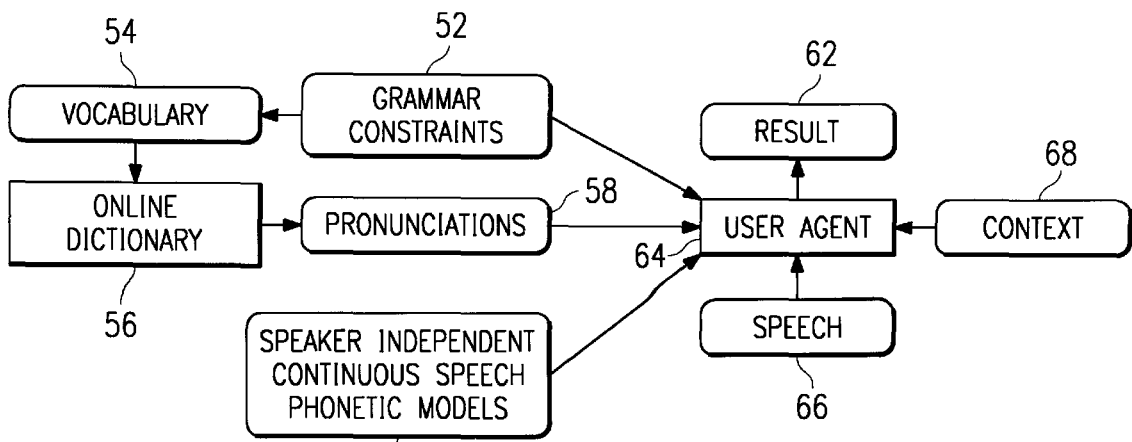


FIG. 5

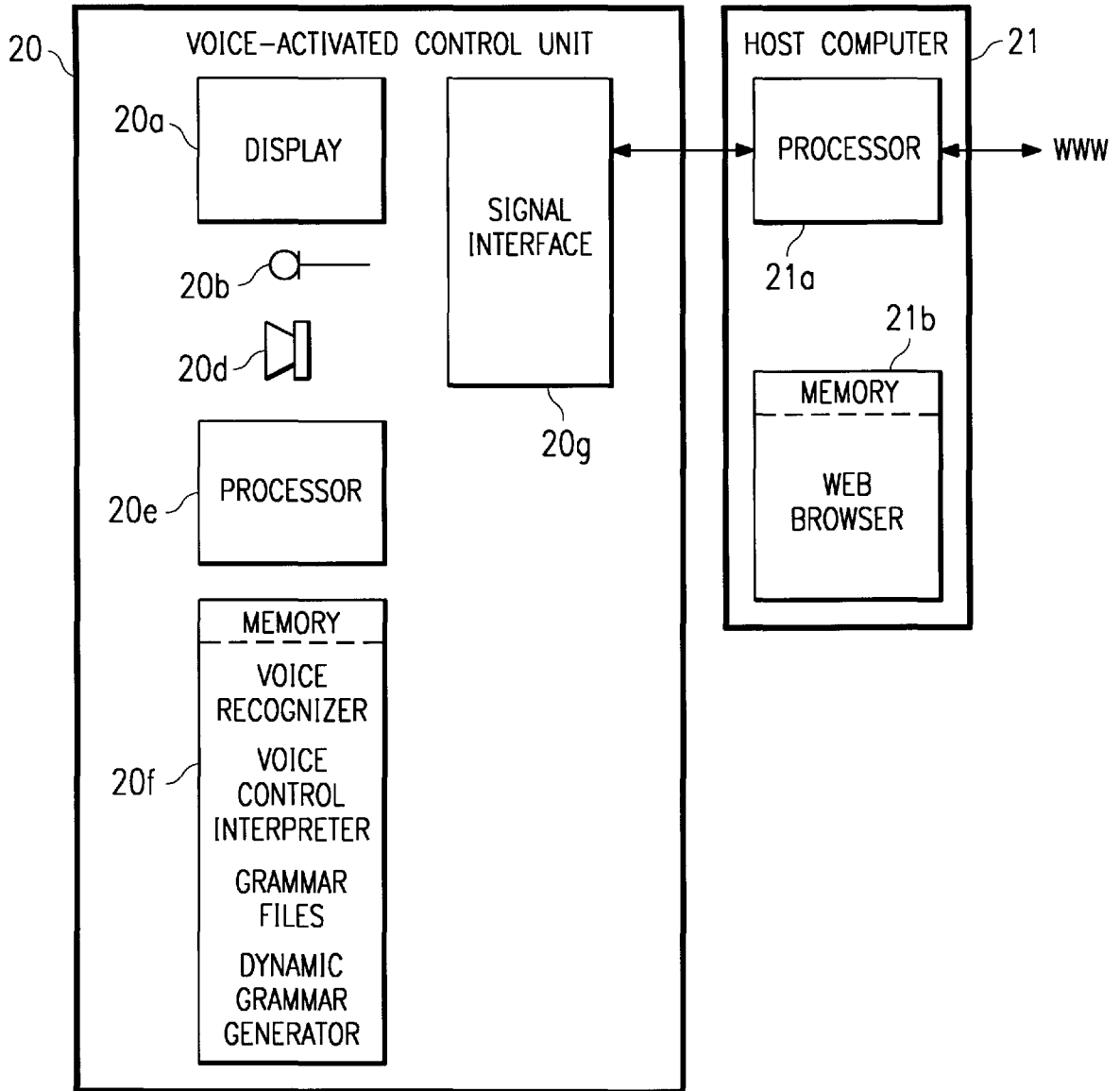


FIG. 2

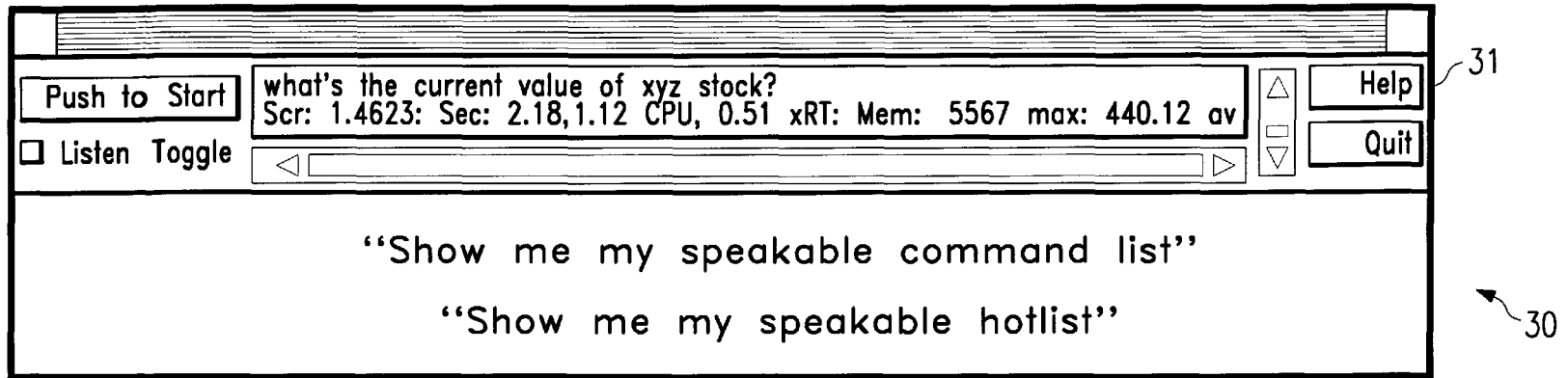


FIG. 3

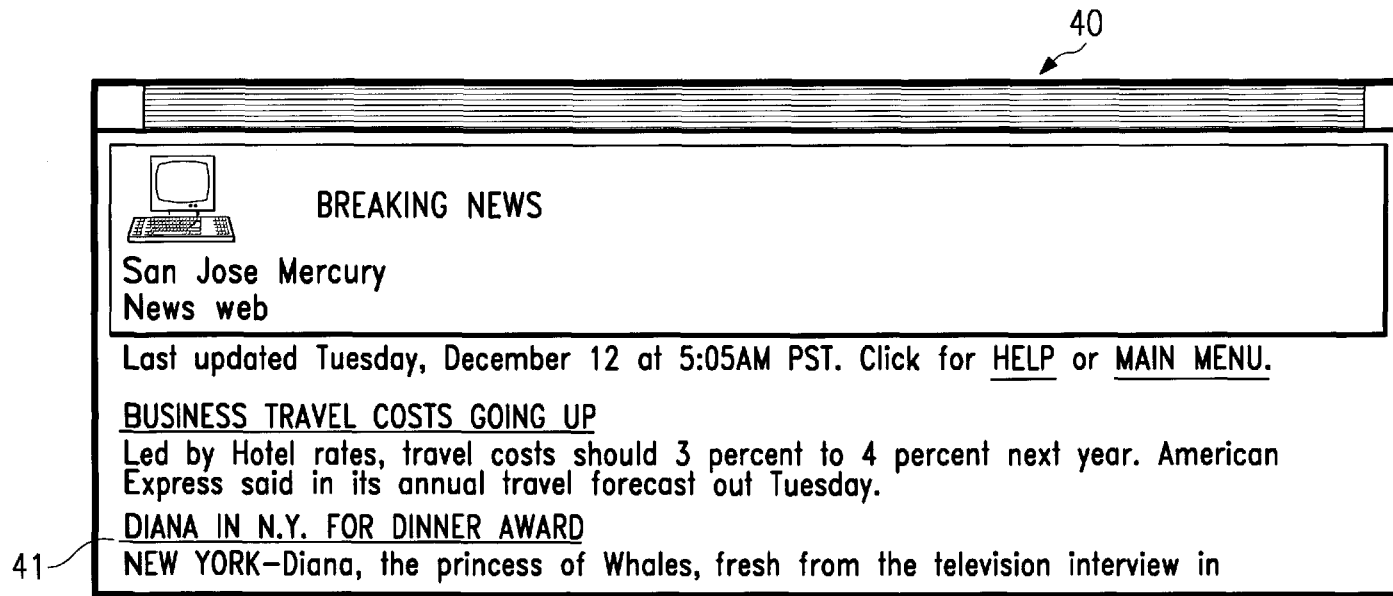


FIG. 4

1

WIRELESS VOICE-ACTIVATED DEVICE FOR CONTROL OF A PROCESSOR-BASED HOST SYSTEM

This application claims benefit of Ser. No. 60/034,685
filed Jan. 6, 1997.

TECHNICAL FIELD OF THE INVENTION

The present invention relates generally to voice recogni-
tion devices, and more particularly to a wireless voice-
controlled device that permits a user to browse a hypermedia
network, such as the World Wide Web, with voice com-
mands.

RELATED PATENT APPLICATIONS

This patent application is related to the following patent
applications, each assigned to Texas Instruments Incorpo-
rated.

U.S. patent U.S. Pat. No. 5,774,628 entitled "Speaker-
Independent Dynamic Vocabulary and Grammar in
Speech Recognition"

U.S. patent application Ser. No. 08/419,229, entitled
"Voice Activated Hypermedia Systems Using Gram-
matical Metadata"

BACKGROUND OF THE INVENTION

The Internet is a world-wide computer network, or more
accurately, a world-wide network of networks. It provides an
exchange of information and offers a vast range of services.
Today, the Internet has grown so as to include all kinds of
institutions, businesses, and even individuals at their homes.

The World-Wide Web ("WWW" or "Web") is one of the
services available on the Internet. It is based on a technology
known as "hypertext", in which a document has links to its
other parts or to other documents. Hypertext has been
extended so as to encompass links to any kind of information
that can be stored on a computer, including images and
sound. For example, using the Web, from within a document
one can select highlighted words or phrases to get definitions,
sources, or related documents, stored anywhere in the world.
For this reason, the Web may be described as a "hyperme-
dia" network.

The basic unit in the Web is a "page", a (usually)
text-plus-graphics document with links to other pages.
"Navigating" the Web primarily means moving around from
page to page.

The idea behind the Web is to collect all kinds of data
from all kinds of sources, avoiding the problems of incom-
patibilities by allowing a smart server and a smart client
program to deal with the format of the data. This capability
to negotiate formats enables the Web to accept all kinds of
data, including multimedia formats, once the proper trans-
lation code is added to the servers and clients. The Web
client is used to connect to and to use Web resources located
on Web servers.

One type of client software used to access and use the
Web is referred as "web browser" software. This software
can be installed on the user's computer to provide a graphic
interface, where links are highlighted or otherwise marked
for easy selection with a mouse or other pointing device.

SUMMARY OF THE INVENTION

One aspect of the invention is a wireless voice-activated
control unit for controlling a processor-based host system,

2

such as a computer connected to the World Wide Web. A
compact hand-held unit has a microphone, a wireless audio
input transmitter, a wireless data receiver, and a display. The
microphone receives voice input from a user, thereby pro-
viding an audio input signal. The audio transmitter wire-
lessly transmits data derived from the audio signal to the
host system. After the host acts on the audio input, it delivers
some sort of response in the form of image data wirelessly
delivered to the receiver. A display generates and displays
images represented by the image data.

Variations of the device can include a speaker for audio
output information. The device can also have a processor
and memory for performing front-end voice recognition
processes or even all of the voice recognition.

An advantage of the invention is that it makes information
on the Web more accessible and useful. Speech control
brings added flexibility and power to the Web interface and
makes access to information more natural.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates one embodiment of a wireless voice-
activated control unit in accordance with the invention.

FIG. 2 illustrates another embodiment of a wireless voice-
activated control unit, specially configured for translating and
interpreting audio input from the user.

FIG. 3 illustrates an example of a display provided by the
speakeable command process.

FIG. 4 illustrates a portion of a Web page and its speak-
able links.

FIG. 5 illustrates a process of dynamically creating gram-
mars for use by the voice recognizer of FIGS. 1 and 2.

DETAILED DESCRIPTION OF THE INVENTION

The invention described herein is directed to a wireless
voice-activated device for controlling a processor-based host
system. That is, the device is a voice-activated remote
control device. In the example of this description, the host
system is a computer connected to the World-Wide Web and
the device is used for voice-controlled web browsing.
However, the same concepts can be applied to a voice-
controlled device for controlling any processor-based sys-
tem that provides display or audio information, for example,
a television.

Various embodiments of the device differ with regard to
the "intelligence" embedded in the device. For purposes of
the invention, the programming used to recognize an audio
input and to interpret the audio input so that it can be used
by conventional web browser software is modularized in a
manner that permits the extent of embedded programming to
become a matter of design and cost.

FIG. 1 illustrates one embodiment of a wireless voice-
activated control unit **10** in accordance with the invention. It
communicates with a host system **11**. As stated above, for
purposes of this description, host system **11** is a computer
and is in data communication with the World-Wide Web.

Control unit **10** has a display **10a** and a microphone **10b**.
Display **10a** is designed for compactness and portability, and
could be an LCD. Microphone **10b** receives voice input from
a user. It may have a "mute" switch **10c**, so that control unit
10 can be on, displaying images and even receiving non-
audio input via an alternative input device such as a keypad
(not shown), but not performing voice recognition. Micro-
phone **10b** may be a microphone array, to enhance the ability
to differentiate the user's voice from other sounds.

In the embodiment of FIG. 1, control unit 10 performs all or part of the voice recognition process and delivers speech data to host computer 11 via transmitter 10g. Host computer 11 performs various voice control interpretation processes and also executes a web browser. However, in its simplest form control unit would transmit audio data directly from microphone 10b to host system 11, which would perform all processing.

In the case where control unit 10 performs all or part of the voice recognition process, control unit 10 has a processor 10e. Memory 10f stores voice recognition programming to be executed by processor 10e. An example of a suitable processor 10a for speech recognition is a signal processor, such as those manufactured by Texas Instruments Incorporated. Where microphone 10b is a microphone array, processor 10a may perform calculations for targeting the user's voice.

If control unit performs only some voice processing, it may perform one or more of the "front end" processes, such as linear predictive coding (LPC) analysis or speech end pointing.

If control unit 10 performs all voice recognition processes, memory 10f stores these processes (as a voice recognizer) as well as grammar files. In operation, the voice recognizer receives audio input from microphone 10b, and accesses the appropriate grammar file. A grammar file handler converts the grammar to speech-ready form, creating a punctuation grammar, and loading the grammar into the voice recognizer. The voice recognizer uses the grammar file to convert the audio input to a text translation.

The grammar files in memory 10f may be pre-defined and stored or may be dynamically created or may be a combination of both types of grammar files. An example of dynamic grammar file creation is described below in connection with FIG. 5. The grammars may be written with the Backus-Naur form of context-free grammars and can be customized. In the embodiment of FIG. 1, and where unit 10 is used for Web browsing, host computer 11 delivers the HTML (hypertext markup language) for a currently displayed Web page to unit 10. Memory 10f stores a grammar file generator for dynamically generating the grammar. In alternative Web browsing embodiments, host 11 could dynamically generate the grammar and download the grammar file to control unit 10.

The output of the voice recognizer is speech data. The speech data is transmitted to host system 11, which performs voice control interpretation processes. Various voice control interpretation processes for voice-controlled Web browsing are described in U.S. patent application Ser. No. 08/419,229, entitled "Voice Activated Hypermedia Systems Using Grammatical Metadata", assigned to Texas Instruments Incorporated and are incorporated herein by reference. As a result of the interpretation, the host system 11 may respond to the voice input to control unit 10 by executing a command or providing a hypermedia (Web) link.

An example of voice control interpretation other than for Web browsing is for commands to a television, where host system 11 is a processor-based television system. For example, the vocal command, "What's on TV tonight?", would result in a display of the television schedule. Another example of voice control interpretation other than for Web browsing is for commands for computer-based household control. The vocal command, "Show me the sprinkler schedule" would result in an appropriate display.

After host system 11 has taken the appropriate action, a wireless receiver 10h receives data from host system 11 for

display on display 10a or for output by speaker 10d. Thus, the data received from host system 11 may be graphical (including text, graphics, images, and videos or audio).

FIG. 2 illustrates an alternative embodiment of the invention, a wireless voice-activated control unit 20 that performs voice control interpretation as well as voice recognition. The voice control interpretation is specific to browsing a hypermedia resource, such as the Web. The host system 21 is connected to the hypermedia resource.

Control unit 20 has components similar to those of control unit 10. However, its processor 20e performs additional programming stored in memory 20f. Specifically, the voice control interpretation processes may comprise a speakable command process, a speakable hotlist process, or a speakable links process. These processes and their associated grammar files reside on control unit 20.

The speakable command process displays a command interface on display 20a and accepts various Web browsing commands. The process has an associated grammar file for the words and phrases that may be spoken by the user.

FIG. 3 illustrates an example of a display 30 provided by the voice control interpretation process. One speakable command is a "Help" command, activated with a button 31. In response, the command process displays a "help page" that describes how to use voice-controlled browsing.

Another speakable command is, "Show me my speakable command list". Speaking this command displays a page listing a set of grammars, each representing a speakable command. Examples are `pagedown_command`, `back_command`, and `help_command`. When the command process receives a translation of one of these commands, it performs the appropriate action.

FIG. 3 also illustrates a feature of the voice recognizer that is especially useful for Web browsing. The user has spoken the words, "What is the value of XYZ stock?" Once the voice recognizer recognizes an utterance, it determines the score and various statistics for time and memory use. As explained below, the request for a stock value can be a hotlist item, permitting the user to simply voice the request without identifying the Web site where the information is located.

Another speakable command is "Show me my speakable hotlist", activated by button 33. A "hotlist" is a stored list of selected Uniform Resource Locators (URLs), such as those that are frequently used. Hotlists are also known as bookmarks. URLs are a well known feature of the Web, and provide a short and consistent way to name any resource on the Internet. A typical URL has the following form:

`http://www.ncsa.uiuc.edu/General/NCSAHome.html`

The various parts of the URL identify the transmission protocol, the computer address, and a directory path at that address. URLs are also known as "links" and "anchors".

The speakable hotlist process permits the user to construct a grammar for each hotlist item and to associate the grammar with a URL. To create the grammar, the user can edit an ASCII grammar file and type in the grammar using the BNF syntax. For example, a grammar for retrieving weather information might define phrases such as, "How does the weather look today?" and "Give me the weather". The user then associates the appropriate URL with the grammar.

The hotlist grammar file can be modified by voice. For example, a current page can be added as a hotlist item. Speaking the phrase, "Add this page to my hotlist" adds the title of the page to the grammar and associates that grammar with the current URL. Speaking the phrase, "Edit my speakable hotlist", permits the user to edit the grammar by adding additional phrases that will cause the page to be retrieved by voice.

The speakable hotlist process is activated when the voice recognizer recognizes a hotlist translation from the hotlist grammar file and passes the translation to the hotlist process. The hotlist process looks up the associated URL. It passes the URL to the browser residing on host computer 11 (via wireless communication), so that the Web page may be retrieved and transmitted to the voice control unit 10 for display on display 10a.

The grammar files for speakable commands and the speakable hotlist are active at all times. This permits the user to speak the commands or hotlist links in any context. A speakable links process may also reside in memory 20e of voice control unit 20. Selected information in a Web page may provide links, for access to other web pages. Links are indicated as such by being underlined, highlighted, differently colored, outlined as in the case of pictures, or otherwise identified. Instead of using a mouse or other pointing device to select a link, the user of voice control unit 10 may speak a link from a page being display on display 10a.

FIG. 4 illustrates a portion of a Web page 40 and its links. For example, the second headline 41 is a link.

The grammar for speakable links includes the full phrase as well as variations. In addition to speaking the full phrase, the speaker may say "Diana in N period Y period" (a literal variation), "Diana in NY", or "Diana in New York".

Making a link speakable first requires obtaining the link/URL pair from its Web page. Because a Web page in HTML (hypertext markup language) format can have any length, the number of candidate link/URL pairs that the recognizer searches may be limited to those that are visible on a current screen of display 20a. A command such as, "Scroll down", updates the candidate link/URL pairs. Once the link/URL pairs for a screen are obtained, a grammar is created for the all the links on the current screen. Next, tokens in the links are identified and grammars for the tokens are created. These grammars are added to the recognizer's grammar files. Correct tokenization is challenging because link formats can vary widely. Links can include numbers, acronyms, invented words, and novel uses of punctuation.

Other challenges for speakable links are the length of links, ambiguity of links in the same page, and graphics containing bit-mapped links. For long links, the speakable links process permits the user to stop speaking the words in a link any time after N words. For ambiguity, the process may either default to the first URL or it may offer a choice of URLs to the user. For bit-mapped links, the process uses an <ALT> tag to look for link information.

The grammars for speakable links may be dynamically created so that only the grammar for a current display is active and is updated when a new current display is generated. Dynamic grammar creation also reduces the amount of required memory 10f.

FIG. 5 illustrates a suitable process of dynamically creating grammar files. This is the process implemented by the dynamic grammar generator of FIGS. 1 and 2. As explained above, dynamic grammar files are created from current Web pages so that speakable links may be recognized. U.S. patent U.S. Pat. No. 5,774,628, incorporated by reference above, further describes this method as applied to a voice-controlled host system 11, that is, voice control without a separate remote control device 10.

A display, such as the display 40 of FIG. 4, affects grammar constraints 52. The grammar constraints 52 are input into a vocabulary 54 and the user agent 64. In turn, the vocabulary 54 feeds the online dictionary 56, which inputs into the pronunciations module 58. The pronunciations module 58, as well as the Speaker Independent Continuous

Speech Phonetic Models module 60, input into the User Agent 64. In addition, the Speech module 66 inputs the user's speech into the User Agent 64. In parallel, the Context module 68 gets inputs from the screen 40 and inputs into the User Agent 64.

An existing RGDAG (Regular Grammar Directed Acyclic Graph) may dynamically accommodate new syntax and vocabulary. Every time the screen 40 changes, the user agent 64 creates a grammar containing the currently visible underlined phrases (links). From this grammar, the user agent 64 tokenizes the phrases to create phrase grammars that can include, for example, optional letter spelling and deleted/optional punctuation. From the tokens, the user agent 64 creates phonetic pronunciation grammars using a combination of online dictionaries and a text-to-phoneme mapping. The voice recognition process then adds the grammars created. This involves several simple bookkeeping operations for the voice recognizer, including identifying which symbols denote "words" to output. Finally, global changes are implemented to incorporate the new/changed grammars. For this, the grammars are connected in an RGDAG relationship. In addition, the maximum depth for each symbol is computed. It is also determined whether the voice recognizer requires parse information by looking for ancestor symbols with output. Then the structure of the grammar for efficient parsing is identified.

Other Embodiments

Although the invention has been described with reference to specific embodiments, this description is not meant to be construed in a limiting sense. Various modifications of the disclosed embodiments, as well as alternative embodiments, will be apparent to persons skilled in the art. It is, therefore, contemplated that the appended claims will cover all modifications that fall within the true scope of the invention.

What is claimed is:

1. A wireless voice-activated control system comprising:
 - a remote processor-based host system;
 - a voice recognition processor operable to perform a voice recognition process and a memory that stores said voice recognition process and grammar files; and
 - a voice activated control unit for remotely controlling said remote processor-based host system comprising:
 - a microphone operable to receive voice command input from a user, thereby providing an audio input signal; said microphone operably coupled to said voice recognition processor, said memory and said grammar files for voice recognition of said voice commands;
 - an audio transmitter operable to wirelessly transmit data derived from said audio input signal to said host system to control said host system;
 - a data receiver operable to wirelessly receive image data from said host system representing voice commanded display images; and
 - a display operable to generate and display said voice commanded images represented by said image data.
2. The control unit of claim 1, wherein said microphone is switchable to an on or off state separately from said display.
3. The control unit of claim 1, wherein said microphone is a multi-element microphone array.
4. The system of claim 1, wherein said voice recognition process comprises linear predictive coding analysis, and wherein said transmitter is operable to transmit the results of said analysis.
5. The system of claim 1, wherein said grammar files are dynamically created, wherein said processor is further operable to perform a dynamic grammar generation process.

7

6. The system of claim 1, wherein said voice recognition processor comprises speech and pointing analysis and wherein said transmitter is operable to transmit the result of said analysis.

7. A wireless voice-activated control system for voice-control of a remote processor-based host system in data communication with a hypermedia resource to permit a user to browse a hypermedia network, comprising:

said remote processor-based host system including a web browser and in data communication with a hypermedia resource;

a voice recognition processor operable to perform a voice recognition process and a memory that stores said voice recognition process and grammar files; and

a voice-activated control unit for remotely controlling said remote processor-based host system comprising:

a microphone operable to receive voice browser commands input from a user, thereby generating an audio input signal; said microphone operably coupled to said voice recognition processor, said memory and said grammar files for voice recognition of said voice commands;

an audio transmitter operable to wirelessly transmit data representing browser commands derived from said audio input signal to said remote processor-based host system to cause said host system to browse said hypermedia network and retrieve selected web page;

a data receiver operable to wirelessly receive image data representing a selected web page from said remote host system; and

a display operable to generate and display web page images represented by said image data and retrieved from said hypermedia resource by said host system.

8. The system of claim 7, wherein said voice recognition processor, said memory, and said grammar files are in said control unit.

8

9. The system of claim 7, wherein said voice recognition processor comprises linear predictive coding analysis, and wherein said transmitter is operable to transmit the results of said analysis.

10. The system of claim 7, wherein said voice recognition processor comprises speech end pointing analysis, and wherein said transmitter is operable to transmit the results of said analysis.

11. The system of claim 7, wherein said grammar files are dynamically created, wherein said processor is further operable to perform a dynamic grammar generation process.

12. The system of claim 7, further comprising a processor operable to perform voice control processes and a memory that stores said voice control processes.

13. The system of claim 12, wherein said voice control processor comprise a speakable commands process such that said user may vocally direct the operations of said host system.

14. The system of claim 12, wherein said voice control processor comprise a speakable hotlist process such that said user may vocally request a particular one of said resources to be retrieved by said host system.

15. The of claim 12, wherein said voice control processes comprise a speakable links process such that said user may vocally request that a link on a current page being displayed on said display be retrieved by said host system.

16. The system of claim 7, further comprising a processor operable to perform dynamic grammar creation processes, and memory that stores said processes.

17. The system of claim 7, wherein said host system performs voice control processes.

18. The system of claim 7, wherein audio data from the microphone is sent to the host system which performs all processing.

* * * * *



US006345389B1

(12) **United States Patent**
Dureau

(10) **Patent No.:** **US 6,345,389 B1**
(45) **Date of Patent:** ***Feb. 5, 2002**

(54) **INTERACTIVE TELEVISION SYSTEM AND METHOD FOR CONVERTING NON-TEXTUAL INFORMATION TO TEXTUAL INFORMATION BY A REMOTE SERVER**

FOREIGN PATENT DOCUMENTS

EP	0 633 661	1/1995
EP	0 689 155	7/1996
EP	0 838 945	4/1998

(75) Inventor: **Vincent Dureau**, Palo Alto, CA (US)
(73) Assignee: **OpenTV, Inc.**, Mountainview, CA (US)

OTHER PUBLICATIONS

“Speech Recognition Methods for Controlling Cable Television,” IBM Technical Disclosure Bulletin, vol. 38, No. 8, Aug. 1995, pp. 285–287.

(*) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

International Search Report, Application No. PCT/US99/24710, mailed Feb. 15, 2000.

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

* cited by examiner

Primary Examiner—Chris Grant

(21) Appl. No.: **09/176,611**
(22) Filed: **Oct. 21, 1998**

(74) Attorney, Agent, or Firm—Conley, Rose & Tayon, PC

(51) **Int. Cl.**⁷ **H04N 7/173**
(52) **U.S. Cl.** **725/116; 725/133**
(58) **Field of Search** 348/1, 2, 6, 10, 348/12, 13, 14; 455/2, 3.1, 6.1, 5.1, 6.2, 6.3; 345/327; 382/187, 189; 704/235; 725/131, 132, 116, 139, 140, 151, 152, 133

ABSTRACT

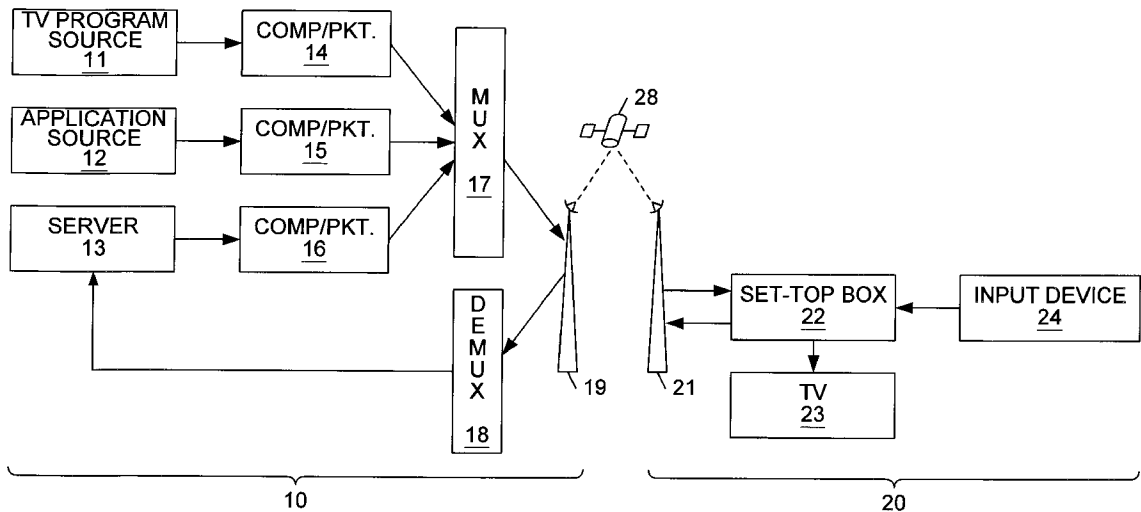
A system and method for providing user input to an application executing on an interactive television system wherein a user provides non-textual information to the interactive television system and this information is converted by a server to textual information which is provided to the application. In one embodiment, a digitizer pad coupled to a set-top box is used to digitize the user’s handwriting. The digitized information is conveyed to a remote server which converts the digitized handwriting data into textual information. The textual information is conveyed to the set-top box, where it is input to an application executing on the set-top box. In another embodiment, a microphone is coupled to a set-top box. The microphone allows the user to input voice information which is digitized and conveyed to the server for conversion into textual information. The textual information is conveyed back to the set-top box and is input to an application executing on the set-top box.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,528,743	A	*	6/1996	Tou et al.	707/541
5,546,538	A	*	8/1996	Cobbley et al.	395/200.01
5,812,931	A	*	9/1998	Yuen	455/5.1
5,831,664	A	*	11/1998	Wharton et al.	348/13
5,861,881	A	*	1/1999	Freeman et al.	345/302
5,875,448	A	*	2/1999	Boys et al.	707/531

22 Claims, 3 Drawing Sheets



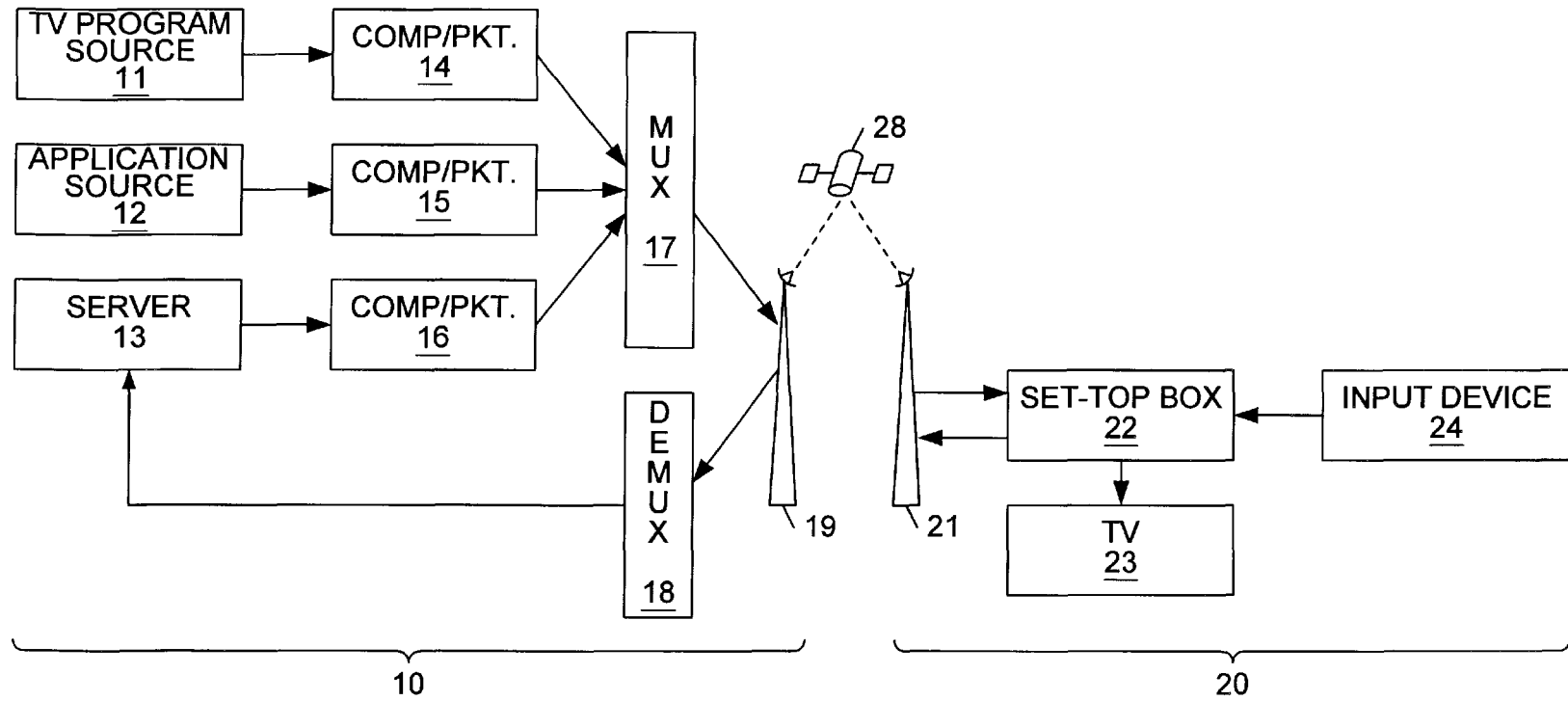


FIG. 1

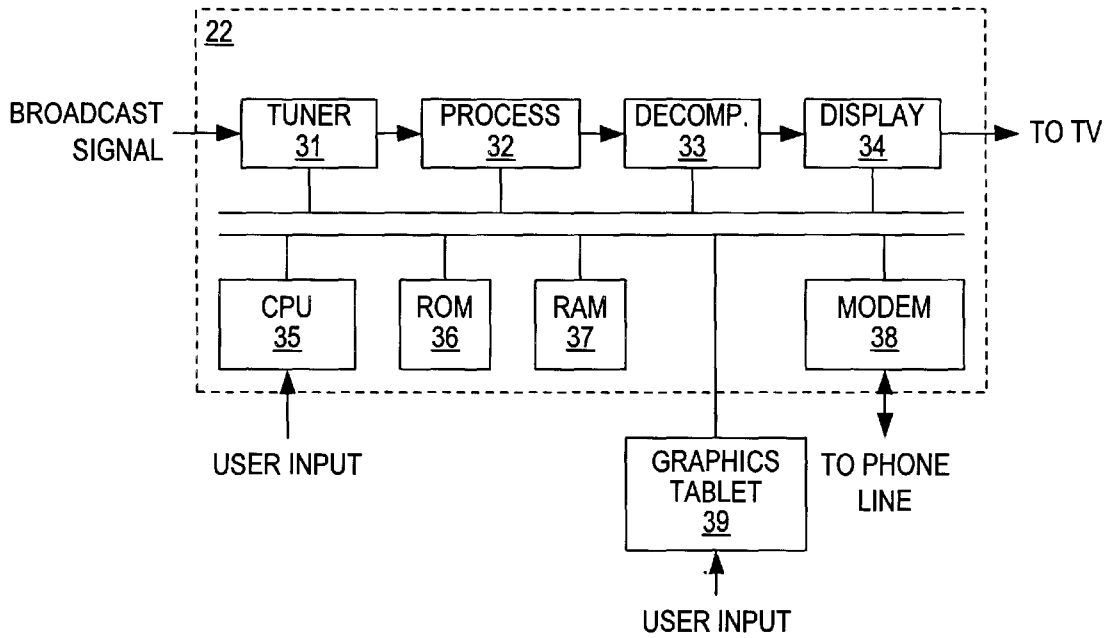


FIG. 2

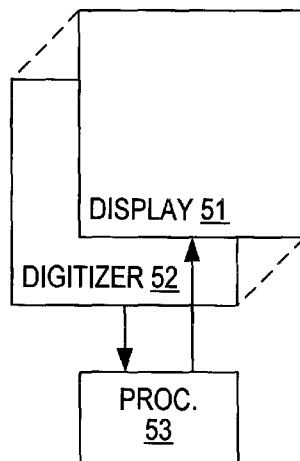


FIG. 3

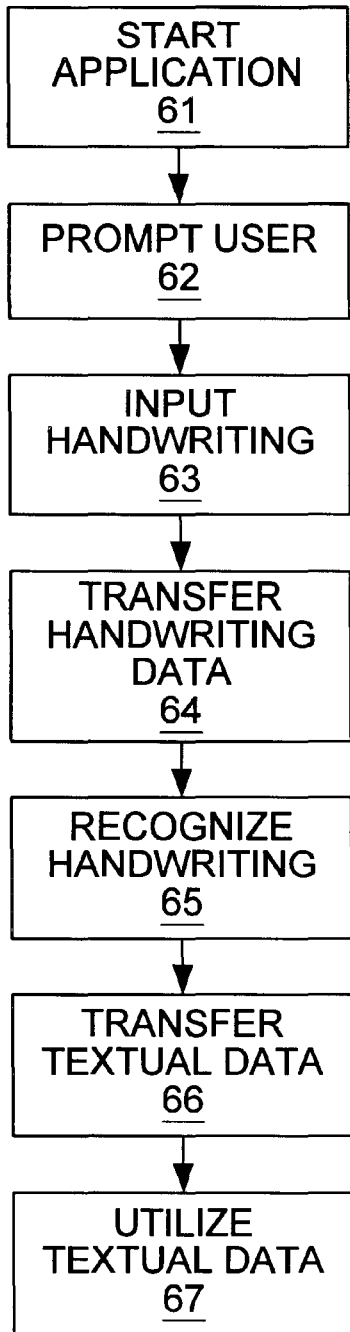


FIG. 4

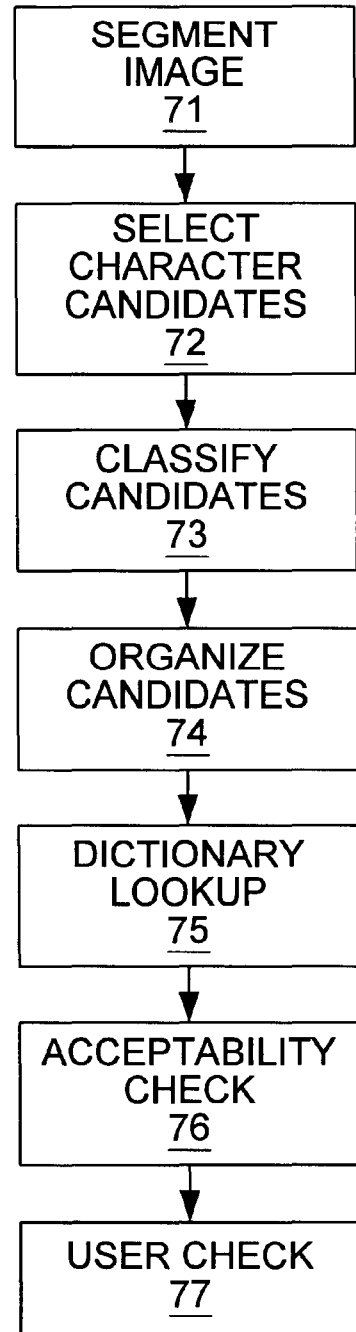


FIG. 5

1

INTERACTIVE TELEVISION SYSTEM AND METHOD FOR CONVERTING NON-TEXTUAL INFORMATION TO TEXTUAL INFORMATION BY A REMOTE SERVER

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates generally to interactive television systems and more particularly to means and methods for using a server to convert user-provided information into a data format which can be used by an interactive television application.

2. Description of the Relevant Art

Interactive television systems can be used to provide a variety of services to users. These systems are capable of displaying text and graphic images to facilitate viewer interaction as well as the audio and video streams associated with ordinary television programs. Interactive television systems enable viewer interaction and thereby allow the systems to be used for marketing and educational purposes in addition to extending the entertainment capabilities of an ordinary television. Viewers can order advertised products or services, request information regarding particular programs, or send electronic messages (e-mail).

In a typical interactive television system, a broadcast service provider generates an interactive television signal for transmission to a viewer's television. The interactive television signal includes an audio-video portion consisting of a television program, as well as an interactive portion consisting of application code or control information. The broadcast service provider combines the audio-video and interactive portions into a single signal for transmission to a receiver connected to the user's television. The signal is typically compressed prior to transmission and transmitted through broadcast channels such as cable television (CATV) lines or direct satellite transmission systems.

The interactive functionality of the interactive television system is controlled by a set-top box connected to the television. The set-top box receives the signal transmitted by the broadcast service provider, separates the interactive portion from the audio-video portion and decompresses the respective portions of the signal. If the interactive portion of the signal comprises an interactive application, the application may be executed while the audio-video information (the normal television signal) is conveyed to the television. The set-top box may combine the audio-video information with interactive graphics or audio generated by the interactive application prior to conveying the information to the television. The interactive graphics and audio may present additional information to the viewer or may prompt the viewer for input and may be designed to function in response to signals in the broadcast or actions taken by the viewer. An interactive television system may also run applications that present the user only with the audio and graphics relating to the interactive application. In other words, this information may be displayed in place of the normal television signal, rather than adding to the signal.

There are various examples of interactive applications which require viewer input. One example is a televised buying service in which a series of products are displayed via a television program and the viewer utilizes the interactive functions of the television system to purchase the displayed products. The viewer must identify the product he or she wishes to purchase and may also need to provide size, color, and other information descriptive of the product. This information may be provided by selecting the appropriate

2

choice from a menu. The application may, however, also require the user's name or shipping information which cannot simply be selected from a menu. Another example of an application which requires textual information is a messaging application for which the viewer must provide the message, as well as information identifying the addressee of the message. Some means is therefore necessary to enter this textual information.

One option for entering text would be to use a keyboard. This might not be the best solution, however, because some users may not feel comfortable using a keyboard. One of the attractions of interactive television systems is the ease with which they may be used. Interactive television applications are designed to provide simplified user interfaces and many require no more input than can be provided using a basic remote control. The requirement of entering textual information via a keyboard may make the interactive television system less attractive to users because of the additional hardware which is required for the system, the perceived complexity of the system or the difficulty which may be experienced by some users in typing the necessary information. Further, in some languages (e.g., Chinese), the complexity of the written language makes text entry via a keyboard difficult even for experienced users.

Other options for providing textual information to an application may involve means for converting non-textual information into a textual form. For example, using voice recognition technology, a user's voice can be sampled and compared to previously sampled speech patterns to determine the words spoken by the user. The words can then be output by the speech recognition system as text. Handwriting recognition systems could also be used to generate textual information for the application. These systems perform the same process on images of the user's handwriting (entered via a graphics tablet or similar input device) to determine the text written by the user. While these technologies have evolved to the point that they can reliably generate accurate textual information from the user's voice or handwriting, they require a great deal of computer resources. The applications may be very large and they need large amounts of processing time to perform the required pattern matching. These technologies therefore cannot be implemented in current set-top boxes, which have very limited storage and processing capacity.

SUMMARY OF THE INVENTION

One or more of the problems outlined above may be solved by the various embodiments of the invention. The invention comprises a system and method for enabling a user to provide non-textual information which is converted by the system to a textual form in which it can be used by the interactive application. The non-textual information is entered by the user at the set-top box of a receiving station and this information is transmitted to a server which may be located at a broadcast station. The server converts the information into textual data so that it can be used by the system. In one embodiment, the server transmits the textual data back to the receiving station, where it can be used by an application executing in the set-top box. In other embodiments, the textual data can be used at the server or transmitted to a part of the system other than the set-top box.

One embodiment comprises an interactive television system comprising a broadcast station and a receiving station. The broadcast station transmits an interactive television application to the receiving station, which then executes the application. The application requires textual data from the

user. The receiving station includes a set-top box which executes the application and a graphics tablet which is coupled to the set-top box for entering information. The user provides the information by writing on the graphics tablet, which generates an image file. The image file is transmitted from the set-top box to the broadcast station. The broadcast station includes a server which uses handwriting recognition software to convert the image file into character or textual data. The textual data is transmitted back to the application at the set-top box, which uses the data as if it had been typed in directly by the user.

Because the handwriting recognition software resides on the server computer instead of the set-top box, more resources are available for execution of the software. The server typically has more available memory and more processing power than the set-top box and consequently provides much faster recognition of handwriting images. The greater available resources enable the software to provide for recognition of additional, complex languages (e.g., Chinese) and extended character sets (e.g., unicode). Since the recognition software is maintained on the server, the software can be quickly and easily updated with the latest handwriting recognition technology, and it does not have to be distributed to the individual subscribers' set-top boxes.

In one embodiment, the textual data need not be sent back to the set-top box, but may instead be used at the server or another location remote from the set-top box. For example, a user may place an order for a product by entering oral or handwritten information and the order may be confirmed later by e-mail or other means. In another embodiment, the server could consist of a human operator who receives the audio or image data and manually transcribes it into textual information. The audio or image data may alternately be converted into textual information by automated means operating under human supervision.

In an alternate embodiment, the interactive television system includes a microphone rather than a graphics tablet for entry of information to the system. The microphone is used to provide voice data, which is recorded and transmitted to a server equipped with a voice recognition application. The voice recognition application converts the voice data into textual data, which is then transmitted back to the application executing on the set-top box.

BRIEF DESCRIPTION OF THE DRAWINGS

Other objects and advantages of the invention will become apparent upon reading the following detailed description and upon reference to the accompanying drawings in which:

FIG. 1 is a block diagram illustrating one embodiment of a system for distribution of interactive television applications and television programs from their sources to a series of viewers.

FIG. 2 is a block diagram of a set-top box in one embodiment of the invention.

FIG. 3 is a block diagram illustrating the components of a graphics tablet in one embodiment of the invention.

FIG. 4 is a flow diagram illustrating the flow of handwritten/textual information in one embodiment of the invention.

FIG. 5 is a flow diagram illustrating the manner in which the server's handwriting recognition application processes the image data to produce textual data in one embodiment of the invention.

While the invention is susceptible to various modifications and alternative forms, specific embodiments thereof

are shown by way of example in the drawings and will herein be described in detail. It should be understood, however, that the drawing and detailed description thereto are not intended to limit the invention to the particular form disclosed, but on the contrary, the intention is to cover all modifications, equivalents and alternatives falling within the spirit and scope of the present invention as defined by the appended claims.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

One embodiment of the inventive method is described in detail below. In this embodiment, an interactive television system transmits an audio-video-interactive signal from a broadcast station to a receiving station via a broadcast channel. ("Broadcast" is used herein to refer to transmission of a single signal to all subscribing receivers.) The broadcast channel may comprise a direct satellite transmission channel or any known means for broadcasting a signal, including non-satellite, cable, telco, MMDS (microwave) and terrestrial transmissions. (A "direct" satellite transmission as used herein means a transmission received by the interactive television receiver directly from the satellite.) The receiving station in this embodiment is additionally configured to receive signals via a modem connection to the broadcast station.

The audio-video-interactive signal broadcast to the receiving station may contain both television programming and interactive information such as control signals or interactive applications. When the broadcast signal is received, it is separated into its components and processed (e.g., decompressed) to reconstruct the respective television programming and interactive signals. An interactive application which was broadcast to the receiving station or which was resident in the receiving station is executed on the micro-processor of the set-top box. The application is one which requires textual data from the user. The user, however, is allowed to input the data in a non-textual form. "Non-textual" as used herein means a form handled by the system as something other than a series of alphanumeric characters, such as the ASCII character set. A graphical representation of handwriting, although textual in nature, is handled as an image or as vectorized handwriting information and not as the characters or words which are written. "Textual," on the other hand, means one or more characters or words. If the system is set up for an English-speaking user, the characters may be from the ASCII character set. If the system is set up for a non-English-speaking user, the characters will be from the character set of the user's language (e.g., Kanji characters for a Japanese user).

The set-top box of the receiving station is configured with a device for non-textual data entry. In one embodiment, this device is a graphics tablet on which the user can inscribe (write) the information required by the application. The graphics tablet digitizes the user's handwriting and stores the digitized data as an image file. The set-top box transmits the file containing the digitized data to the broadcast station for conversion into a textual form. (The image data need not all be contained in a single file, and may be transmitted to the server in segments to spread out the processing load.) The broadcast station includes a server which is configured to transform image data into textual data using handwriting recognition algorithms. The broadcast station then transmits the textual data back to the receiving station. The application executing on the set-top box identifies the received textual data as corresponding to the image data which it earlier transmitted to the broadcast station. The set-top box accord-

ingly utilizes the textual data as if it had been entered as text by the user of the set-top box and continues execution of the interactive television application.

Referring to FIG. 1, a block diagram illustrating a system for distribution of interactive television applications and television programs from their sources to a series of viewers is shown. Broadcast station **10** has a television program source **11** and an interactive application source **12**. The television program source may include remote broadcast network feeds, videotape recorders, computers, data storage devices, and the like. Interactive application sources **12** may provide interactive applications, control information or audio or video information which is to be included in the interactive television signal. Additionally, broadcast station **10** includes a server **13**, which is used to process non-textual data received at the broadcast station and generate textual data. The information generated by the television program source **11**, interactive application source **12** and server **13** is typically processed by compression/packetization units **14–16** before it is broadcast. Usually, the information is also compressed in order to conserve bandwidth. (It should be noted that while this embodiment comprises a digital broadcast system, other embodiments may comprise analog broadcast systems in conjunction with means for transmitting digital application data. The analog broadcast systems may include existing television networks, and the means for transmitting the application data may include transmissions in the vertical blanking interval or parallel transmissions via telephone lines or out-of-band cable signals.)

Any of a number of compression algorithms, such as one of the Motion Picture Expert Group (MPEG) compression standards, may be used if appropriate for a particular program or application. The information is packetized to enable error checking, interleaving of data and other transmission-related functions. Additional data may accordingly be appended to the application and programming data. For example, error check sums may be appended for error detection/correction and time stamps may be included for the purpose of synchronizing associated audio and video signals. The packetized information from compression/packetization units **14–16** is fed into multiplexing unit **17**, which intersperses the packets prior to transmission. The interspersed packets are then broadcast to the receiving stations **20**. (Although only one receiving station is shown in the figure, it is contemplated that the audio-video-interactive signal is broadcast to a group of subscribing receiving stations.) In the figure, the audio-video-interactive signal is depicted as being transmitted via satellite broadcast through antenna **19**.

The broadcast signal is relayed by communications satellite **28** and received by receiving station **20**. Although the figure illustrates a satellite transmission, it is contemplated that any broadcast medium (e.g., CATV or direct satellite transmission) may be used. Receiving station **20** is contemplated to be one of a number of such stations which are subscribers of the broadcast service provider operating broadcast station **10**. The broadcast signal is collected by receiving antenna **21** and fed to set-top box **22**. Set-top box **22** processes the packetized signal to reconstruct the television programs and interactive applications embodied in the signal. The reconstructed applications are executed in the set-top box, while the reconstructed television programs are passed to television **23**, where they are displayed. (It is noted that television **23** can be any suitable monitor or display device.) The interactive applications may generate graphics or audio which are combined with the television program prior to being displayed. The interactive applications may also be displayed in place of the television programming.

In addition to the broadcast channel between the broadcast station and receiving station, there may be other channels, such as a modem channel (which may also be referred to as an http channel, or hypertext transfer protocol channel.) These types of channels serve two functions in the system: they allow the set-top box to provide data to the broadcast station; and they provide an alternate path for from sources **11–13** to be delivered to receiving station **20**. It should also be noted that, if receiving station **20** and server **13** are connected via a transmission medium other than the broadcast station's broadcast channel and return path, the server is not constrained to be located at the broadcast station. It may be more convenient to locate server **13** at a site which is separate from broadcast station **10** in order to reduce the workload of the broadcast station, to more efficiently convey data between the receiving station and the server, or for other reasons.

Referring to FIG. 2, a block diagram of a set-top box **22** in one embodiment is shown. The broadcast signal is received and fed into tuner **31**. Tuner **31** selects the channel on which the broadcast audio-video-interactive signal is transmitted and passes the signal to processing unit **32**. (Tuner **31** may be replaced by other means, all collectively referred to herein as input ports, for receiving signals from various signal sources.) Processing unit **32** demultiplexes the packets from the broadcast signal if necessary and reconstructs the television programs and/or interactive applications embodied in the signal. The programs and applications are then decompressed by decompression unit **33**. The audio and video information associated with the television programs embodied in the signal is then conveyed to display unit **34**, which may perform further processing and conversion of the information into a suitable television format, such as NTSC or HDTV audio/video. Applications reconstructed from the broadcast signal are routed to random access memory (RAM) **37** and are executed by microprocessor **35**. Graphics tablet **39** provides a means for the user to supply handwritten information for conversion to text and subsequent use by the applications. (In other embodiments, graphics tablet **39** may be replaced by a microphone for supplying voice data or some other type of input device for supplying non-textual information to the system.)

Microprocessor **35** may comprise various types of microprocessors, microcontrollers, digital signal processors (DSPs), or other types of software instruction processing devices, as are appropriate to the particular design. RAM **37** may include memory units which are static (e.g., SRAM), dynamic (e.g., DRAM), volatile or non-volatile (e.g., Flash memory), as required to support the functions of the set-top box. When power is applied to the set-top box, microprocessor **35** executes operating system code which is stored in ROM **36**. (In some embodiments, ROM **36** may comprise Flash memory or EEPROMs.) The operating system code executes continuously while the set-top box is powered in the same manner the operating system code of a typical personal computer (PC) and enables the set-top box to act on control information and execute interactive and other applications. The set-top box also includes modem **38**. Modem **38** provides both a return path by which viewer data can be transmitted to the broadcast station and an alternate path by which the broadcast station can transmit data to the set-top box.

Although the term "set-top box" is used herein, it is understood that this term refers to any receiver or processing unit for receiving and processing a transmitted signal and conveying the processed signal to a television or other monitor. The set-top box may be in a housing which

physically sits on top of a television, it may be in some other location external to the television (e.g., on the side or back of the television or remotely located from the television), or it may be incorporated into the television itself. Alternatively, the functionality of the set-top box may be entirely removed from the television and placed in a more remote location, such as outside a house in which the set-top box is located. Set-top box **22** serves to demodulate (if necessary) the signal received from broadcast station **10** and to separate the components of the signal, such as different television programs and interactive applications. Other embodiments of the set-top box may have different components or interconnections than those shown in FIG. **2**. Similarly, television **23** may be a television or a video monitor employing any suitable television format (e.g., NTSC or HDTV), or it may be replaced by other devices, such as a video recorder, depending on the particular embodiment.

Referring to FIG. **3**, a block diagram illustrating the components of a graphics tablet **50** is shown. The main components of graphics tablet **50** are display **51**, digitizer **52** and processor **53**. Graphics tablet **50** can have a wide range of sizes, from several inches across (as used in pen computing systems) to several feet across (as used in some CAD systems). It is contemplated that the most convenient size for the graphics tablet will be the size of a small or medium sized notepad. The surface of graphics tablet **50** should be flat and smooth to facilitate use as a writing surface. Although graphics tablet **50** is contemplated to be connected to set-top box **22** by a data cable, infrared transmission or any other suitable transmission means may also be used to transmit data between graphics tablet **50** and set-top box **22**. To the extent that graphics tablet **50** is not constrained by these data transmission means, the user should be able to move the tablet to a convenient and comfortable writing position.

Display **51** and digitizer **52** are overlaid so that the path of a digitizer pen (not shown) can be displayed as the user writes on the tablet, thereby simulating writing on a notepad. Display **51** is contemplated to be a flat panel display using LCD (liquid crystal diode) or similar technologies. These technologies use substances between two plates of glass to control the amount of light which passes through the display. Depending upon the chosen technology, the display may simply control the amount of light from any source behind the display, or it may generate varying amounts of light from each pixel in the display. The display can be configured to provide feedback to the user by displaying his or her handwriting, or it may display icons or other images representative of the user's input. The display may also be configured to display forms, or other prompts, in response to which the user can provide handwritten information. In an alternate embodiment, graphics tablet **50** may be simplified by eliminating display **51**. In such an embodiment, the user could instead receive visual feedback, including prompts, forms, handwriting and other images through television **23**.

The combination of display **51** and digitizer **52** should be selected to provide sufficient resolution to allow an accurate digital representation of the user's handwriting. It is contemplated that the resolution of the digitizer and display should be roughly equivalent to that of a fairly high-quality printer. The combination should also provide display quality (e.g., brightness and contrast) which makes the handwritten information easy for the user to see. Microprocessor **53** should be fast enough that the path of the digitizer pen is displayed without any substantial delay as it is drawn by the user. Although microprocessor **53** is shown separately from

microprocessor **35**, it is contemplated that a single device may be sufficient to perform both functions.

Digitizer **52** may also be constructed using one of a number of technologies. Although early technologies often used opaque structures, technologies used in modem digitizers are more likely to incorporate transparent panels which are designed to be used in conjunction with overlaid displays. Digitizer **52** may use electrostatic, resistive film or capacitive/electrostatic film technologies. Various embodiments of the invention may incorporate different ones of the technologies which are best suited to the particular embodiments. Digitizer **52** senses the position of the digitizer pen and transmits the corresponding X-Y coordinates to the set-top box. In one embodiment, this X-Y data is recorded as a series of darkened pixels in a bitmap image of the text written by the user. The image is then transmitted to the server for processing and recognition of the written data. The recognition of the bitmap image is sometimes referred to as off-line recognition.

In an alternate embodiment, digitizer **52** may sense not only X-Y position, but also the direction of the pen's path, the pen's angle, speed and acceleration, and other information which forms a vectorized representation of the user's handwriting. This vectorized information can be transmitted to the server as it is produced so that recognition can be performed as the user is writing. The use of vectorized information to recognize handwritten information is sometimes referred to as on-line recognition.

The receiving station is operatively connected to the broadcast station by a broadcast channel. This broadcast channel can utilize various transmission media and is contemplated to include media such as coaxial cable and free space (e.g., as used for direct satellite transmissions.) The broadcast channel forms a transmission path between the broadcast station and the receiving station. The broadcast station and receiving station are also connected by a return path. The return path typically consists of a pair of modems, one in the receiving station and one in the broadcast station, each connected to a standard telephone line. Other means for establishing a return path (e.g., using a portion of the bandwidth of the broadcast channel) are also contemplated. The system illustrated in FIG. **1** utilizes the broadcast channel to establish a return path for communicating data from the receiving station to the server.

In one embodiment, an application is transmitted from the broadcast station to the receiving station, where it is reconstructed and executed. In alternate embodiments, the application may be resident in the set-top box or may be provided to the set-top box by means other than the broadcast channel (e.g., flash card.) FIG. **4** is a flow diagram illustrating the flow of handwritten/textual information in the system. While the application is executing **61**, it prompts the user for some sort of textual information **62**. The user enters the information on the graphics tablet **63**, writing the information in his or her normal handwriting. The graphics tablet digitizes the image of the user's writing and conveys the digitized data to the set-top box, which transmits the information to the server **64** via the return path to the broadcast station. The image data is received by the broadcast station and passed on to the server. The server processes the image data **65**, recognizing characters and/or words in the image and producing the equivalent textual data. This textual data is then transmitted back to the receiving station **66**, either via the broadcast channel or via the return channel. The textual data is then utilized by the application executing in the set-top box **67** as if the text had been directly entered by the user.

FIG. **5** is a flow diagram illustrating the manner in which the server's handwriting recognition application processes

the image data to produce textual data. An off-line, bitmap recognition system is used in this embodiment. It is assumed in this instance that the user's handwriting is not contained in a form, but is instead free-form input to the graphics tablet. Further, in one embodiment, the handwriting image is transmitted to the server in segments as they are generated by the graphics tablet. This may serve both to reduce the peak loading of the server resulting from recognition of the image and to allow the server to provide simultaneous feedback (i.e., recognized characters) to the user. ("Simultaneous" as used here means that some recognized text is transmitted to the set-top box as the user continues to write on the graphics tablet so that the user can see the results of the handwriting recognition.) If the image instead combines text with a known form, the form is first identified and removed. Then, if the handwriting was confined to several boxes or fields on the form, these fields are isolated and handled individually. (Free-form handwritten entries may be distinguished from each other in the same manner as form entries by providing different dialog boxes in which the user can write the entries.)

The handwriting in a field is recognized by first breaking the image into segments showing characters or pieces of characters 71. The individual image segments and combinations of these image segments are selected as character candidates 72. These character candidates are assigned character classes and associated values representative of the confidence with which the recognition application places the segment in the associated character class 73. The character candidates are organized into groups for which a dictionary look-up algorithm can be performed 74. A dictionary look-up determines the word entries which best match the groups of character candidates 75 and may assign a level of confidence to each word. If desired, an acceptability check can be performed to determine whether the confidence level of the recognized words is sufficiently high 76. If the confidence level is too low, the word can be rejected and the image will be considered unrecognizable. The user can also be given the option of checking the textual output of the recognition application and accepting or rejecting all or part of the output 77. After the user has verified the accuracy of the recognized text, the data can be provided to the application executing in the set-top box. It should be noted that the description is intended to be illustrative rather than restrictive and that the process of recognizing vectorized handwriting data or voice data will differ from the foregoing description.

The recognition of the user's handwriting may be assisted by the association of contextual information with the handwriting. If the user's handwriting comprises entries on a form, identifying the type of information requested for each entry may make it easier to interpret the entry itself. For example, an entry in a box requesting a social security number should contain nine digits. A character in this entry which might be interpreted as a "1", an "I" or and "1" must be the numeral "1". The identification of keywords such as "to" or "cc:" may likewise distinguish the handwriting which follows as a name or address.

The system described above can be used with a number of different applications. For example, an interactive television service provider may wish to provide e-mail service to subscribers. The user can select the e-mail application furnished by the service provider and proceed to write the message which he or she wishes to send on the graphics tablet. In the message, the user writes the address of the intended recipient and the message to be sent to the recipient. The graphical data is transmitted to the server, which

may segment the image data and then convert the data to text, or it may convert the entire image to text and then parse the text to determine the recipient's address. In an alternate embodiment, only the address of the message is converted to text while the body of the message is transmitted to the recipient as an image. Although the address of the e-mail must be computer-readable so it can be properly routed to the recipient, the body of the message need not be converted to text—the recipient can read the message embodied in the image whether or not it is converted to text data. In fact, it may in some cases be preferable to deliver some messages as image data so that the sender can communicate drawings or other non-textual data. Delivery of the image of a handwritten note may also be preferable because simple text messages may be considered impersonal.

The user may be given the option of selecting an addressee from an address book or a list of previous addressees. A menu can be presented to the user via the graphics tablet to allow selection of one of these addressees. New addressees which are handwritten can be added to the user's address book. The user can also select other options (e.g., sending an image versus sending text only) in the same manner as selecting an addressee. The information which the user enters via the graphics tablet may therefore be a mix of handwriting and selection of particular predefined inputs. A system which uses a microphone for non-textual input may also be configured to allow selection of predefined inputs by using audio prompts and corresponding menus.

In another embodiment, the system may be configured to send faxes instead of e-mail. The foregoing description of the e-mail-configured system is in large part applicable to the fax-configured system. The user can enter a destination fax number and/or addressee via the graphics tablet, and this information can then be transmitted to the server for conversion into textual information. The textual information is then transmitted back to the set-top box and used by the fax application to dial the destination fax. As for the e-mail-configured system, the transmitted fax may be an image of the user's handwriting, or it may be an image of textual information corresponding to the user's handwriting. (It should be noted that in this instance, the system may convert the user's handwriting to textual information so that the information is more legible than handwriting or so that more information will fit on a page, but the transmitted fax will by its nature consist of image data rather than textual data.)

Another example of an application with which the system can be employed is electronic commerce service. Electronic commerce services are those through which the user can buy items or otherwise conduct business. These services include online catalogs and home shopping services. When using an online catalog, the user can browse through the catalog to determine which products he or she would like to order. The user can provide item information via menu entries, but must enter non-standardized information such as a shipping address via the graphics tablet.

In another embodiment, the user can enter his information by voice. The user can use a microphone or a telephone handset to provide voice data to the system. The microphone may be a special-purpose microphone for use with the interactive television system or it may be a telephone handset. A special-purpose microphone may be connected to the set-top box, or it may be built into a remote control for the system. A telephone handset may be connected to the set-top box, or it may be connected directly to the return path (i.e., telephone line.) The voice data is transmitted to the server, which uses voice recognition software to convert the voice data into textual data. The textual data is returned to the

set-top box, where it can be displayed to the user. The user can correct the text or confirm that the text has been accurately generated from the voice data.

While the present invention has been described with reference to particular embodiments, it will be understood that the embodiments described above are illustrative and that the scope of the invention is not limited to these embodiments. Many variations, modifications, additions and improvements to the described embodiments are possible. These variations, modifications, additions and improvements are intended to be within the scope of the invention as detailed within the following claims.

What is claimed is:

1. An interactive television system comprising:
 - a receiving station (20) configured to receive a broadcast signal, wherein said receiving station is configured to receive executable interactive application code corresponding to an interactive application via said broadcast signal, wherein said receiving station (20) is configured to execute said interactive application, wherein said interactive application includes a function which requires input in a textual format, and wherein said receiving station includes an input device configured to receive non-textual information from a user; and
 - a remote server coupled to said receiving station by a transmission medium, wherein said server is configured to receive said non-textual information from said receiving station and to convert said non-textual information into textual information, wherein said server is further configured to provide said textual information to said interactive application.
2. The interactive television system of claim 1 further comprising a broadcast station coupled to said receiving station by a broadcast channel.
3. The interactive television system of claim 2 wherein said server is coupled to said broadcast station and wherein said transmission medium comprises a return path between said broadcast station and said receiving station.
4. The interactive television system of claim 3 wherein said receiving station includes a first modem, wherein said broadcast station includes a second modem and wherein said return path comprises a telephone line coupled to said first and second modems.
5. The interactive television system of claim 3 wherein said return path comprises a portion of the bandwidth of said broadcast channel between said broadcast station and said receiving station.
6. The interactive television system of claim 1 wherein said input device comprises a graphics tablet.
7. The interactive television system of claim 6 wherein said graphics tablet comprises a digitizer configured to generate data corresponding to handwriting inscribed thereon.
8. The interactive television system of claim 7 wherein said graphics tablet further comprises a display overlaid with said digitizer and configured to display an image of said handwriting as said handwriting is inscribed on said digitizer.
9. The interactive television system of claim 1 wherein said receiving station further comprises a display and wherein said receiving station is configured to present said textual information on said display for verification of said textual information by said user.
10. The interactive television system of claim 1 wherein said input device comprises a microphone.
11. A set-top box for use in an interactive television system, said system having a remote server configured to

convert non-textual information to corresponding textual information, the set-top box comprising:

- receiving means (31) configured to receive executable interactive application code corresponding to an interactive application via a broadcast signal;
 - a microprocessor configured to execute said interactive application, wherein said interactive application includes a function which requires textual information from a user;
 - an input device for receiving non-textual information from said user; and
 - transmitting means coupled to said input device for transmitting said non-textual information to said remote server; and
- wherein said receiving means is configured to receive said corresponding textual information from said server, and convey said corresponding textual information to said microprocessor for use by said interactive application.
12. The set-top box of claim 11 wherein said input device comprises a graphics tablet configured to receive handwriting inscribed thereon by said user and wherein said non-textual information comprises digitized information corresponding to said handwriting.
 13. The set-top box of claim 11 wherein said input device comprises a microphone configured to receive voice information.
 14. The set-top box of claim 11 wherein said transmitting means comprises a modem.
 15. The set-top box of claim 11 wherein said receiving means comprises a broadcast receiver.
 16. The set-top box of claim 11 wherein said microprocessor is configured to prompt said user for said non-textual information, to receive said non-textual information and to provide said non-textual information to said transmitting means.
 17. A method in an interactive television system for providing textual input to an interactive application executing in a set-top box of said system, the method comprising:
 - receiving a broadcast signal, wherein said broadcast signal comprises executable interactive application code corresponding to an interactive application, wherein said interactive application includes a function which requires textual input;
 - executing said interactive application;
 - providing information in a non-textual form to said interactive application in said set-top box;
 - conveying said information in said non-textual form from said set-top box to a server;
 - converting said information from said non-textual form to a textual form in said server;
 - providing said information in said textual form to said interactive application.
 18. The method of claim 17:
 - wherein said non-textual form of said information comprises images of characters; and
 - wherein providing said information in said non-textual form comprises drawing said images on a graphics tablet coupled to said interactive television system.
 19. The method of claim 17:
 - wherein said non-textual form of said information comprises digitized image data; and

13

wherein providing said information in said non-textual form comprises drawing images of characters on a graphics tablet coupled to said interactive television system and digitizing said images and produce said digitized image data.

20. The method of claim **17**:

wherein said non-textual form of said information comprises images of characters; and

wherein converting said information from said non-textual form to said textual form comprises executing a handwriting recognition application on said server and providing said images to said application to produce said textual form of said information.

14

21. The method of claim **17**:

wherein said non-textual form of said information comprises voice data; and

wherein providing said information in said non-textual form comprises speaking into a microphone coupled to said interactive television system and said microphone producing said voice data from spoken words.

22. The method of claim **17** wherein converting said information from said non-textual form to a textual form in said server comprises converting said non-textual form of said information to ASCII characters.

* * * * *

5
10



US005841431A

United States Patent [19]
Simmers

[11] **Patent Number:** **5,841,431**
[45] **Date of Patent:** **Nov. 24, 1998**

- [54] **APPLICATION OF SPLIT- AND DUAL-SCREEN LCD PANEL DESIGN IN CELLULAR PHONES**
- [75] Inventor: **Charles Russell Simmers**, Phoenix, Ariz.
- [73] Assignee: **Intel Corporation**, Santa Clara, Calif.
- [21] Appl. No.: **749,486**
- [22] Filed: **Nov. 15, 1996**
- [51] **Int. Cl.**⁶ **G09G 5/00; G09G 3/36; G06F 1/00; G06F 1/16**
- [52] **U.S. Cl.** **345/211; 345/103; 364/705.05; 364/707; 364/708.1**
- [58] **Field of Search** 345/211, 103, 345/104, 98, 100, 131, 156, 173; D14/137, 138; 364/705.05, 707, 708.1; 379/433, 428, 440; 395/750, 750.04; 455/556, 566, 574, 575, 89, 90

D. 374,227	10/1996	Williams	D14/138
D. 377,341	1/1997	Imai et al.	D14/138
4,816,816	3/1989	Usui .	
5,189,632	2/1993	Paajanen et al.	364/705
5,392,058	2/1995	Tagawa	345/104
5,410,329	4/1995	Tagawa et al.	345/104
5,534,892	7/1996	Tagawa	345/104
5,548,765	8/1996	Tsunoda et al.	395/750
5,663,745	9/1997	Ishikawa et al.	345/98

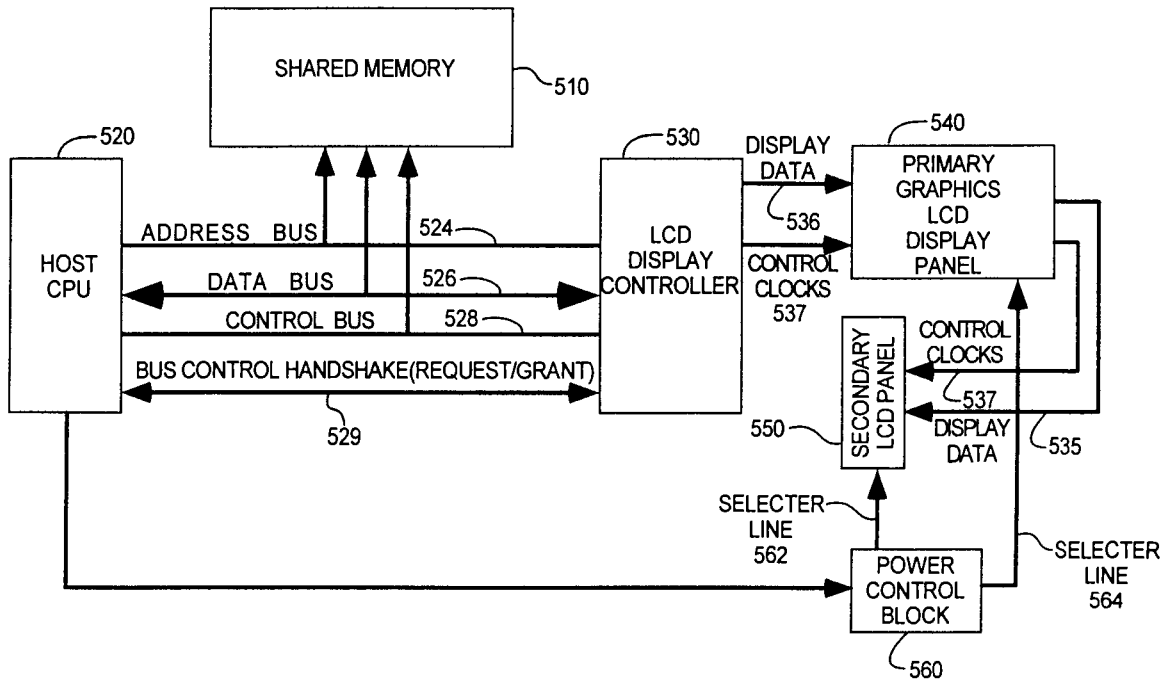
Primary Examiner—Steven J. Saras
Assistant Examiner—David L. Lewis
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

An apparatus for conserving power in information devices with dual functions. A single display panel is logically split into two sub-panels. Each sub-panel can be powered up or down separately as is required by the function of the device. The display panel has a plurality of improved segment drivers which are provided power signals enabling the set of segment drivers corresponding to a sub-panel to be separately powered. In systems with two separate display panels, each of the panels may be powered up or down by the use of similar improved segment drivers as necessary.

- [56] **References Cited**
U.S. PATENT DOCUMENTS
D. 370,673 6/1996 Happo et al. D14/138

11 Claims, 6 Drawing Sheets



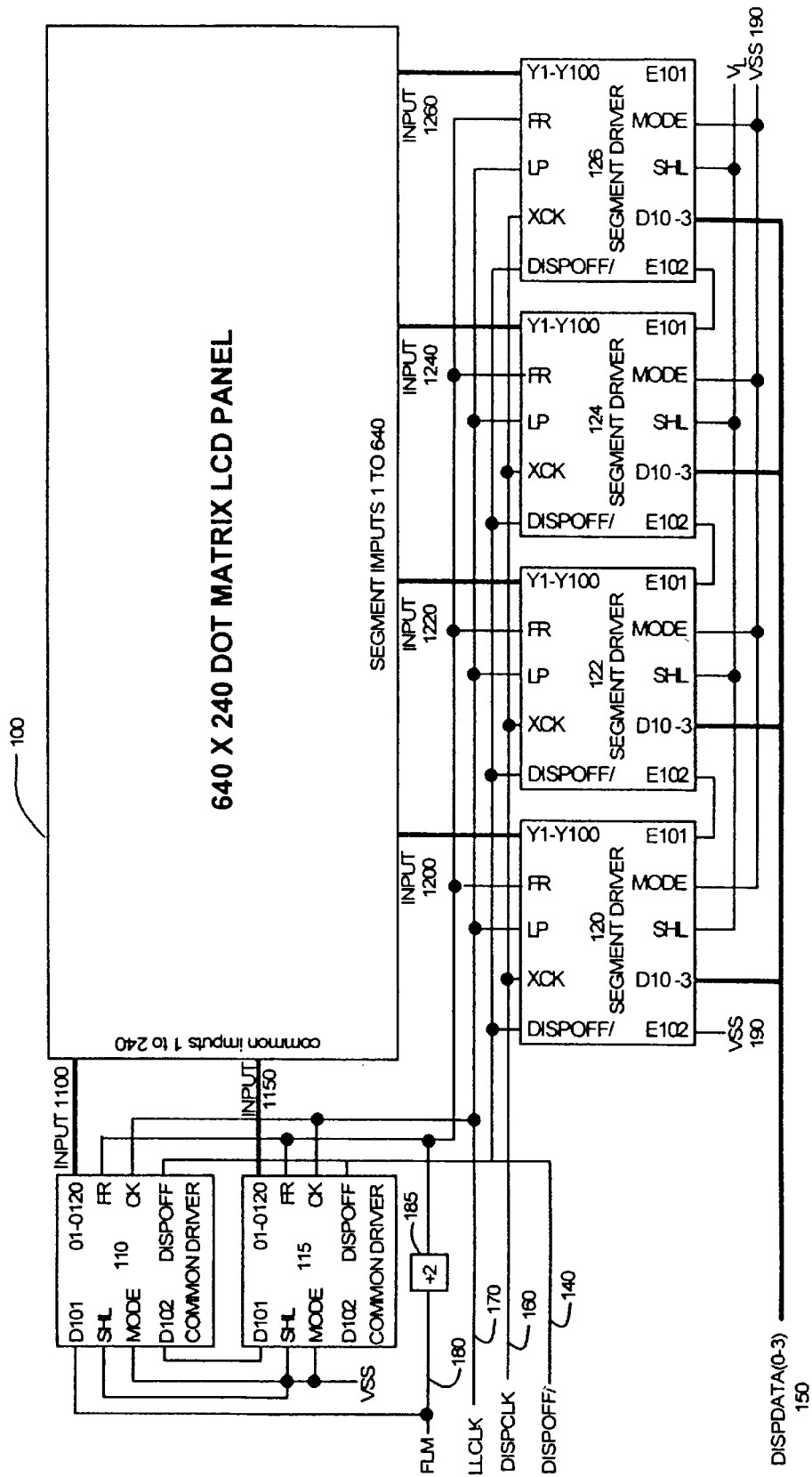


FIG. 1 (PRIOR ART)

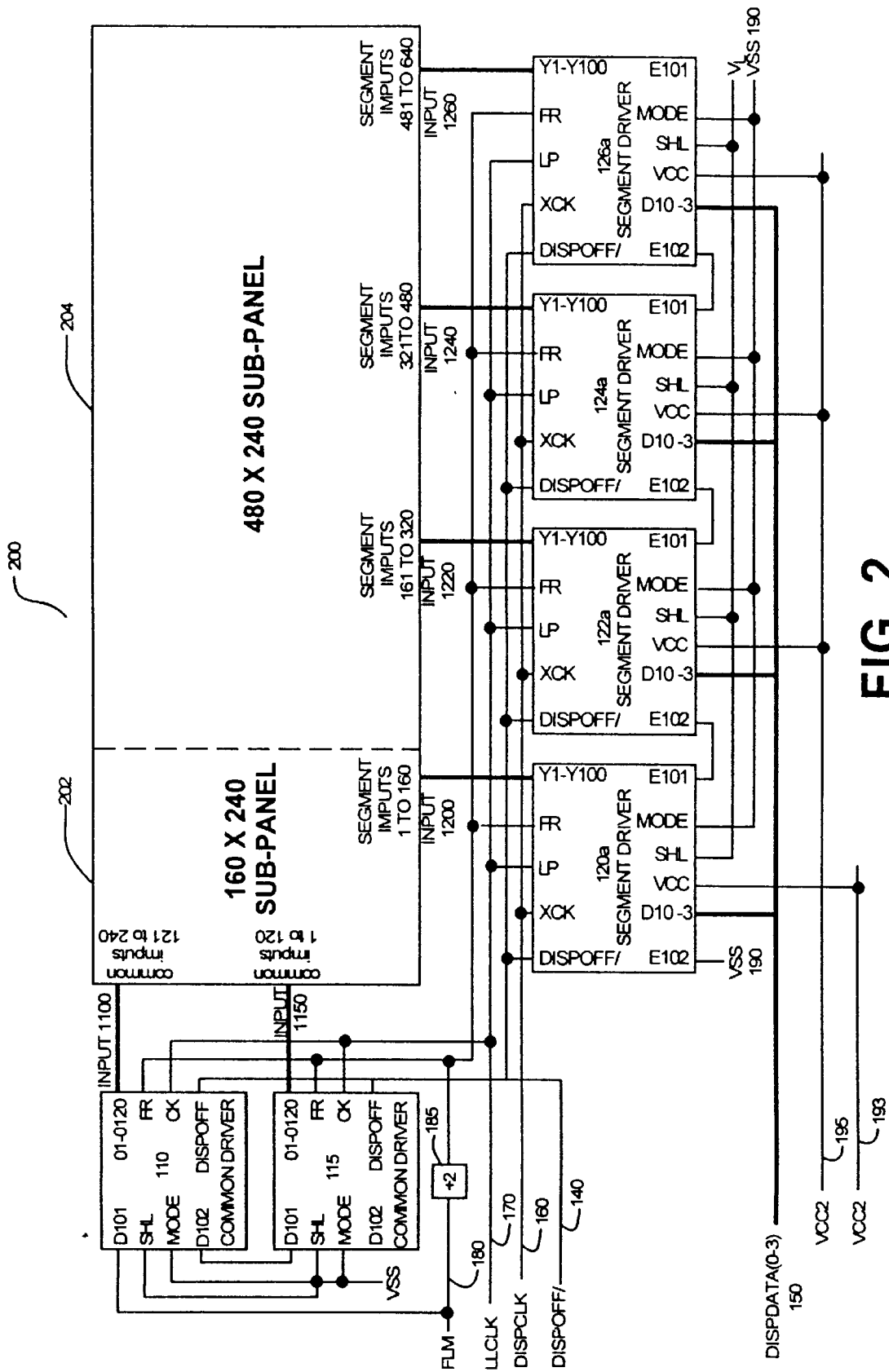


FIG. 2

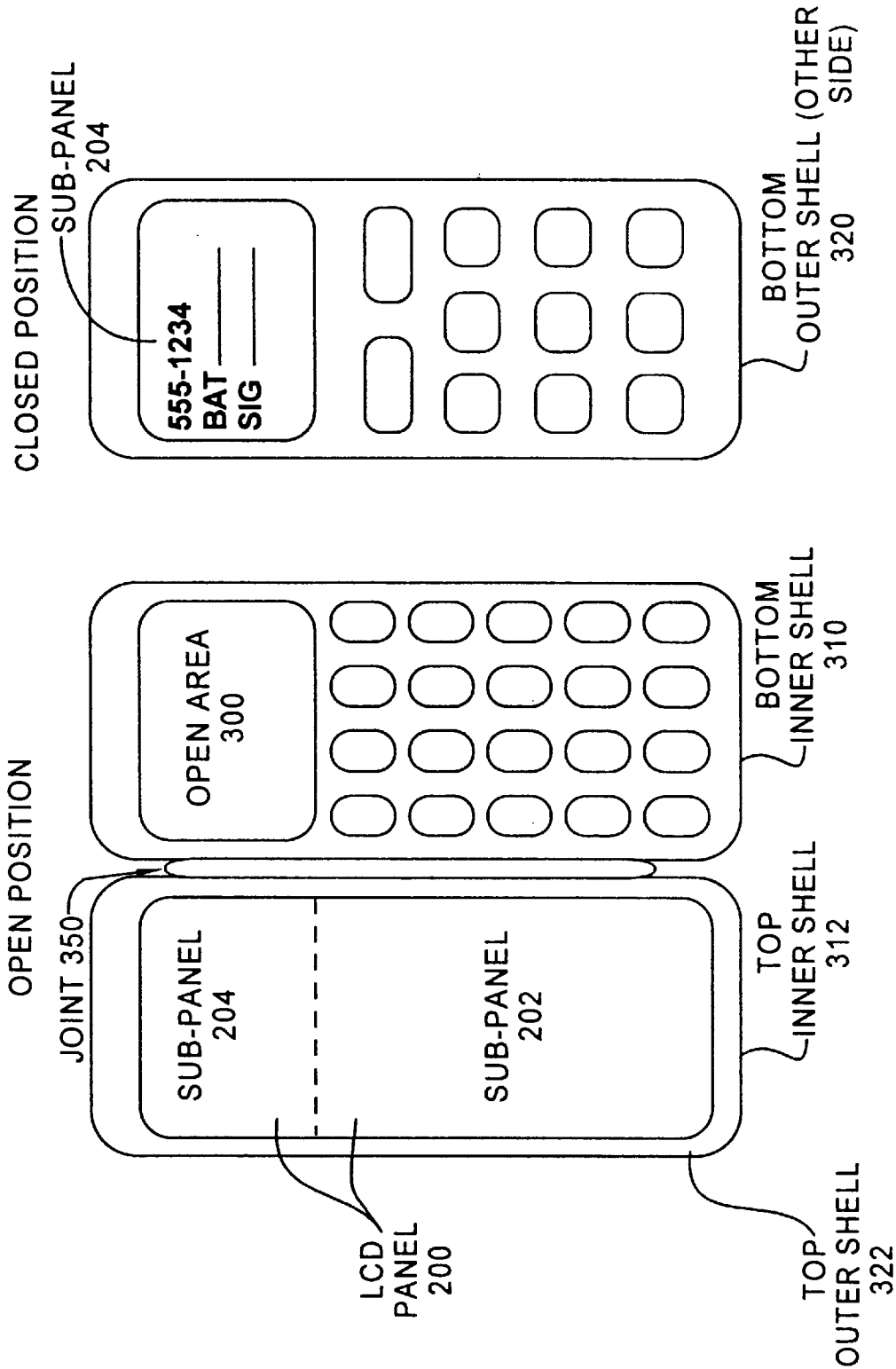


FIG. 3

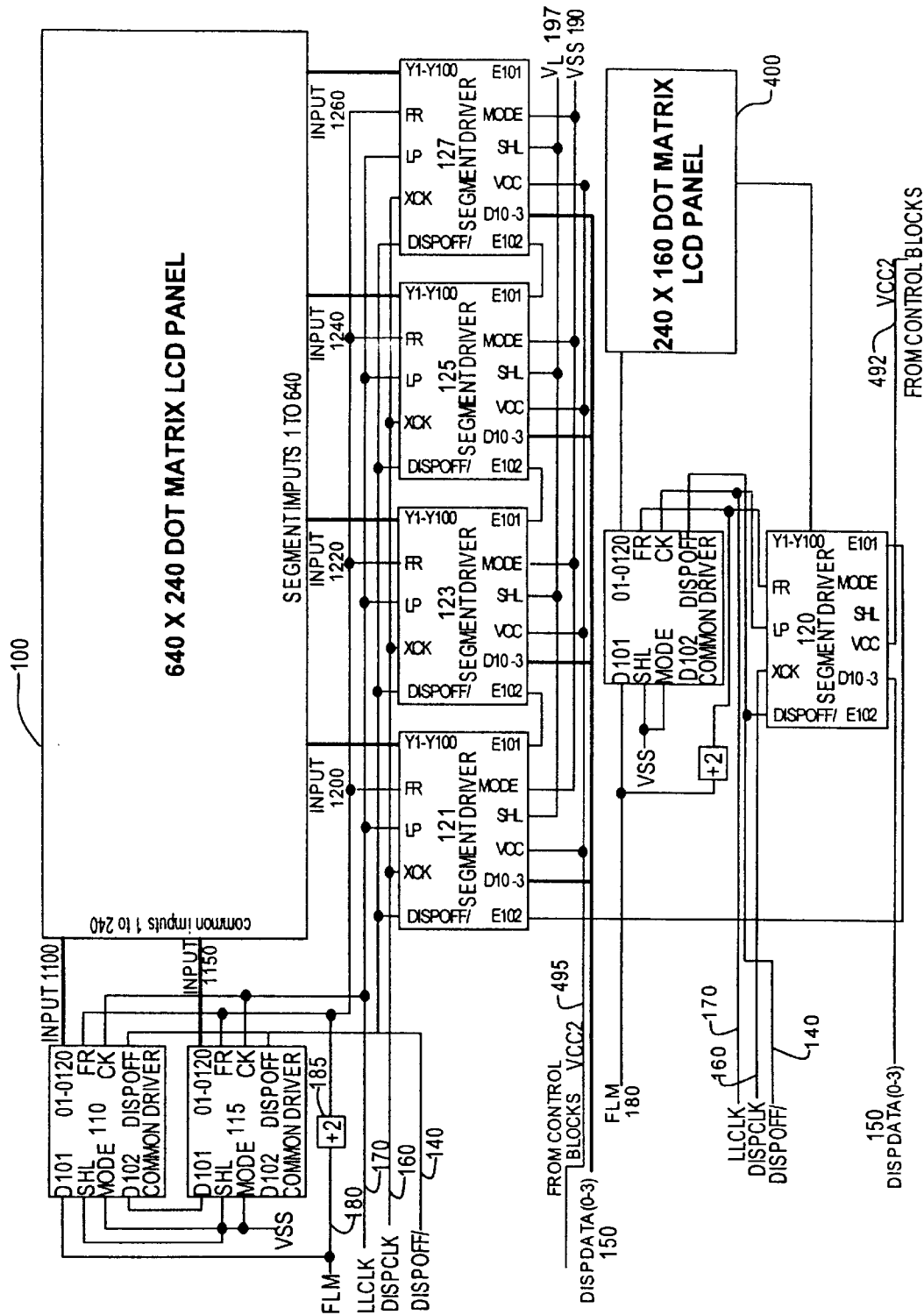


FIG. 4

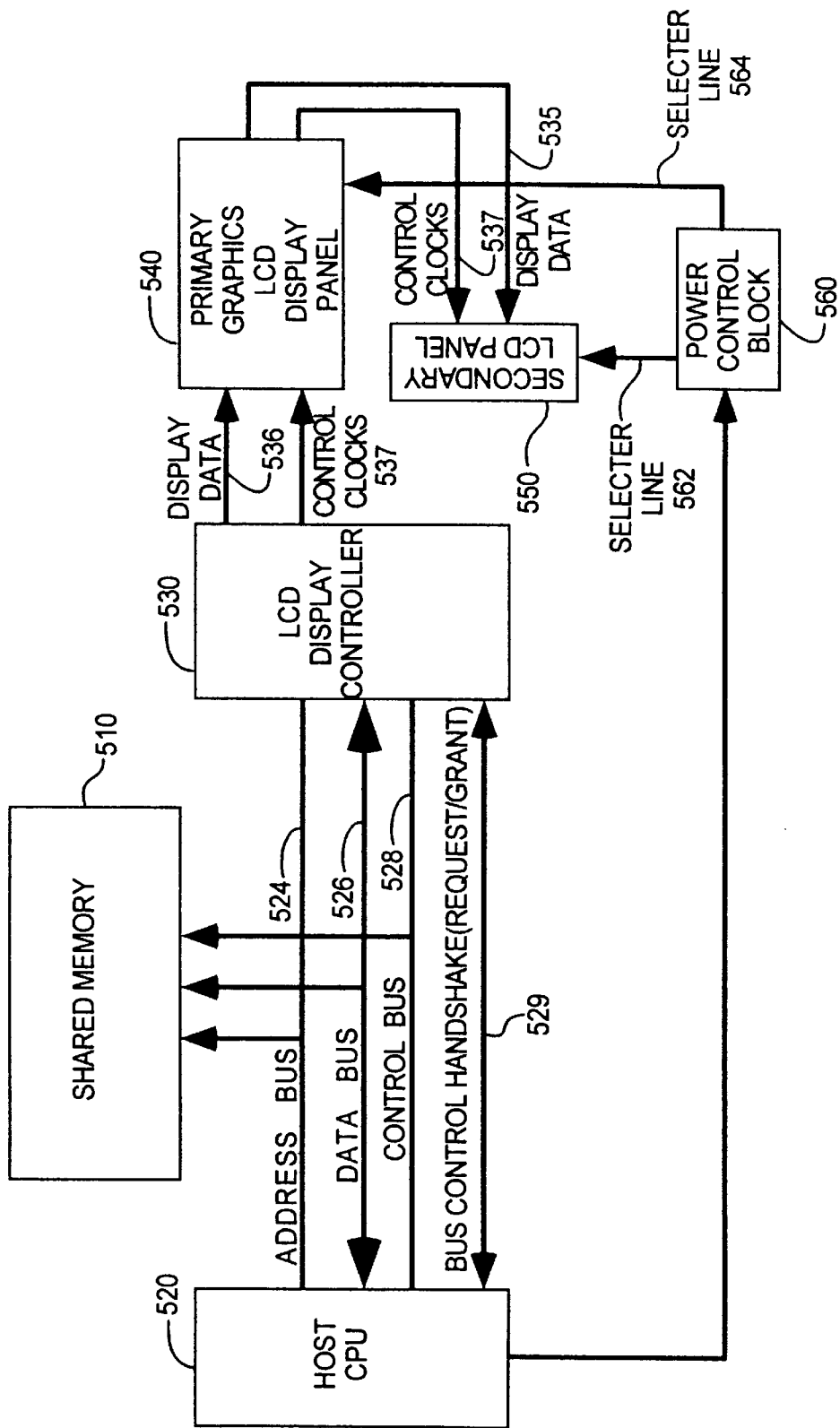


FIG. 5

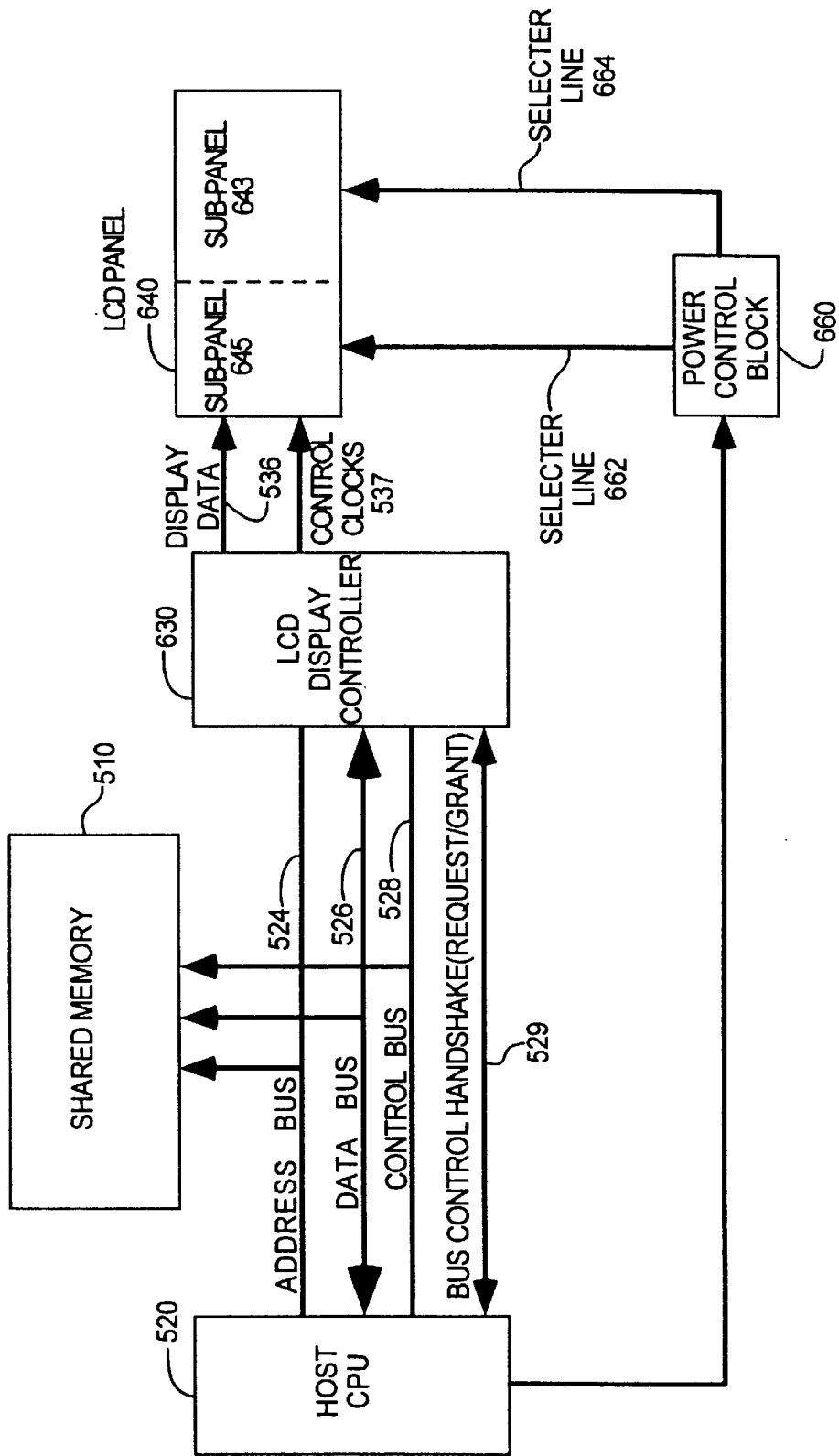


FIG. 6

APPLICATION OF SPLIT- AND DUAL- SCREEN LCD PANEL DESIGN IN CELLULAR PHONES

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to the field of display devices. More specifically, the present invention relates to graphical displays connected to information devices.

2. Description of Related Art

In high-end "smart" cellular phones, which function both for telecommunications and for storing and retrieving information (e.g., a Personal Digital Assistant (information device)), it is often necessary to provide two displays, one for each function. The smaller of the displays, used for the telecommunications function, commonly consists of between ten and twenty characters across (columns) and three to eight rows. The larger of the displays, used for the information device function, is a graphical display with a resolution of typically 640 pixel columns across by 240 pixel rows.

Traditionally, each display was treated as a separate system since the smaller display operates continuously, while the larger display operates more sparingly. In periods of non-use, the large display is powered-down. Disadvantageously, each display has its own controller to convert information into displayable pixels and its own integrated circuits which drive the pixels to be output on the display panels. In battery-operated and power-conscious devices such as PDAs, the redundancy of having two sets of drivers, integrated circuits and controllers is expensive and can also increase the mean-time-between-failure for the devices. Further, where a single display is used for both functions, the entire display must be active, even when only a small sub-panel of the display is required to operate (i.e., for telecommunications). In such a circumstance, the power drain is excessive for the function served, and, therefore, highly inefficient.

Thus, there is a need to reduce the power drain of such devices by allowing independent operation of only one display, in the case of two separate displays, and a sub-panel in the case of single physical display.

SUMMARY

In the case of some dual-function information devices such as a cellular phone with PDA, two separate physical displays are controlled by a single video controller. The video controller provides a plurality of control signals to drivers which drive pixels onto the displays. The invention provides a power control block which is coupled to those drivers to selectively power-down drivers for the larger of the two displays, while keeping powered-up the smaller of the displays. The power control block can be programmed by a user/software to power-up or power down the displays as dictated by the use of the Information device. The power control block is, therefore, coupled to a CPU or other such processor from which it receives commands regarding which display to keep powered-up and which to power down.

Alternatively, in dual-function information devices where there is only one physical display for the information device, a similar power control block can be programmed by instructions being entered by the CPU to selectively power-down certain pixel drivers for the display and thereby create a logical "sub-panel". A single display screen may be split

into two or more logical sub-panels, each of which has corresponding drivers which output pixels to their portion of the display, and are independently powered-up or down as the application requires.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is an operational diagram of a typical LCD display according to the prior art.

FIG. 2 is an operational diagram of a split screen LCD display according to one embodiment of the invention.

FIG. 3 is an illustration of how a split screen LCD display may be utilized in a information device.

FIG. 4 is an operational diagram of a dual screen LCD display according to one embodiment of the invention.

FIG. 5 is a system diagram of a computer system with a dual LCD panel display system according to one embodiment of the invention.

FIG. 6 is a system diagram of a computer system with a split-screen LCD panel according to one embodiment of the invention.

DETAILED DESCRIPTION OF THE DRAWINGS

FIG. 1 is an operational diagram of a typical LCD display according to the prior art.

FIG. 1 shows a 640 by 240 dot-matrix liquid crystal display (LCD) panel **100** which is driven by two inputs **1100** and **1150** for rows running down the vertical axis of panel **100** and also by four inputs—**1200**, **1220**, **1240** and **1260**—driving pixels in columns across the horizontal axis of panel **100**. Thus, on a 640 by 240 pixel display such as LCD panel **100**, input **1100** is responsible for the first 120 rows of pixels and input **1150** is responsible for the last 120 rows of pixels.

For LCD panel **100**, input **1200** drives the first 160 columns of pixels, input **1220** the second 160 columns of pixels, input **1240** the third 160 columns of pixels and input **1260** the last 160 columns of pixels of LCD panel **100**. LCD panel **100** may be used on a notebook computer, a personal digital assistant (PDA), cellular phone or for use in any information device capable of utilizing an LCD output.

The output of such LCD panels are typically driven by "segment" drivers driving the pixel columns on the horizontal axis and by "common" drivers which enable pixel rows on the vertical axis of the panel. The physics of driving pixel output on display panels is well-known in the art and will not be described in depth. Common driver **110** and common driver **115** generate input signal **1100** and input signal **1150**, respectively, while segment driver **120**, segment driver **122**, segment driver **124** and segment driver **126** generate input signals **1200**, **1220**, **1240** and **1260**, respectively. Each of these segment drivers convert serial data into parallel data and generate for output level translator signals which map an incoming digital signal into certain voltage levels which the LCD panel converts into pixel intensities based on the voltage level differentials. The common drivers activate a particular row for displaying output generated by the segment drivers.

Also shown in FIG. 1 are several control signals originating from the display controller of the cellular phone, information device or computer system that utilize the display capability of LCD panel **100**. Shown are a display off control (DISPOFF) signal **140**, a display data (DISPDATA) signal **150**, a display clock (DISPCLK) signal **160**, a line latch clock (LLCLK) signal **170**, and a first line marker (FLM) signal **180**. Also input to the drivers are two voltage signals, VL **197** and VSS **190**. These voltage signals,

VL 197 and VSS 190, can be used to indicate different logic levels to the pins they supply.

DISPDATA 150 is a signal comprised of four or eight bits—0, 1, 2 and 3 or 0 through 7—which are transmitted in parallel and represent light/color intensity levels to be output on LCD panel 100 and originates from a display controller device. DISPDATA deviate 150 is output on LCD panel 100 with bit 0 in the upper left corner of the screen and bits 1, 2 and 3 output on the same row from left to right starting after bit 0. The serial to parallel conversion of DISPDATA 150 is carried out by the timing signal DISPCLK 160 which originates from a clocking mechanism. DISPCLK 160 clocks the 4 bits of DISPDATA 150 into shift registers contained in the segment drivers. Once the shift registers in segment driver 120 are full, then another or similar clocking mechanism asserts the line latch clock (LLCLK) 170 signal to common driver 110. As shown in FIG. 1, the line latch clock is also connected to a latch pulse (LP) pin or input on segment drivers 120, 122, 124 and 126, such that when the shift registers are filled with bits of display data and the LLCLK signal 170 has been asserted, the bits stored in the shift registers are latched and transferred over input line 1200 to LCD panel 100. The LLCLK signal 170 which essentially loads an entire row of pixels to LCD panel 100, also clocks the common driver incrementing the shift register of the common driver 110 by one such that the LCD panel can enable the next row of the panel for pixels driven by the segment drivers once a row has been completed. DISPDATA 150 transmits a four-bit signal (in parallel), corresponding to four pixels for the LCD panel, to the segment drivers.

Once all of the rows of pixels have been output in this manner, such that the display of pixels is completed for one image frame, a first line marker (FLM) signal 180 is again asserted, which is also clocked with the line latch clock LLCLK 170. First line marker signal 180 propagates through all of the shift registers of all common and segment drivers resetting the shift registers to zero, such that the common driver 110 is set to enable the next new row of pixel data to be output by the segment drivers. Likewise, segment drivers 120, 122, 124, 126 are also reset to receive the next set of pixel data from DISPDATA 150.

The DISPOFF signal 140 shown in FIG. 1, when driven active, disables the output for all pins and thereby blanks LCD panel 100 such that no pixels are output to the panel. FLM 180 is also divided by two by a divider circuit 185 to periodically reverse the polarity of the pins where inputs 1200, 1220, 1240, 1260 and inputs 1100 and 1150 are output by the segment and common drivers. Periodically, reversing polarity is necessary because the typical LCD requires an alternating current (AC) signal such that the liquid crystal does not “plate-out” against the electrodes and turn black. Thus, the FR pin is periodically reversed and sets the internal shift registers at one, rather than zero. Other pins shown in the segment drivers 120, 122, 124 and 126 are an XCK pin, which receives the DISPCLK signal 160, the DISPOFF pin, which receives the DISPOFF signal 140, and an output pin labeled Y1–Y160, which transmits the pixels which are stored in the shift registers of the segment drivers to the LCD panel 100. Also shown in FIG. 1 are external input/output expansion pins EIO-1 and EIO-2 for each of the segment drivers 120, 122, 124 and 126.

The expansion pins EIO-1 and EIO-2 are connected together such that the EIO-1 pin of segment driver 120 loads a ground or loads a negative voltage value from EIO-1 of segment driver 120 to EIO-2 of segment driver 122, indicating that the first 160 pixels have been output by segment

driver 120 and that the next 160 pixels of the row may be output by segment driver 122. This daisy-chaining is provided also for the segment drivers 122 to 124 and 124 to 126 by propagating either ground/negative value to these segment drivers to complete the pixel row. Likewise, on common drivers 110 and 115 are pins DIO-1 and DIO-2, which are daisy-chained together such that when the first 120 rows of pixels enabled by common driver 110 are completed, common driver 115 receives the remainder of the data and completes pixel rows 121 through 240. The FR pin, or frame pulse pin, of common drivers 110 and 115 operate similarly to the FR pins of segment drivers 120, 122, 124 and 126 and will not be described further. Likewise, the DISPOFF pins of the common drivers 110 and 115 operate similarly to the DISPOFF pins of segment drivers 120, 122, 124 and 126 and will not be described further. Common driver 110 has a CK pin which is driven from LLCLK signal 170 and, in a given time index, represents the number of rows which have been output to the LCD panel 100. The SHL pin of the segment drivers 120, 122, 124 and 126, as well as the SHL pin of common drivers 110 and 115 serve to indicate in which direction pixels representing the image are output to the display, whether left to right, right to left or, in the case of the common drivers, top to bottom, or bottom to top.

Further, a mode pin is provided on all of the segment drivers and the common drivers which, when input a certain logic level from VSS 190, indicates a mode in which the drivers operate. VL 197 is shown as an input level to the SHL pins of the segment drivers and by its logic level indicates what direction the image is being output in. The physics underlying the liquid crystal display is well known in the art and will not be described so as not to obscure the invention. According to the prior art, therefore, the entire bank of segment drivers 120, 122, 124 and 126 is always powered-up and enabled for output. There is no signal or mechanism to power separately, any of the segment drivers. Thus, when only a portion of the panel has displayable output such as when the information device functions as a telecommunications device, the power consumed by the rest of the panel and their segment drivers is wasted.

FIG. 2 illustrates an operational diagram of a split screen LCD panel according to one embodiment of the invention.

All pins of the segment and common drivers, input and control signals which toggle them as described with respect to FIG. 1 with identical reference numbers operate similarly with regard to this embodiment of the invention and will not be repeated. However, the invention provides for additional control by way of VCC pins on each of the segment drivers as well as splitting VCC into separate signals VCC1 195 and VCC2 193. Thus, a typical segment driver circuit would need to be modified as follows to provide for split panel LCD operation.

According to the embodiment shown in FIG. 2, a single 640 by 240 resolution dot matrix LCD panel 200, which is similar to the 640 by 240 resolution LCD panel 100 of FIG. 1, is split logically into a 160 by 240 size sub-panel 210 and a 480 by 240 size sub-panel 204. By logically splitting a single LCD panel into two sub-panels, it is possible to save power by powering down the sub-panel of LCD panel 200 which is not being used. As shown in FIG. 1, the DISPOFF signal powers down the entire panel and does not allow powering down a sub-panel (i.e., certain segment drivers) of the entire panel. The power savings results from certain of the segment drivers no longer being clocked and no longer consuming power. Further power savings and probably the greater proportion of power savings is gained from not having to drive or toggle the states of the pixels in sub-panel 204.

5

The invention provides an improved segment driver circuit with the capability of being enabled or powered independent of other segment drivers. Specifically, a VCC pin is provided to each of the modified segment drivers **120a**, **122a**, **124a** and **126a**. These VCC pins are the positive power rails to each segment driver.

As shown in FIG. 2, sub-panel **202** has all **240** rows of pixels but occupies only 120 pixel columns. Thus, to independently operate sub-panel **202**, only segment driver **120a**, which drives the first 120 pixel columns (see description of similar driver **120** of FIG. 1), needs to be controlled.

Therefore, the invention provides control of VCC2 **193** coupled to the VCC pin of segment driver **120a**. When VCC2 **193** is enabled (on), the VCC pin on segment driver **120a** will power-on the segment driver to output pixels. When VCC2 **193** is disabled (switched off), the segment driver **120a** is powered-down or off and cannot drive pixel output to the display panel.

Likewise, another signal VCC1 **195** is coupled to the VCC pins of each segment drivers **122a**, **124a** and **126a**, which drive pixels on the other sub-panel **204**. When VCC1 **195** is on, the segment drivers **122a**, **124a** and **126a** are all powered on and enabled to drive pixel output to the panel **200** (in sub-panel **204**). When VCC **195** is off, all of the segment drivers **122a**, **124a** and **126a** are powered down and cannot drive pixels to the display panel. The three segment drivers **122a**, **124a** and **126a** all utilize a single source for their VCC pins since, according to the embodiment, they drive the same sub-panel.

Thus, sub-panel **202** and sub-panel **204** are capable of being independently powered, and thereby selected by the use of separate signals. VCC1 **195** and VCC2 **193** will be on when both sub-panels must be powered. One skilled in the art will recognize that a single panel may be split into as many logical sub-panels as segment drivers will allow. In this case, panel **200** may be split into four logical sub-panels, one for each segment driver, each segment driver powered by its own VCC signal.

The power source VCC1 **195** and VCC2 **193** are controlled from some software/hardware which selects the functionality of the panel, and therefore, indicates which sub-panels are to be powered (see power control block **660** of FIG. 6).

FIG. 3 shows the casing structure for an information device according to one embodiment of the invention. The information device is capable of functioning both as a cellular phone for telecommunications and as a PDA. The LCD panel **200**, is split logically into sub-panel **204** and sub-panel **202**. The information device has a top outer shell **320** and a bottom outer shell **322** as well as a top inner shell **310** and a bottom inner shell **312**. Top inner shell **310** and its obverse side top outer shell **320** bounds and contains LCD panel **200** and is connected to joint **350** about which the top information device is able to fold. Likewise bottom inner shell **312** with its obverse side bottom outer shell **322**, is also able to fold about joint **350**. Bottom inner shell **312** and bottom outer shell **322** may both contain input keys such as alpha-numeric and function keys with which a user can input data, make telephone calls and/or control operation of the information device. Top inner shell **310** has an open area **300** which may be open aperture or some transparent panel which closed upon LCD panel **200**, makes visible the image in sub-panel **204**, thus allowing monitoring of the friction for which sub-panel **202** is intended.

The information device is "closed" when bottom inner shell **312** and top inner shell **310** about one another by folding

6

the Information device about joint **350**. When the information device is closed, the open area **300** closely abuts the area of sub-panel **204** such that the image contents (pixels) on output sub-panel **204** are visible to the user. When closed, the outer shell **320**, which may or may not be transparent (excepting open area **320**), covers sub-panel **202** which is contained in inner shell **312**. Upon closing the information device, a switch, relay or contact disposed about or within joint **350** will operate to power down the driver(s) for sub-panel **202** while leaving the driver(s) for sub-panel **204** powered-up. This relay or contact will toggle the VCC pins of the appropriate segment drivers as discussed in FIG. 2. Thus, when the information device is closed, sub-panel **204** is operational while sub-panel **202** is disabled thereby saving power and screen life. This still allows the user to monitor the telecommunications function of the information device. Further, by using one physical display rather than two separate physical displays, the information device saves by reducing device complexity and cost.

As shown in FIG. 3, when closed, sub-panel **204** shows a telephone number, a "BAT" indicator indicating the level of battery life in the device and a "SIG" indicator all which are still visible to the user. Underneath, the portion of LCD panel **200** covered by outer shell **320**, i.e., sub-panel **202**, is powered down and inoperative. Thus, the telecommunications display of the information device can be viewed on a sub-panel while the information device one display is closed and data sub-panel is powered down.

When the information device is in the "open" position, both sub-panels **202** and **204** are powered. In this mode, both the data function and telecommunications functions can be displayed on panel **200**. Thus, all segment drivers are powered when the Information device is open. In this embodiment, the selection of individual sub-panels via software is not needed since the position of the information device makes the selection.

FIG. 4 shows a information device with two separate displays, according to one embodiment of the invention.

When an information device, by design has two separate displays located on different physical planes, the invention provides for powering down one display, while keeping the other active, depending on what function is being carried out on the device. Shown in FIG. 4 is a first LCD panel **100** and a second display LCD panel **400**. LCD panel **400**, if it is to use the same controller signals as the LCD panel **100**, must have an equal number of rows of pixels as LCD panel **100**, and consequently, the same duty cycle. Without the same vertical resolution, the controller would need to refresh to the larger of the two resolutions thereby undermining bus bandwidth, memory resources and frame rates.

System software and the video controller would treat the combination of LCD panel **100**, which has a 640 by 240 resolution, and LCD panel **400** with a 160 by 240 resolution, as a single logical panel of 800 by 240. Segment driver **420** which drives LCD panel **400** receives the first set of the input data bits from DISPDATA signal **150** and upon filling its shift registers, propagates a daisy-chaining command to segment driver **121**. Segment driver **121** is modified from segment driver **120a** of FIG. 2 in that the EIO2 pin is extended to EIO1 output pin of segment driver **420**. VSS is now connected to EIO2 so that segment driver **420** receives the input stream before segment drivers of LCD panel **100**.

The separate display panels **100** and **400** are controlled similar to the split-screen (sub-panel) embodiment described above for FIG. 2. Each of the segment drivers **121**, **123**, **125** and **127** of panel **100** and segment driver **420** is provided

with separate VCC pins. The invention also provides a power source VCC2 493 coupled to the VCC pin of segment driver 420. When VCC2 493 is on, the VCC pin on segment driver 420 will power-on the segment driver 420 to output pixels to panel 400. When VCC2 493 is off, the segment driver 420 is powered-down or off and cannot drive pixel output to panel 400.

Likewise, another power source VCC1 495 is coupled to the VCC pins of each segment drivers 121, 123, 125 and 127, which drive pixels on display panel 100. When VCC1 495 is on, the segment drivers 121, 123, 125 and 127 are all powered on and enabled to drive pixel output to the panel 100. When VCC1 495 is off, all of the segment drivers 121, 123, 125 and 127 are powered down and cannot drive pixels to the display panel. The three segment drivers 121, 123, 125 and 127 all utilize a single signal for their VCC pins since, according to the embodiment, they drive the same sub-panel.

Thus, display panels 100 and 400 are capable of being independently powered, and thus, independently selected. VCC1 495 and VCC2 493 will be on when both display panels must be powered.

FIG. 5 is a system diagram of a computer system in which a dual LCD panel display system according to one embodiment of the invention may be utilized.

FIG. 5 shows well known elements of a computer system such as a host CPU 520, a shared memory 510 and a display controller 530. In this embodiment, LCD display controller 530 drives and controls a primary graphics LCD display panel 540 and a secondary LCD panel 550. Display controller 530 provides the signals shown in FIG. 4 such as DISPDATA 150. Specifically, display data 535 of FIG. 5 corresponds to DISPDATA 150 of FIG. 4 and control clocks 537 of FIG. 5 refer to all other controller signals, such as FLM 180 which are provided to the LCD. Primary graphics LCD display panel 540 and secondary LCD panel 550 are shown as single blocks in FIG. 5, but include all necessary segment drivers and common drivers, as well as internal input lines and dividers, as shown in FIG. 4 to enable output to the actual LCD panels.

Shared memory 510 services both host CPU 520 and LCD display controller 530 by way of an address bus 524. Address bus 524 carries memory addresses of shared memory 510 to/from CPU 520 and display controller 530. Data bus 526 is capable of sending and receiving to either the CPU 520 or the display controller 530. Data bus 526 delivers raw data to the LCD display controller 530 from which display controller 530 can generate actual display data 535 which are pixels to be output on LCD panels 540 and 550. Control bus 528 is used to control the flow of information from shared memory 510 which is delivered over data bus 526. A bus control (handshake) line transmits request and grant pairs to arbitrate use of the address, control and data bus between CPU 520 and display controller 530.

CPU 520 is a central processing unit, such as the Intel Pentium™ processor and is capable of processing information according to code delivered to it by software or hardware through data bus 526, address bus 524 and control bus 528. The structural detail and functioning of CPU 520 as well as shared memory 510, display controller 530, address bus 524, data bus 526 and control bus 528 are well known to one reasonably skilled in the art of computer systems and will not be described further.

FIG. 5 shows a key feature of the invention which is power control block 560. Power control block 560 may be composed of multiplexers, switches, and transistors and is implemented in accordance with the specifications of CPU

520 and system architecture. Power control block 560 centrally controls the selection and thus, powering of primary LCD panel 540 and secondary LCD panel 550 through the use of selector lines 562 and 564. Power control block 560 drives selector line 562 active when secondary LCD panel 550 is to be enabled for output. If both primary display panel 540 and secondary LCD panel 550 are to be enabled for output, then power control block 560 will also activate selector line 564, such that the primary graphics display panel 540 will also be enabled for output. In the case where the user or software only requests that secondary LCD panel 550 be enabled but not primary LCD panel 540, the power control block 560 will deactivate selector line 564.

The signals VCC1 495 and VCC2 493 of FIG. 4 may originate directly from selector lines 564 and 562, respectively. When selector lines 564 and 562 dictate that both panels 540 and 550 are to be enabled for output, VCC1 495 and VCC2 493 of FIG. 4 will be enabled. Likewise, when only secondary LCD panel 550 is to be enabled, selector line 564 can enable VCC2 493 and disable VCC1 495, thereby powering down the primary graphics LCD display panel 540.

CPU 520, when instructed that only the telecommunications function of the information device is to be used, will send a command to power control block 560. A transistor-implemented switching mechanism or multiplexer will then drive selector line 562 active, while inactivating selector line 564. The switching mechanism or multiplexer(s) are capable of necessary control signals from the CPU in response to a change in function of the information device. The selector lines are independently switched on/off within the power control block allowing more control over power usage.

FIG. 6 is a system diagram of a computer system in which a dual LCD panel display system according to one embodiment of the invention may be utilized.

Where a split-screen LCD panel embodiment is desired, a system similar to that shown in FIG. 2 may be equipped so that selector lines 662 and 664 control enabling of certain segments by providing signals VCC1 195 and VCC2 193, respectively (see FIG. 1 and associated description). For instance, when both selector lines 662 and 664 set high both VCC1 195 and VCC2 193, all four segment drivers and, consequently, both sub-panels 643 and 645 will be powered. When VCC2 193 is set high (by selector line 662) and VCC1 195 is set low, only segment driver 120 and consequently, sub-panel 645 will be powered. Within power control block 660, the selector lines 662 and 664 are independently switched using transistor or multiplexers upon receiving commands from CPU 520 regarding device function.

Power control block 660 which generates the signals on selector lines 662 and 664 is coupled to host CPU 520 and consists of components similar to those shown in and described for FIG. 5. Further, CPU 520, shared memory 510, address bus 524, data bus 526, control bus 528 and bus control handshake 529 operate similar to their counterparts shown in and described for FIG. 5 and will not be repeated. LCD display controller 630 of FIG. 6 may be slightly different from counterpart controller 530 in that controller 630 has only one physical display to drive (as opposed to two) and thus, may not require the hardware/software complexity of controller 530 of FIG. 5.

While the present invention has been particularly described with reference to the various figures, it should be understood that the figures are for illustration only and should not be taken as limiting the scope of the invention. Many changes and modifications may be made to the

invention, by one having ordinary skill in the art, without departing from the spirit and scope of the invention.

What is claimed is:

1. In an information device having a CPU, display controller and a display panel, said display panel split logically into sub-panels, an apparatus comprising:

a plurality of segment drivers coupled between said display panel and said display controller, said segment drivers receiving input data from said controller, said segment drivers translating said data into pixels displayable on said display panel; and

a power control block coupled to said CPU and to said segment drivers to disable a first power source which powers down a first set of said segment drivers, said powering down disabling a first set of sub-panels of said display panel from outputting pixels, said power control block disabling said first power source upon receiving a command from said CPU that said first set of sub-panels are to be powered down, said information device functioning as one of a cellular communications device and a personal digital assistant, said first set of sub-panels displaying information relevant to said personal digital assistant function, further wherein said display panel includes a second set of sub-panels displaying information relevant to said cellular communications functions.

2. An apparatus according to claim 1 wherein said power control block disables a second power source which powers down a second set of said segment drivers, said powering down disabling a second set of sub-panels from outputting pixels, said power control block disabling said second power source upon receiving a command from said CPU that said second set of sub-panels are to be powered down.

3. An apparatus according to claim 2 wherein said first power source and said second power source are independently switched by said power-control block to enable outputting of pixels on said first set of sub-panels and said second set of sub-panels, respectively.

4. An apparatus according to claim 1 wherein said information device has a normally open latch, said power control block to disable said first power source when said latch is closed.

5. In an information device having a CPU, display controller, and two display panels, an apparatus comprising:

a first set of segment drivers coupled to said display controller to receive as input a first set of data, said first set of segment drivers translating said first set of data into pixels output on a first of said display panels;

a second set of segment drivers coupled to said display controller and said first set of segment drivers to receive a second set of data, said second set of segment drivers translating said second set of data into pixels output on a second of said display panels; and

a power control block coupled to said CPU and to said first and second set of segment drivers to disable a first power source which powers down said second set of segment drivers, said powering down disabling said second display panel from outputting pixels, said information device functioning as one of a cellular communications device and a personal digital assistant, said second displaying panel displaying information relative to said personal digital assistant function, further wherein said first display panel displaying information relevant to said cellular communications function.

6. An apparatus according to claim 5 wherein said power control block disables a second power source which powers

down said first set of segment drivers, said powering down disabling said first display panel.

7. An information device having a single display panel logically split into a first and second sub-panel, said device comprising:

a top shell including a top inner shell and top outer shell, said top outer shell on the opposing side of said top inner shell, said top inner shell containing said display panel;

a joint coupled to said top shell for folding said device; and

a bottom shell coupled to said top shell through said joint, said bottom shell including a bottom inner shell and a bottom outer shell, said bottom outer shell on the opposing side of said bottom inner shell, said bottom shell having an open area, wherein said open area leaves visible said first sub-panel and hides said second sub-panel when said device is closed about said joint, wherein when said device is closed, a first power signal is disabled to power down said second sub-panel and a second power signal is enabled to power said first sub-panel, said information device functioning as one of a cellular communication device and a personal digital assistant, said second sub-panel displaying information relevant to said personal digital assistant function, and said first sub-panel displaying information relevant to said cellular communications function.

8. An information device according to claim 7 wherein when said device is open, said first signal is enabled to power said second sub-panel and said second power signal is enabled to power said first sub-panel.

9. An information device according to claim 7 wherein said information device is capable of performing a certain function when closed about said joint, said function monitored through said open area.

10. An information device having a two separate display panels, each display panel on separate physical planes, said device comprising:

a top shell including a top inner shell and a top outer shell, said top outer shell on the opposing side of said top inner shell, said top inner shell containing both said display panels;

a joint coupled to said top shell for folding said device; and

a bottom shell coupled to said top shell through said joint including a bottom inner shell and a bottom outer shell, said bottom outer shell on the opposing side of said bottom inner shell, said bottom shell having an open area, wherein said open area leaves visible said first display panel and hides said second display panel when said device is closed about said joint, wherein when said device is closed, a first power signal is disabled to power down said second display panel and a second power signal is enabled to power said first display panel, said information device functioning as one of a cellular communications device and a personal digital assistant, said second display panel displaying information relevant to said personal digital assistant function, and said first display panel displaying information relevant to said cellular communications function.

11. An information device according to claim 10 wherein when said device is open, said first power signal is enabled to power said second display panel and said second power signal is enabled to power said first display panel.

* * * * *



US006035197A

United States Patent [19]
Haberman et al.

[11] Patent Number: 6,035,197
[45] Date of Patent: Mar. 7, 2000

[54] METHOD AND SYSTEM FOR PROVIDING A HANDOFF FROM A CDMA CELLULAR TELEPHONE SYSTEM
[75] Inventors: Michael Haberman, Morris Plains; Glenn Pierson, East Hanover, both of N.J.
[73] Assignee: Cellco Partnership, Bedminster, N.J.
[21] Appl. No.: 08/366,352
[22] Filed: Dec. 29, 1994
[51] Int. Cl.⁷ H04Q 7/20
[52] U.S. Cl. 455/439; 455/432; 455/436; 370/331
[58] Field of Search 379/58, 59, 60; 455/33.1, 33.2, 54.1, 422, 432, 434, 435, 436, 437, 438, 439, 442, 517, 524, 525; 375/216, 200, 205, 206; 370/18, 331, 332, 335, 465

5,243,598 9/1993 Lee .
5,257,401 10/1993 Dahlin et al. 455/33.2
5,267,261 11/1993 Blakeney, II et al. .
5,276,906 1/1994 Felix .
5,278,892 1/1994 Bolliger et al. .
5,278,991 1/1994 Ramsdale et al. .
5,285,469 2/1994 Vanderpool .
5,289,499 2/1994 Weerackody .
5,295,153 3/1994 Gudmundson .
5,301,356 4/1994 Bodin et al. .
5,309,503 5/1994 Bruckert et al. .
5,313,489 5/1994 Menich et al. .
5,317,623 5/1994 Sakamoto et al. .
5,323,446 6/1994 Kojima et al. .
5,327,574 7/1994 Monma et al. .
5,327,577 7/1994 Uddenfeldt .
5,345,467 9/1994 Lomp et al. 375/1

Primary Examiner—Dwayne D. Bost
Assistant Examiner—Nay Maung
Attorney, Agent, or Firm—McDermott, Will & Emery

[56] References Cited

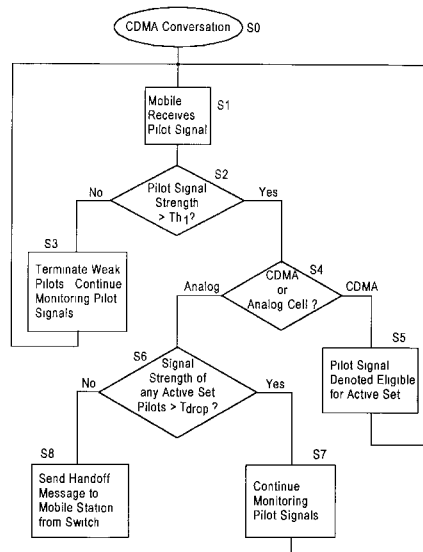
U.S. PATENT DOCUMENTS

4,754,453 6/1988 Eizenhöfer .
4,765,753 8/1988 Schmidt .
4,901,307 2/1990 Gilhousen et al. .
4,984,247 1/1991 Kaufmann et al. .
5,042,082 8/1991 Dahlin .
5,101,501 3/1992 Gilhousen et al. 379/60
5,109,390 4/1992 Gilhousen et al. .
5,117,502 5/1992 Onoda et al. .
5,127,100 6/1992 D'Amico et al. .
5,164,958 11/1992 Omura .
5,179,571 1/1993 Schilling .
5,195,090 3/1993 Bolliger et al. .
5,200,957 4/1993 Dahlin .
5,210,771 5/1993 Schaeffer et al. .
5,224,120 6/1993 Schilling .
5,228,053 7/1993 Miller et al. .
5,235,615 8/1993 Omura .
5,241,685 8/1993 Bodin et al. .

[57] ABSTRACT

A mobile assisted handoff of a mobile station transitioning from a CDMA portion of a cellular telecommunication network to an analog portion of the network is provided as a function of a CDMA pilot signal transmitter located at each analog cell in the analog portion of the network. The mobile station monitors a CDMA pilot signal transmitted from each CDMA cell and each analog cell of the cellular telecommunication network, measures the signal strength of each received CDMA pilot signal and transmits the signal strength to the MTSO. The MTSO determines whether the pilot signal is associated with a CDMA base station or an analog base station. When the mobile station receives a pilot signal from an analog cell whose signal strength is above a first predetermined threshold and the received signal strengths of all pilot signals of the CDMA base stations with which the mobile station is currently in communication are below a second predetermined threshold, the MTSO directs a CDMA to analog handoff message to the mobile station.

13 Claims, 3 Drawing Sheets



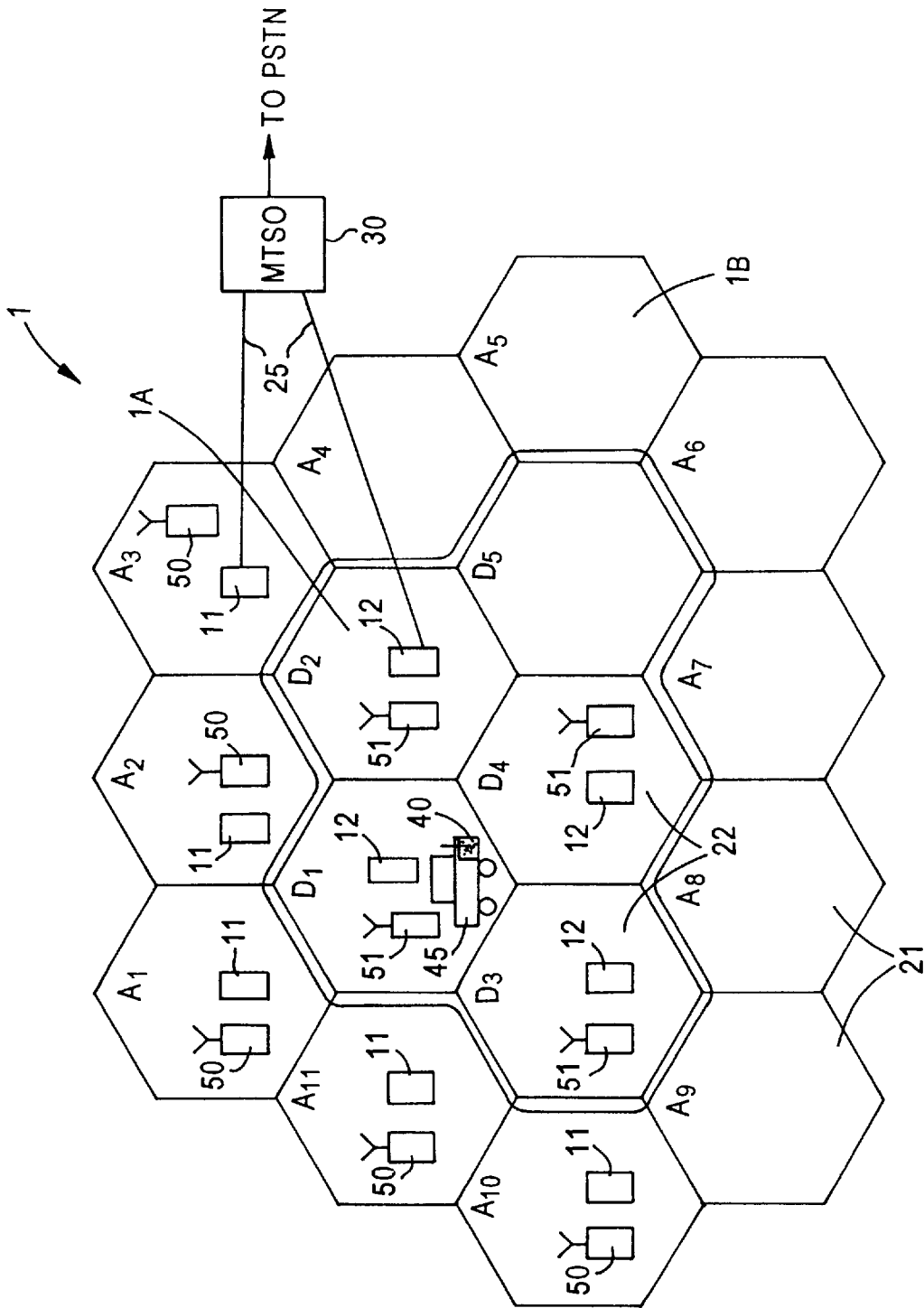


FIG. 1

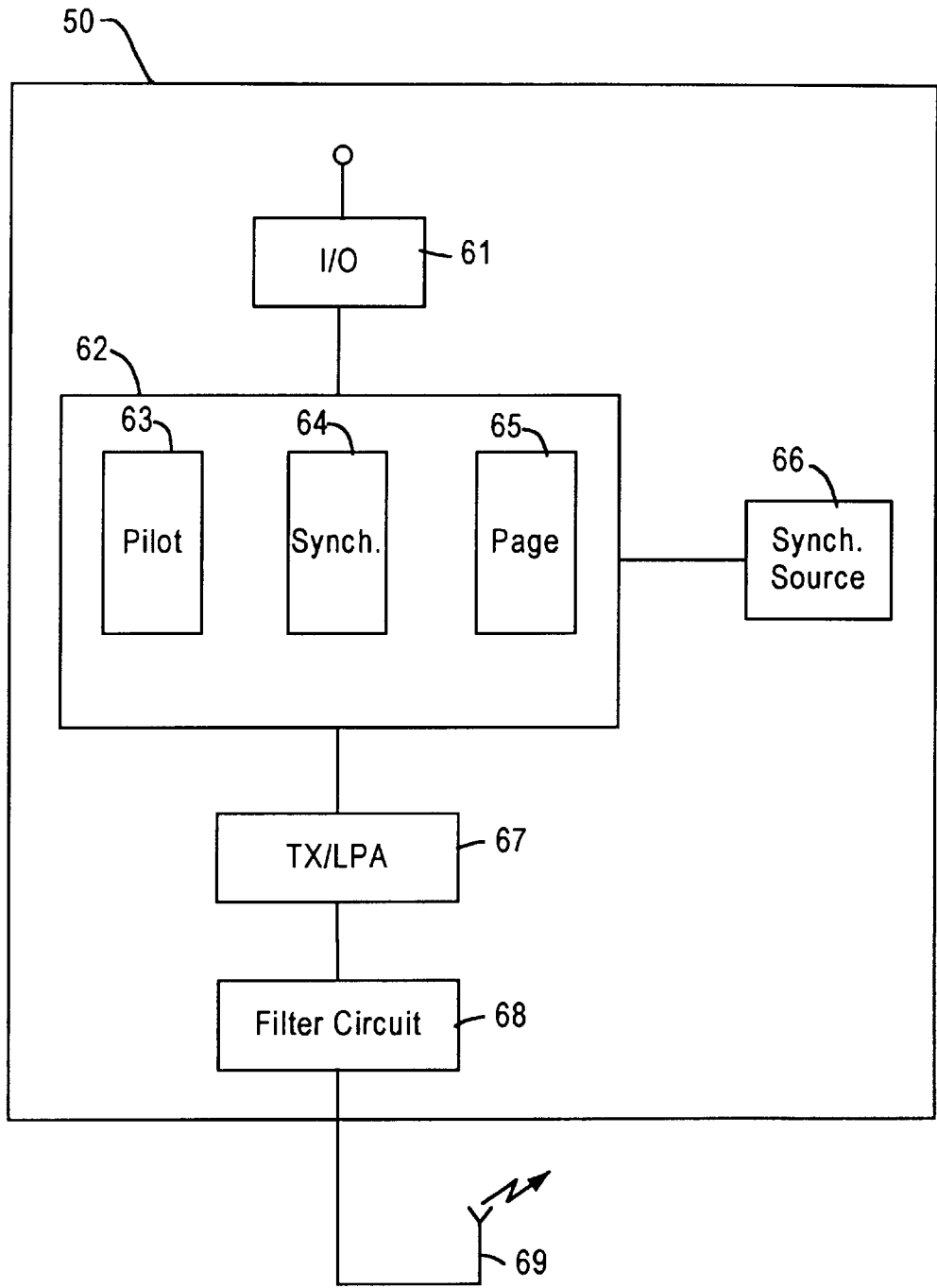


FIG. 2

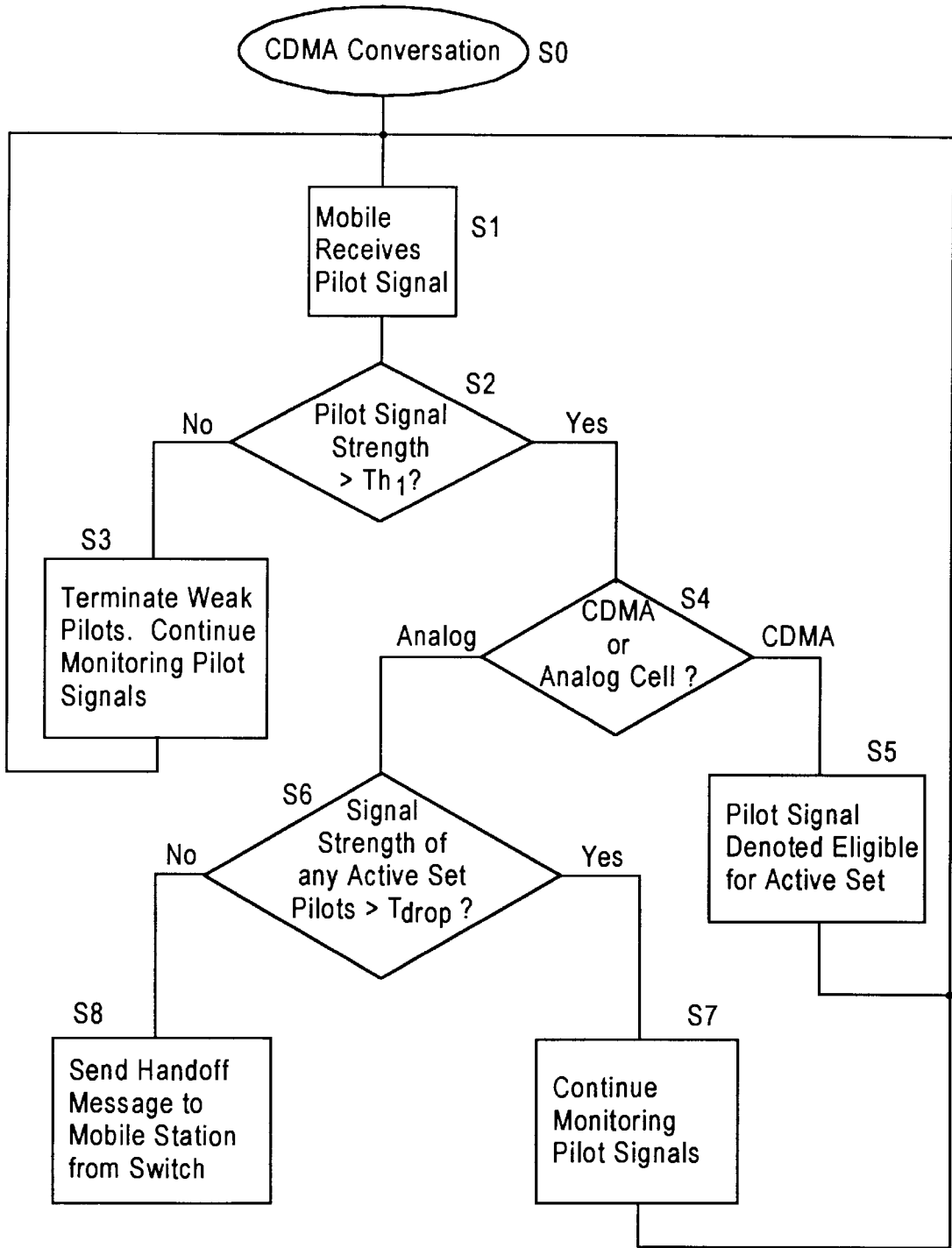


FIG. 3

**METHOD AND SYSTEM FOR PROVIDING A
HANDOFF FROM A CDMA CELLULAR
TELEPHONE SYSTEM**

FIELD OF THE INVENTION

The present invention relates generally to wireless telecommunication systems. More particularly, the present invention relates to a method and system for providing handoff of a mobile telephone from a CDMA cellular telecommunication system to an analog cellular telecommunication system.

BACKGROUND INFORMATION

Wireless telecommunication systems provide information services traditionally provided by land-line or copper wire systems. Examples of wireless communications applications include Advanced Mobile Phone Service (AMPS) analog cellular service and AMPS-D digital cellular service in North America, and Group Special Mobile (GSM) cellular service in Europe.

Although the particular application may vary, the components of a wireless communication system are generally similar. For example, a wireless communication system usually includes a radio terminal or mobile station, a radio base station, a switch or network control device, often referred to as a mobile telephone switching office (MTSO), and a network to which the wireless communications system provides access, such as the Public Switched Telephone Network (PSTN).

The various wireless communication applications use different modulation techniques for transmitting information to more efficiently utilize the limited available frequency spectrum. For example, frequency division multiple access (FDMA), time division multiple access (TDMA) and code division multiple access (CDMA) modulation techniques are used to build high-capacity multiple access systems. Telecommunication systems designed to communicate with many mobile stations occupying a common radio spectrum are referred to as multiple access systems.

For example, in an FDMA analog cellular system, such as an AMPS analog cellular radio system, the available frequency spectrum is divided into a large number of radio channels, e.g., pairs of transmit and receive carrier frequencies, each of which corresponds to a message transmission channel. The bandwidth of each transmit and receive frequency channel is narrowband, generally 25–30 kHz. Thus, the FDMA system permits information to be transmitted in a bandwidth comparable to the bandwidth of the transmitted information, such as a voice signal. The cellular service area in the FDMA system is generally divided into a plurality of cells, each cell having a set of frequency channels selected so as to minimize co-channel interference between cells.

Frequency division is often combined with time division so that transmission circuits are trunked in both the frequency and time domain, e.g., a FD/TDMA system. In a digital FD/TDMA (commonly referred to as TDMA) cellular system, a narrowband frequency channel is reformatted as a digital transmission path which is divided into a number of time slots. The data signals from different calls are interleaved into assigned time slots and sent out with a correspondingly higher bit rate, the time slot assigned to each mobile station being periodically repeated. Although the TDMA bandwidth may be somewhat larger than the FDMA bandwidth, a bandwidth of approximately 30 kHz is generally used for AMPS-D digital TDMA cellular systems.

A very different approach to cellular multiple access modulation is CDMA. CDMA is a spread spectrum technique for transmitting information over a wireless communication system in which the bandwidth occupied by the transmitted signal is significantly greater than the bandwidth required by the baseband information signal (e.g., the voice signal). Thus, CDMA modulation spectrally spreads a narrowband information signal over a broad bandwidth by multiplex modulation, using a codeword to identify various signals sharing the same frequency channel. Recognition of the transmitted signal takes place by selecting the spectrally-coded signals using the appropriate codeword. In contrast to the narrowband channels of approximately 30 kHz used in FDMA and TDMA modulation techniques, a CDMA system generally employs a bandwidth of approximately 1.25 MHz or greater.

Regardless of the modulation technique used in a cellular telecommunication system, when a mobile station is in communication with its base station, for example to provide telephone service between a mobile station and a calling party, the cellular system must maintain uninterrupted service for the call despite movement of the mobile station through the cellular system. For example, in an analog cellular system, when the mobile station transitions from one cell to another cell, the mobile station must change frequencies because each cell supports a different set of frequencies. The process by which a cellular telecommunication system enables a mobile station to maintain an established connection when moving through cells of a cellular system is referred to as “handoff,” and is generally controlled by the MTSO.

In a conventional analog cellular system, a handoff is triggered when the base station currently providing the link between the mobile station and the MTSO detects that the received signal strength from the mobile station has dropped below a predetermined level. The low signal strength from the mobile station usually indicates that the mobile station is approaching the boundary of the cell. When the received signal strength is below the predetermined value, the base station requests the MTSO determine whether another base station, e.g., a neighboring base station, is receiving a stronger signal from the mobile station than the current base station.

In response to the request from the current base station, the MTSO sends a message to the appropriate neighboring base stations to measure the received signal strength from the mobile station. The neighboring base stations, using a scanning receiver, monitor the frequency channel of the mobile station and measure the received signal strength, if possible. The measurements made by the neighboring stations are reported to the MTSO. If one of the neighboring base stations receives the mobile station signal above a predetermined level, then the MTSO directs a handoff of the mobile station from its current base station to a new base station in an adjoining cell. In particular, the MTSO informs the mobile station of a new frequency to be used with the new base station, while the MTSO also switches the call from the current base station to the new base station. If the handoff is unsuccessful, however, the call will be lost, e.g., terminated. This type of handoff is often referred to as a system-assisted handoff because the cellular system controls the detection of the need for, and the execution of, the handoff.

Another type of handoff is referred to as a mobile-assisted handoff (MAHO). For example, in a digital CDMA cellular system, each base station transmits a CDMA pilot signal on a common frequency, each pilot signal being differentiated

by its phase offset compared to other pilot signals. A mobile station located in a digital CDMA cellular system regularly monitors the pilot signal strength received from the various pilot signals of neighboring base stations. The mobile station detects when the received signal strength of a pilot signal from its current base station has dropped below a predetermined level and the received signal strength of a neighboring base station pilot signal exceeds a predetermined level. The mobile station transmits these signal strength measurements to the MTSO via the base station with which the mobile station is in communication. The MTSO directs a handoff from one base station to another base station based on the signal strength measurements made by the mobile station.

A conventional narrowband analog cellular system, such as an AMPS FDMA cellular system, cannot support MAHO because in the analog system there is no pilot signal, the mobile station does not take measurements of the signals transmitted by the analog base station, and the handoff is controlled by the base stations and MTSO. Moreover, a 30 kHz analog cell base station cannot transmit a 1.25 MHz CDMA pilot signal.

Similar to the CDMA system MAHO, in a digital TDMA cellular system, each base station can transmit a unique 30 kHz beacon signal that is received and measured by the mobile station and reported to the MTSO. Based on the frequency of the beacon signal, the MTSO can identify the cell site associated with each beacon signal. When the received beacon signal strength drops below a predetermined value, then the mobile station reports the measurement to the MTSO, via a base station, and the MTSO can direct a handoff of the mobile station to another base station, either analog or digital TDMA, associated with a sufficiently strong beacon signal.

A TDMA to analog handoff is possible because both the TDMA system and the analog system are narrowband systems using 30 kHz frequency channels. Thus, a 30 kHz analog cell base station can support a TDMA MAHO handoff using a 30 kHz TDMA pilot signal. The TDMA MAHO has problems with false handoffs, however, because a mobile station can receive a 30 kHz signal that is not a beacon signal but rather is a communication signal from another mobile station. For example, a mobile station at a high elevation may transmit a 30 kHz signal on the same frequency as a particular beacon signal that is mistakenly detected as a beacon signal by another mobile station at a lower elevation, thus causing an unwarranted handoff and possibly a lost call.

In CDMA cellular telecommunication systems, a handoff is usually accomplished via a "soft handoff" from one base station to another base station. In a soft handoff, the mobile station is in communication with more than one base station simultaneously, and thus the mobile station performs a "make before break" transition from one base station to another base station. The soft handoff is possible because in CDMA cellular telecommunication systems, numerous mobile stations communicate with each base station on the same frequency channel, each mobile station having a unique spreading code for distinguishing the information signals broadcast by the numerous mobile stations. Thus, when a mobile station moves from one CDMA cell to another CDMA cell, the same frequency is used in each CDMA cell and the unique spreading code identifies the mobile station to the new base station.

In contrast to the soft handoff used in CDMA cellular systems, narrowband frequency modulation systems, such

as FDMA and TDMA systems, employ a "hard handoff." The hard handoff, which is a "break before make" connection, is necessary in narrowband cellular systems because each mobile station is communicating with a base station on a particular narrowband frequency channel. The available frequency channels in adjoining cells differ, and thus when a mobile station moves from one cell to another cell, a new frequency channel must be used.

The advantage of employing a narrowband modulation scheme, such as FDMA, would be defeated if such a system utilized a soft handoff. For example, a narrowband FDMA cellular telecommunication system using a soft handoff would require that the mobile station simultaneously communicate with at least two base stations in adjoining cells on either the same or different frequencies. If the mobile station communicated on the same frequency to two adjoining base stations, co-channel interference would result from two base stations broadcasting on the same frequency to the mobile station, precisely the type of interference the narrowband system was designed to avoid. Alternatively, requiring the mobile station transmit its communication signal to at least two base stations in adjoining cells on two separate frequencies simultaneously is not possible because such simultaneous communication capability is not possessed by conventional mobile stations.

As spread spectrum modulation techniques, such as CDMA, are implemented within existing cellular telecommunications systems, compatibility issues arise regarding the integration of CDMA cell sites into existing analog cellular telecommunications systems. The commercial success of a cellular service provider is dependent in part on the provider's ability to provide seamless integration of new CDMA cell sites into existing analog systems, and in particular, the ability to have unnoticeable handoffs as a mobile station transitions from the CDMA portion of the system into the analog portion of the system.

One problem with integrating CDMA cells into existing analog cellular systems is the inability of conventional mobile stations to support CDMA and analog communications simultaneously. Conventional mobile stations provide a dual mode capability for generating and receiving spread spectrum and narrowband signals. The mobile stations, however, can operate in only one mode at a time. Therefore, while a mobile station is communicating on the cellular system via a CDMA channel, e.g., a 1.25 MHz channel, it is not possible for the mobile station to simultaneously communicate via a narrowband channel of the system, e.g., a 30 kHz channel.

Another problem is that a narrowband base station cannot receive a spread spectrum CDMA signal to measure the received signal strength necessary to perform a system-assisted handoff, as the CDMA signal is spread over a bandwidth that is larger than the narrowband channel which the narrowband base station is designed to receive. Also, a narrowband base station transmits a narrowband signal, e.g., a 30 kHz signal, and thus cannot provide a CDMA pilot signal to be received and measured by the mobile station to facilitate a MAHO to an analog base station. The handoff of a mobile station from a CDMA cell site to an analog cell site represents one of the more significant problems with integrating CDMA cell sites into existing cellular systems.

Current approaches to the problem of handoff of a mobile station from a CDMA portion of a cellular telecommunications network to an analog portion of the telecommunications network are inefficient and affect performance. For example, an additional analog cell can be placed in the

5

CDMA cell for an internal handoff of the mobile station prior to handoff of the mobile station to the existing analog system.

Under this approach, when a handoff of a mobile station from the CDMA portion of the system to the analog portion is necessary, a handoff is first performed from the CDMA base station to the additional analog base station in the same cell, i.e., the CDMA cell is actually a digital/analog cell, capable of supporting both types of modulation. Assuming, however, that the mobile station is transitioning beyond the boundary of the digital/analog cell to an analog cell, another handoff is required to an analog base station of the existing analog system. Thus, two hard handoffs are required for the transition of a mobile station across only one boundary, whereas only one handoff would be desirable.

In addition to requiring an unnecessary handoff within the CDMA cell, the above approach presents other problems. For example, the handoff from the CDMA base station to the analog base station in the same cell is a "blind handoff." As described above, the bandwidths of spread spectrum and narrowband frequency channels are incompatible, as are the types of modulation techniques. Thus, the CDMA to analog handoff in the same cell site is directed without the benefit of knowing that the target analog base station is indeed the best target base station, or with what strength the communication signals from the mobile station will actually be received by the target analog base station. As a result of the lack of information on the suitability of the handoff, it is possible that the handoff might not be properly executed, resulting in a lost call, e.g., termination of the call.

Another problem with this approach to CDMA to analog handoffs is the necessity of reducing the available coverage area of the CDMA cell. A benefit of a CDMA cell, as well as digital cells generally, is that the cellular service area provided by the CDMA cell, often referred to as its "footprint," is larger than the footprint of conventional analog cells. However, in order to perform a CDMA to analog handoff within the CDMA cell that has a significant chance of success, the footprint of the CDMA cell must be reduced so that there is sufficiently strong analog coverage when the CDMA to analog handoff in the same cell site actually occurs.

A further problem with this approach to CDMA to analog handoff is unwarranted handoffs to the analog portion of the cellular telecommunications system. For example, using the MAHO scheme that is implemented in a CDMA cellular system, a handoff is directed when the received signal strength of the pilot signal from a base station with which the mobile station is in communication drops below a predetermined level. Thus, a mobile station may enter an area within the coverage area of a CDMA cell but for some reason, the CDMA pilot signals received by the mobile station are attenuated. For example, the mobile station could enter an underground parking garage. If the pilot signals are below the predetermined value, then the MTSO will direct a handoff to the analog portion of the cellular system. The handoff, however, generally will not solve the problem of the mobile station's poor reception in the underground parking garage, and thus prematurely takes the mobile station off the CDMA portion of the cellular system. In conventional cellular systems having analog and CDMA portions, there is generally no provision for handoff from an analog portion of the system back to the CDMA portion of the system. Therefore, this approach to CDMA to analog handoff allows unwarranted and unnecessary handoffs to the analog portion of a cellular telecommunications system when the actual preference of the cellular system is to keep the mobile station on the digital portion of the system as long as possible.

6

Another approach to CDMA to analog handoff is a direct handoff from the CDMA cell site to the desired analog cell site, thus avoiding an interim analog handoff. This approach, however, has some of the same problems as the above CDMA to analog handoff method. In particular, the direct handoff approach is a blind handoff to the analog cell site due to the lack of information available to the mobile station about the analog base station while the mobile station is in communication with the CDMA cell site.

Therefore, a need exists for a method of directing a handoff of a mobile station from a spread spectrum portion of a cellular system to an analog portion of the cellular system which minimizes additional equipment costs, avoids unnecessary handoffs to the analog portion of the system, and ensures continuation of an existing call in the analog portion of the system upon completion of the handoff.

SUMMARY OF THE INVENTION

The method according to the present invention provides a mobile assisted handoff of a mobile station transitioning from a CDMA portion of a cellular telecommunications network to an analog portion of the network utilizing a CDMA pilot signal transmitter located at each analog cell in the analog portion of the network.

According to the present invention, the mobile station constantly monitors the CDMA pilot signal transmitted from each CDMA cell. The mobile station also monitors the CDMA pilot signal transmitted from each analog cell. The mobile station measures the signal strength of each received CDMA pilot signal and transmits the signal strength to the MTSO via the base station(s) with which the mobile station is currently in communication. The MTSO includes the capability of distinguishing which base station, e.g., a CDMA base station or an analog base station, with which each pilot signal is associated based on the phase offset of the pilot signal relative to a time standard of the cellular system.

When the mobile station receives a pilot signal from an analog cell whose signal strength is above a first predetermined threshold level, and the received signal strengths of all pilot signals of the CDMA base station with which the mobile station is currently in communication are below a second predetermined threshold level, the MTSO directs a CDMA to analog handoff message to the mobile station. As a result, the mobile station is handed off to the appropriate analog base station in the analog portion of the system.

The method according to the present invention minimizes the additional equipment expenses needed for implementing an efficient and reliable CDMA to analog handoff by placing a conventional CDMA pilot signal transmitter at each analog cell site and providing the MTSO with the capability of recognizing and processing measurements made by a mobile station of CDMA pilot signals broadcast from analog cell sites. In addition, the method according to the present invention prevents unwarranted handoffs by directing a CDMA to analog handoff only when the CDMA pilot signal from an analog cell site is the best candidate for handling the mobile station compared to other CDMA cell pilot signals.

In further embodiments of the present invention, the method provides for handoff of a mobile station from a CDMA cellular telecommunications system to a personal communications system (PCS) or to a digital TDMA cellular telecommunications system.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 shows a mobile station transitioning through a cellular telecommunication system according to the present

invention including a CDMA portion of the cellular telecommunication system and an analog portion of the cellular telecommunication system.

FIG. 2 shows a block diagram of a pilot signal transmitter according to the present invention.

FIG. 3 is an illustrative flowchart of the CDMA to analog handoff method according to the present invention.

DETAILED DESCRIPTION OF THE INVENTION

FIG. 1 shows a cellular telecommunication system 1 according to the present invention having a digital portion 1A and an analog portion 1B. The digital portion 1A includes a plurality of (for example, five) digital cells 22, labeled D₁ to D₅. A digital cell 22 supports cellular telecommunication using digitally modulated signals, such as CDMA signals. The digital cells 22 also can be digital/analog cells, which have the capability of supporting both digital cellular communications and analog cellular communications. The analog portion 1B includes a plurality of (for example, eleven) analog cells 21, labeled A₁ to A₁₁. An analog cell 21 supports cellular telecommunication using analog signals, such as FDMA modulated signals. Each digital cell 22 includes a base station 12 and a digital pilot signal transmitter 51. Each analog cell 21 includes a base station 11 and a digital pilot signal transmitter 50 according to the present invention.

FIG. 2 shows an exemplary digital pilot signal transmitter 50 according to the present invention that can be placed in an analog cell 21. The pilot signal transmitter 50 includes, for example, an input/output (I/O) device 61 which allows connection via, for example, a dedicated phone line or wireless connection, to the MTSO 30 for parameter modification and for monitoring operation of the pilot signal transmitter 50. Coupled to the I/O device 61 is a line card shelf 62. Contained in the line card shelf 62 are a pilot signal line card 63, a synchronization line card 64 and a paging signal line card 65. The pilot signal line card 63 generates a pilot signal compatible with the modulation architecture used in the digital portion 1B of the cellular system 1, e.g., CDMA modulation. The synchronization line card 64 provides for synchronization of the pilot signal generated by the pilot signal line card 63. The paging signal line card 65 provides for the generation of paging signals. The line cards 63, 64 and 65 are powered by a power supply included in line card shelf 62.

A synchronization source 66 is coupled to the synchronization line card 64 for providing accurate timing for operation of the pilot signal transmitter 50. The synchronization source 66 can be, for example, a Global Positioning System (GPS) receiver, a LORAN receiver or a rubidium oscillator. The pilot signal transmitter 50 also includes a transmitter/low power amplifier stage 67 for taking the pilot signal generated by the pilot signal line card 63 from its baseband frequency to a radio frequency at the desired power level. A filter circuit 68 eliminates any unnecessary noise signals and limits out-of-band emissions. A transmit antenna 69 is coupled to the filter circuit 68 for broadcasting the pilot signal generated by the pilot signal transmitter 50.

The pilot signal transmitter 51 can have, for example, a similar construction to the pilot signal transmitter 50. The CDMA pilot signals generated by the pilot signal transmitters 50, 51 are similar except for the phase offset used to identify the location of each pilot signal transmitter.

As shown in FIG. 1, the base stations 11, 12 are coupled to a Mobile Telephone Switching Office (MTSO) 30 via a dedicated telephone line, cable, optical fiber or microwave

link 25. The MTSO 30 typically provides system control to the base stations 11, 12. The MTSO 30 also controls the routing of telephone calls from a public switched telephone network (PSTN) to the appropriate base stations 11, 12 for transmission to a mobile station 40.

A mobile station 40 is located in a vehicle 45 that is currently in a digital cell 22 and moving towards an analog cell 21. Thus, the mobile station 40 is transitioning from the digital portion 1A of the cellular system to the analog portion 1B of the system. The mobile station 40 constantly monitors the pilot signals transmitted by each pilot signal transmitter 50, 51, both during idle time, i.e. no telephone call being processed, and busy times.

While the mobile station 40 is located in the digital portion 1A of the cellular system 1, the mobile station 40 receives a digitally modulated telephone call routed from the MTSO 30 through a digital base station 12 to the mobile station 40. The signal transmitted to the mobile station 40 can be, for example, a CDMA signal. As the mobile station 40 moves across the boundary from its current digital cell 22 to a new candidate digital cell 22, for example from cell D₂ to D₁, the mobile station 40 moves closer to the candidate digital cell D₁ base station 12. Accordingly, a handoff is necessary from the current base station in cell D₂ to the candidate base station in cell D₁ to maintain the established connection for the telephone call.

The handoff of the mobile station 40 from its current digital cell D₂ to the candidate digital cell D₁ involves a mobile-assisted handoff (MAHO) wherein the mobile station 40 receives the pilot signal transmitted by each pilot signal transmitter 51 located in each digital cell 22 which is sufficiently strong to reach the mobile station 40. The pilot signal transmitter 51 can be incorporated into the base station 12 equipment or it can be a stand-alone device. The mobile station 40 measures the phase offset and power level (signal strength) of each received pilot signal and transmits the measurements to the MTSO 30, via its current base station 12, so that the MTSO 30 can direct a handoff. The handoff is typically a soft handoff wherein the mobile station 40 communicates simultaneously with two or more base stations 12. Thus, the mobile station 40 transmits only one signal which may be received by multiple base stations 12, but signals to the mobile station 40 can be transmitted from multiple base stations 12. Based on the signal strength measurements made by the mobile station 40 and transmitted to the MTSO 30, the mobile station 40 establishes and/or terminates communications with base stations 12 upon direction from the MTSO 30, thus completing the soft-handoff.

As the mobile station 40 continues to move across the boundary from a digital cell 22 to an analog cell 21, for example, from cell D₁ to cell A₁₁, a handoff is necessary from the CDMA digital portion 1A of the cellular system to the analog portion 1B of the system in order to maintain the established connection for the telephone call involving the mobile station 40. Unlike the transition of the mobile station 40 from one digital cell 22 to another digital cell 22, when the mobile station 40 is crossing the boundary from a CDMA digital cell 22 to an analog cell 21, a soft handoff is not possible because the mobile station 40 cannot simultaneously communicate via a CDMA channel with a base station 12 and also via an analog channel with a base station 11. Thus, the MTSO 30 must provide a handoff message to the mobile station 40 for handoff to the analog base station 11.

The message format for directing a handoff of a mobile station between CDMA cells and also from a CDMA cell to

an analog cell is provided by the industry standard specification IS-95, for example. To facilitate CDMA handoffs, IS-95 classifies the pilot signal offsets of the CDMA pilot signals associated with each base station 12 into one of three sets: the ACTIVE SET; the CANDIDATE SET; and the REMAINING SET.

The ACTIVE SET identifies the pilot signal offsets of the pilot signals associated with base stations 12 through which the mobile station 40 is to communicate. The CANDIDATE SET identifies the pilot signal offsets of the pilot signals associated with the base station 12 with which communication is likely or for which pilot signals have been received at the mobile station 40 with sufficient signal strength to be placed in the ACTIVE SET, but have not yet been placed in the ACTIVE SET. The REMAINING SET identifies the pilot signal offsets of the pilot signals associated with the remaining base station 12 in the cellular system 1, excluding those pilot signal offsets currently in the ACTIVE and CANDIDATE sets. By providing the mobile station 40 with the pilot signal offset information, the mobile station 40 knows during which time period it should be receiving a pilot signal.

The MTSO 30 provides the mobile station 40 with an initial ACTIVE SET list of at least one pilot signal offset of a pilot signal from a pilot signal transmitter 51 associated with a base station 12 with which the mobile station is to communicate. The MTSO 30 also provides the mobile station 40 with a CANDIDATE SET list of pilot signal offsets corresponding to base stations 12 with which communication is likely. For example, base stations which are in a geographic area near the mobile station 40 are placed in the CANDIDATE SET.

According to the present invention, a CDMA pilot signal transmitter 50 is placed in each analog cell 21, thus associating a CDMA pilot signal with each analog base station 11. The MTSO 30 can identify the phase offset of the pilot signals broadcast from the analog cells 21 received by the mobile station 40 and transmitted to the MTSO 30, and thus associate the pilot signal with the analog base station 11. Therefore, the phase offsets of the pilot signals transmitted from the analog cells 21 can be incorporated into the IS-95 classifications. Pilot signal offsets associated with the pilot signals broadcast by pilot signal transmitters 50 located in the analog cells 21, however, cannot be placed in the ACTIVE SET until the mobile station 40 is actually handed off to the analog base station 11 and communications with any base stations 12 have been terminated because a mobile station 40 cannot simultaneously communicate with digital base station 12 and an analog base station 11.

The cellular system 1 according to the present invention thus includes a CDMA pilot signal transmitter 50 in each analog cell 21 for facilitating handoff of mobile station 40 from a CDMA cell 22 directly to an analog cell 21. Accordingly, the handoff method according to the present invention minimizes the additional equipment necessary for providing CDMA to analog handoffs and the possibility of lost calls.

FIG. 3 is an illustrative flowchart of the process for handoff of the mobile station 40 as the mobile station 40 crosses the boundary from the digital cell 22 to the analog cell 21 according to the present invention.

In step S0, the mobile station 40 is processing a CDMA modulated telephone call between the mobile station 40 and a calling party via the digital portion 1B of the cellular system 1. In step S1, the mobile station 40 receives a pilot signal transmitted from a base station 11, 12 in the cellular

system 1. The mobile station 40 measures the phase offset and the power level of the pilot signal and transmits the offset and signal strength measurements to the MTSO 30 via the base stations 12 with which the mobile station is in communication.

The mobile station 40 continuously scans for pilot signals and can perform, for example, continuous measurements of the pilot signals transmitted by pilot signal transmitters 50, 51. In step S2, the MTSO 30 determines whether the signal strength of the pilot signal measured by the mobile station 40 exceeds a first predetermined threshold value, $Threshold_1$, and therefore is eligible to be added to the ACTIVE SET or CANDIDATE SET. If the signal strength of the pilot signal measured by the mobile station 40 is less than $Threshold_1$, the pilot signal offset is not eligible to be added to the ACTIVE SET.

If the signal strength of the pilot signal from a base station 12 with which the mobile station 40 is in communication drops below a second predetermined threshold value, T_{drop} , then the pilot signal is dropped from the ACTIVE SET. In step S3, the MTSO 30 can direct termination of communications between the mobile station 40 and the base station 12 corresponding to the pilot signal having a signal strength measurement below T_{drop} . The mobile station 40 then returns to step S1 and continues measuring the offset and signal strength of received pilot signals and forwarding the measurements to the MTSO 30 via the base stations 12.

If the signal strength of a pilot signal received by the mobile station 40 exceeds $Threshold_1$, in step S4 the MTSO 30 determines whether the pilot signal is associated with a digital cell 22 or an analog cell 21. Based on the phase offset measurement made by the mobile station 40, the MTSO 30 can identify the pilot signal and its associated base station 11, 12, (i.e., a CDMA base station 12 or an analog base station 11). If the pilot signal is associated with a CDMA digital cell 22, in step S5 the pilot signal offset is denoted by the MTSO 30 as eligible for inclusion in the ACTIVE SET or CANDIDATE SET stored in the mobile station 40. Depending on the availability of the base station 12 of the CDMA cell 22, the MTSO 30 directs the mobile station 40 to establish communication with the new base station 12 via, for example, a soft handoff. The mobile station 40 then continues monitoring for pilot signals in step S1.

If the pilot signal has a signal strength which exceeds $Threshold_1$ and is associated with an analog cell 21, the pilot signal offset can be included in the CANDIDATE SET. In step S6 the MTSO 30 determines whether the signal strength of any of the pilot signals in the ACTIVE SET exceed the second predetermined threshold value, T_{drop} . If any of the pilot signals in the ACTIVE SET exceed T_{drop} , then the mobile station 40 remains on the digital portion 1B of the cellular system and in step S7 the mobile station 40 returns to step S1 to continue monitoring the pilot signals.

If, however, no pilot signals in the ACTIVE SET exceed T_{drop} , in step S8 the MTSO 30 sends a handoff message to the mobile station 40, via at least one of the base stations 12, directing the mobile station 40 to terminate communications with base stations 12 and to tune to an analog frequency of the analog cell 21. The MTSO 30 also directs the analog cell 21 base station 11 to tune a receiver to the same frequency provided to the mobile station 40 and the MTSO 30 reroutes the telephone call through the analog base station 11. Once the handoff of the mobile station 40 from the digital base station 12 to the analog base station 11 is completed, call processing is handled in a manner similar to call processing in conventional analog cellular systems.

11

Although the present invention has been described with respect to handoff of a mobile station from a CDMA cell to an analog cell, the principles of the present invention also can be used for handoff of a mobile station from a CDMA cell to another wireless communication system or portion thereof having a different frequency band or modulation technique than the CDMA cell with which the mobile station is currently in communication. For example, the method according to the present invention can be used for handoff of a mobile station from a CDMA cellular telecommunications system to a personal communications system (PCS) or to a digital TDMA cellular telecommunications system.

What is claimed is:

1. A method for handoff of a mobile station from a code division multiple access (CDMA) portion of a cellular telecommunication system to a non-CDMA portion of the cellular telecommunication system, the method comprising the steps of:

receiving, in the mobile station, a first CDMA pilot signal compatible with the CDMA portion of the cellular telecommunication system, the first CDMA pilot signal being transmitted from a CDMA transmitter in the non-CDMA portion of the cellular telecommunication system; and

directing a handoff of the mobile station from the CDMA portion of the cellular telecommunication system to the non-CDMA portion of the cellular telecommunication system as a function of the first CDMA pilot signal received by the mobile station.

2. The method according to claim 1,

wherein the receiving step includes

detecting a phase offset value of the first CDMA pilot signal and a signal strength value of the first CDMA pilot signal, and

transmitting the first CDMA pilot signal detected phase offset and signal strength values to a controller of the cellular telecommunication system, and

wherein the directing step includes

determining, in the controller, whether to handoff the mobile station from the CDMA portion of the cellular telecommunication system to the non-CDMA portion of the cellular telecommunication system as a function of the first CDMA pilot signal detected phase offset and signal strength values.

3. The method according to claim 2, further comprising the step of:

receiving, in the mobile station, a second CDMA pilot signal compatible with the CDMA portion of the cellular telecommunication system and transmitted from the CDMA portion of the cellular telecommunication system; and

wherein the step of receiving the second CDMA pilot signal includes

detecting a phase offset value of the second CDMA pilot signal and a signal strength value of the second CDMA pilot signal, and

transmitting the second CDMA pilot signal detected phase offset and signal strength values to the controller of the cellular telecommunication system.

4. The method according to claim 3,

wherein the directing step further includes

determining whether the signal strength of the first CDMA pilot signal exceeds a first threshold value, determining whether the first CDMA pilot signal is associated with one of a CDMA base station and a non-CDMA base station,

12

determining whether the second CDMA pilot signal is associated with one of the CDMA base station and the non-CDMA base station, and

if the second CDMA pilot signal is associated with the CDMA base station, determining whether the signal strength of the second CDMA pilot signal exceeds a second threshold value.

5. The method according to claim 4,

wherein the directing step further includes

if the first CDMA pilot signal is associated with the non-CDMA base station and the signal strength of the second CDMA pilot signal exceeds the second threshold value, maintaining communication of the mobile station on the CDMA portion of the cellular telecommunication system, and

if the first CDMA pilot signal is associated with the non-CDMA base station and the signal strength of the second CDMA pilot signal is at least as small as the second threshold value, directing handoff of the mobile station to the non-CDMA portion of the cellular telecommunication system.

6. The method according to claim 1, wherein the non-CDMA portion of the cellular telecommunications network includes an analog cellular telecommunication system.

7. The method according to claim 1, wherein the non-CDMA portion of the cellular telecommunications network includes a personal communications system (PCS).

8. The method according to claim 1, wherein the non-CDMA portion of the cellular telecommunications network includes a time-division multiple access cellular telecommunication system.

9. A cellular telecommunication system providing cellular service to a mobile station, the cellular telecommunication system having a code division multiple access (CDMA) portion and a non-CDMA portion, comprising:

a CDMA base station in the CDMA portion of the cellular telecommunication system;

a first CDMA pilot signal transmitter associated with the CDMA base station for transmitting a unique first CDMA pilot signal, the first CDMA pilot signal transmitter being in the CDMA portion of the cellular telecommunication system;

a non-CDMA base station in the non-CDMA portion of the cellular telecommunication system;

a second CDMA pilot signal transmitter associated with the non-CDMA base station for transmitting a unique second CDMA pilot signal, the second CDMA pilot signal transmitter being in the non-CDMA portion of the cellular telecommunication system; and

a controller in communication with the CDMA base station and the non-CDMA base station, wherein the controller provides for a handoff of the mobile station from the CDMA portion of the cellular telecommunication system to the non-CDMA portion of the cellular telecommunication system as a function of the unique first CDMA pilot signal and the unique second CDMA pilot signal.

10. The cellular telecommunication system according to claim 9, wherein the non-CDMA portion includes an analog cellular telecommunication system.

11. The cellular telecommunication system according to claim 9, wherein the non-CDMA portion of the cellular telecommunications network includes a personal communications system (PCS).

12. The cellular telecommunication system according to claim 9, wherein the non-CDMA portion of the cellular

13

telecommunications network includes a time-division multiple access cellular telecommunication system.

13. The method according to claim **1**, wherein the non-CDMA portion of the cellular telecommunication system

14

includes a CDMA pilot signal transmitter for transmitting the first CDMA pilot signal from the non-CDMA portion.

* * * * *



222 Delaware Avenue • Suite 900
P.O. Box 25130 • Wilmington, DE 19899
Zip Code For Deliveries 19801

Writer's Direct Access:
(302) 429-4232
sbrauerman@bayardlaw.com

June 26, 2017

VIA CM/ECF & HAND DELIVERY

The Honorable Richard G. Andrews
United States District Court for the District of Delaware
844 North King Street
Wilmington, Delaware 19801

Re: *IPA Technologies, Inc. v. Alco Electronics, Ltd.*, C.A. No. 16-1169-RGA,
IPA Technologies, Inc. v. Dish Network Corp., et al., C.A. No. 16-1170-RGA,
IPA Techs., Inc. v. TCL Comm'cn Tech. Holdings, Ltd., et al., C.A. No. 16-1236-RGA,
IPA Technologies, Inc. v. Amazon.com, Inc., et al., C.A. No. 16-1266-RGA,
IPA Technologies, Inc. v. Sony Electronics, Inc., et al., C.A. No. 17-55-RGA,
IPA Technologies, Inc. v. LG Electronics Inc., et al., C.A. No. 17-121-RGA,
IPA Technologies, Inc. v. Lenovo Group, Ltd., et al., C.A. No. 17-235-RGA,
IPA Technologies, Inc. v. Huawei Technologies Co., Ltd., et al., C.A. No. 17-248-RGA,
IPA Technologies, Inc. v. Kyocera International, Inc., C.A. No. 17-263-RGA
IPA Technologies, Inc. v. nVidia Corp., C.A. No. 17-287-RGA

Dear Judge Andrews,

Plaintiff IPA Technologies, Inc. ("IPA") provides the following proposed claim constructions for certain terms in the claims of the asserted patents and explanations of how they illustrate patent-eligibility, pursuant to the Court's Order. (C.A. No. 16-1266-RGA, D.I. 21.) In the context of motions to dismiss, the Court should adopt non-movant IPA's proposals as correct for purposes of the pending Section 101 motions.¹

¹ IPA provides these claim construction proposals solely for purposes of consideration of the pending Section 101 motions at the pleadings stage. Discovery has not yet begun, and Defendants have not produced any documents concerning the accused instrumentalities. The parties have not exchanged contentions, identified claim terms for construction, or provided proposed claim constructions. IPA reserves its rights to modify or withdraw any of the claim construction proposals herein, if necessary, in the claim construction exchange process later in the case.



Claim Term	Proposed Construction
navigation query	an electronic query, form, series of menu selections, or the like; being structured appropriately so as to navigate a particular data source of interest in search of desired information
electronic data source	source of information in numerical form that can be digitally transmitted or processed and that is implemented on or by means of a computing device
rendering an interpretation of the spoken request	determining a meaning of the spoken request using a computing device, such as that provided by extracting speech data from acoustic voice signals or data and linguistically parsing the speech data
constructing a navigation query based upon the interpretation / constructing at least part of a navigation query based upon the interpretation	combining or arranging elements of (at least part of) the navigation query based upon the interpretation

IPA's proposed constructions for these terms, which appear in the claims of all three asserted patents, demonstrate how the claimed inventions are directed to patent-eligible technological solutions or improvements specific to navigating electronic data sources, rather than an abstract idea of using speech to obtain any kind of information. *See Alice Corp. Pty. Ltd. v. CLS Bank Int'l*, 134 S. Ct. 2347, 2358 (2010) (claims that "solve a technological problem" or "improve[] an existing technological process" are eligible under Section 101); *DDR Holdings, LLC v. Hotels.com, L.P.*, 773 F.3d 1245 (Fed. Cir. 2014) (claims are patent-eligible where "the claimed solution is necessarily rooted in computer technology in order to overcome a problem specifically arising in the realm of computer networks").

The construction of "navigation query" flows from the express definition of the term in the specification, (*see* '021 Patent at 8:55-62), and places the claimed invention firmly in the realm of **electronic** navigation of data sources. IPA previously proposed the construction for "navigation query" and discussed how it grounds the claimed technological solutions to technological problems specific to existing computing-based systems. (C.A. No. 16-1266-RGA, D.I. 15 at 7-8.) The term "electronic data source" underscores the focus of the claims on specific technological solutions, as "data" is unpacked based on its plain meaning as information in numerical form that can be digitally transmitted or processed, and "electronic" is further unpacked as implemented on or by means of a computing device. The construction encompasses the range of electronic data sources discussed in the specification, including "database(s), Internet/web site(s), ... multimedia content, such as movies or other digital video and audio content, other various forms of entertainment data, or other electronic information." ('021 Patent at 4:11-20.)

The specification addresses the claim phrase "rendering an interpretation of the spoken request" in a passage stating that "[w]hen a spoken input request is received from a user, it is interpreted, such as by using a speech recognition engine to extract speech data from acoustic voice signals, and using a language parser to linguistically parse the speech data." ('021 Patent at 2:30-34.) The specification's description of using a speech recognition engine—a software solution for



extracting speech data from acoustic voice signals—is incorporated into the construction, as is the use of a language parsing software solution. The focus on data is a highlight of the specification passage and further grounds the claimed inventions as technological solutions. The specification goes on to explain, as reflected in the construction, that “[t]he interpretation of the spoken request can be performed on a computing device locally with the user or remotely from the user.” (*Id.* at 2:34-36.) The construction of the claim phrase “constructing a navigation query based upon the interpretation” also reflects the software outputs of rendering an interpretation, because the navigation query is constructed by combining or arranging elements of the navigation query based on outputs of the software that interprets the spoken request. Both of these constructions show how the claimed inventions address the technological problem of speech-based navigation of complex and heterogeneous electronic data sources: software can extract and parse the speech data, and then can construct navigation queries that meaningfully connect the interpreted spoken request to electronic repositories of digital information.

In sum, IPA’s proposed constructions demonstrate for all the asserted patents that the claimed inventions are directed to specific technological solutions or improvements in the context of computing devices. Courts have rejected patent-eligibility challenges to such claimed solutions rooted in computing technologies. *Enfish, LLC v. Microsoft Corp.*, 822 F.3d 1327, 1337-38 (Fed. Cir. 2016); *DDR Holdings*, 773 F.3d at 1255-59; *see also* cases cited at Case No. 16-1266-RGA, D.I. 15 at 11-12 & n.6. This Court should similarly reject the patent-eligibility challenges here.

Respectfully submitted,

/s/ Stephen B. Brauerman

Stephen B. Brauerman (sb0922)

cc: All counsel of record

Building Brains for Rooms: Designing Distributed Software Agents

Michael H. Coen

MIT AI Lab
545 Technology Square
Cambridge, MA 02139
mhcoen@ai.mit.edu

Abstract

This paper argues that complex, embedded software agent systems are best constructed with parallel, layered architectures. These systems resemble Minskian Societies of Mind and Brooksian subsumption controllers for robots, and they demonstrate that complex behaviors can be had via the aggregates of relatively simple interacting agents. We illustrate this principle with a distributed software agent system that controls the behavior of our laboratory's Intelligent Room.

Introduction

This paper argues that software agent systems that interact with dynamic and complex worlds are best constructed with parallel, layered architectures. We draw on Brooks' subsumption architecture (Brooks, 1985) and Minsky's Society of Mind (Minsky, 1986) theory to dispel the notion that sophisticated and highly capable agent systems need elaborately complex and centralized control.

Towards this end, we present an implemented system of software agents that forms the backbone of our laboratory's "Intelligent Room" (Torrance, 1995). These agents, known collectively as the *Scatterbrain*, control an environment very tenuously analogous to the intelligent rooms so familiar to Star Trek viewers --- i.e., rooms that listen to you and watch what you do; rooms you can speak with, gesture to, and interact with in other complex ways.

The Scatterbrain consists of approximately 20 distinct, intercommunicating software agents that run on ten different networked workstations. These agents' primary task is to link various components of the room (e.g., tracking cameras, speech recognition systems) and to connect them to internal and external stores of information (e.g., a person locator, the World Wide Web). Although an

individual agent may in fact perform a good deal of computation, we will focus our interest on the ways in which agents get connected and share information rather than how they internally manipulate their own data. And while the Intelligent Room is a fascinating project in itself, we will treat it here mainly as a test-bed to learn more about how software agents can interact with other computational and real entities.

Our approach has also been modeled on a somewhat unorthodox combination of the Brooks (Brooks, 1991) and Minsky approaches to core AI research. As pointed out in (Kautz et al., 1994), it is difficult to find specific tasks for individual agents that are both feasible and useful given current technology. Many of the non-trivial tasks we would like software agents to perform are simply beyond the current state of the art. However, taking our cue from Minsky, we realize interesting and complex behaviors can be had via the aggregates of simpler ones; groups of simple agents can be combined to do interesting things. We also found Brooks' subsumption architecture useful for guiding the creation of the Scatterbrain, particularly for building parallel layers of behaviors that allow the room to process multiple events simultaneously and to change contexts quickly. In many ways, the room is similar to a disembodied robot, so it comes as no surprise that the robotics community can provide insight into how the room's brain should be designed. We argue, however, that this does not preclude insights obtained in creating the Scatterbrain from applying to other distributed software agents systems. Rather, as argued by Etzioni (Etzioni, 1994, 1996; Etzioni et al. 1994), even agents who solely interact with the online world (and don't have cameras for eyes and microphones for ears) can be viewed as a kind of simulated, virtual robot. More important than its connection with the real world, what the Scatterbrain shares with Brooks' robots is its organizational structure and its lack of central processing; all of the Scatterbrain's agents work together in parallel with different inputs and data being processed simultaneously in different places.

The next section of this paper describes the Intelligent Room's physical infrastructure. After this, we introduce the room's most recent application, a *Tour Guide* agent that helps a person present our lab's research to visitors.

¹Copyright © 1997, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

²This material is based upon work supported by the Advanced Research Projects Agency of the Department of Defense under contract number F30602-94-C-0204, monitored through Rome Laboratory and Griffiss Air Force Base.

Next, we present in detail the room's software agent architecture, including the design and implementation of several components of the Scatterbrain and Tour Guide agents. We also contrast our approach with several earlier monolithic efforts taken in our lab to program and control the behavior of the Intelligent Room.

Part of the motivation for this work has been to push the envelope of software agent design. Much has been made over the lack of obvious "killer applications" for software agents. After all, how many automated meeting schedulers does the world need? We are interested in exploring new realms of complex interactions for software agents which in and of themselves constitute these "killer apps" that have been seemingly so elusive from the single-agent perspective. Minsky argues that societies of agents can do things that seem inordinately complex when their behavior is viewed as the work of a single entity. Our experiments with fairly large assemblies of software agents mark an early attempt towards establishing that this is indeed the case.

The Intelligent Room

The Intelligent Room project explores new forms of interaction and collaboration between people and computers. Our objective is to create a new kind of environment capable of interpreting and contributing to activity within it. On a grand scale, we are examining a paradigmatic shift in what it means to use a computer. Rather than view a computer as a box with a keyboard and monitor used for traditional computational tasks, we envision computers being embedded in the environment and assisting with ordinary, traditionally non-computational activity. For example, if I lose my keys in the Intelligent Room, I'd someday simply like to ask the room where I left them.

The Intelligent Room is an excellent environment in which to conduct core AI research according to the criteria of both Brooks (Brooks, 1991) and Etzioni (Etzioni, 1994). The room is "physically" grounded in the real-world. The room's cameras and microphones allow it to deal with the kinds of complex, unpredictable and genuine phenomena that Brooks argues is essential for a core AI research testbed. However, the room also processes abstract, symbolic information that actually represents something extant, thereby satisfying Etzioni's desiderata. For example, if a person asks the room, *What is the weather in Boston?*, the room needs to recognize more than a meaningless *weather* token - it needs to get that information and display it to the user. This is done using a variety of information retrieval systems available on the World Wide Web.

This section first describes the room's physical infrastructure. We then present the room's most recent application, a *Tour Guide* agent that helps a person present our lab's research to visitors. In the next section, we discuss in detail the room's software agent architecture, including the design and implementation of the

Scatterbrain and Tour Guide agents.

Infrastructure - From the bottom up

Figure 1 diagrams the room's physical layout. The Intelligent Room's infrastructure consists of several hardware and software systems that allow the room to observe and control its environment in real-time. The positions of people in the room are determined using a multi-person tracking system. Hand pointing actions are recognized by two separate gesture recognition systems. The one used in the application described below allows the room to determine where someone is pointing on either of two images projected on a room wall from high-intensity, ceiling mounted VGA projectors. A speech recognition system developed by (Zue, 1994) allows the room to listen to its inhabitants, and it is used in conjunction with a speech generator to enable the room to engage in sustained dialogues with people. The room interfaces with the START natural-language information retrieval system (Katz, 1990) to enhance its ability to understand complex linguistic input. The room also controls two VCRs and several other video displays in addition to the ceiling mounted projectors. A matrix switcher allows arbitrary connections between the room's audio/visual inputs and outputs.

The room's hardware systems are directly interfaced with low-level C programs to insure their real-time operation. For example, the room's tracking cameras have 30 Hz frame rates and their data streams need to be synchronously processed using direct operating system calls.

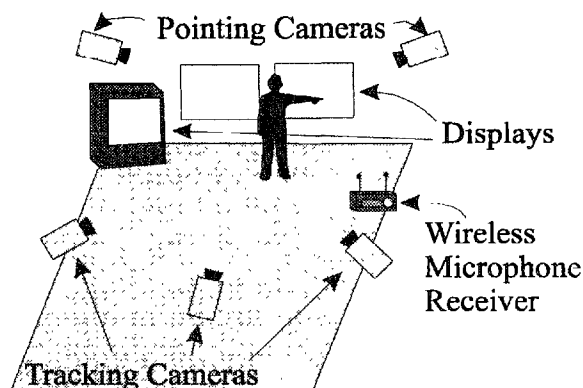


Figure 1 - Intelligent Room Floor Plan

The Tour Guide Agent

The room's most recent application provides support for someone giving tours of our laboratory. These tours typically involve a group of visitors meeting with a graduate student who discusses and answers questions about the lab's research and shows several video clips. Rather than have these presentations given in an ordinary

conference room, we have decided to have them in the Intelligent Room so the room can assist the human tour guide. A typical dialogue between the room and student tour guide is:

Tour guide: *Computer, load the AI Lab tour.*

Room: *I am loading the AI Lab tour.* Right projector now displays a Netscape browser window with a special Lab Tour home page.

Tour guide: Using hand, points at link to a research project displayed on the wall and says, *Computer, follow this link*

Room: Loads the indicated page into the browser.

Tour guide: *Computer, show me the Intelligent Room home page.*

Room: Loads the URL corresponding to the name of the page. Then says, *I have a video clip for this research. Would you like to see it?*

Tour guide: *Computer, yes.*

Room: Moves appropriate video cassetetape to correct position and starts the clip playing on left projector.

Tour guide: (watches video for a few seconds) *Computer, stop the video. Computer, play the Virtual Surgery clip.*

Room: Performs requested action. Stops video when clip is done.

Tour guide: *Computer, how many graduate students are there at the AI Lab?*

Room: *I am asking the START system for the answer...The Laboratory's 178 members include 17 faculty members, 26 academic staff, 29 research and support staff, and 106 graduate students.* Also displays web page with elaborated answer.

Other applications include a control center for planning hurricane disaster relief and an intelligent living room.

Control Architectures

The room has been discussed so far at its most concrete and most abstract: namely, its hardware infrastructure and its high-level software applications. How these applications are actually created on top of this infrastructure, i.e., how the room actually works, is the subject of this section.

Monolithic Control

In its early stages of development, each of the room's components was wrapped inside a TCP client or server that connected with a monolithic C-language program that controlled the room's behavior. Figure 2 contains this controller along with each of the programs it connected with. (Included in parentheses with each component is the name of the workstation it ran on.)

From a conceptual point of view, the most serious flaw with the centralized controller was that it failed to distinguish between basic functioning common to all room contexts –such as noticing when someone came through the doorway – and unique activities associated with a particular room application. Furthermore, adding new functionality to the room required modifying the monolithic controller and manually determining the interactions and conflicts between old and new room

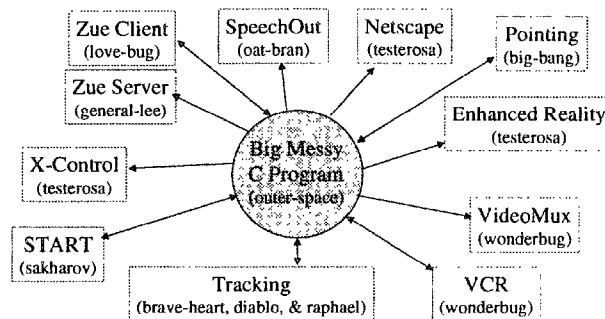


Figure 2 - The Monolithic Controller

functions. There was no way to modularly add new room capabilities on top of old ones and assume everything would continue working as expected.

Also, directing the information flow among the room's various components –one of the main functions of the controller – was overly difficult in a language like C. We needed higher-level mechanisms for describing how room information moved among its producers and consumers.

From a practical point of view, the monolithic controller also made it difficult to reconfigure the room dynamically or restart pieces of the room independently of others. We often found while working on the room that in order to restart one component, it was necessary to restart the entire room. This was particularly frustrating because starting the room required the coordinated activity of several people to start particular programs (in a predetermined order) and configure various room hardware. It was also difficult to move components of the room to different workstations because that required modifying hard-coded machine dependencies in the code.

SodaBot

Although we managed to use the monolithic approach for several very simple applications, it seemed unlikely to scale to the more complex interactions we had in mind for the room. Our initial dissatisfaction with this architecture led to the adoption of the SodaBot software agent platform (Coen, 1994) for duplicating the functionality of our initial monolithic room controller with a system of distributed software agents.

SodaBot provides both a programming language and runtime platform for software agents, and it simplifies writing interactive programs by providing high-level primitives for directing flows of online information. For example, it provides mechanisms for writing agent-wrappers that interface with preexisting software either via text-based or graphical user interfaces (X-windows and Windows 95/NT).

For example, we created a SodaBot *Netscape Agent* that controls interactions with a Netscape browser. It offers functions to other agents such as those listed below.

Function	Purpose
New (host)	Runs a new browser on given host
Load(url)	Loads URL in browser
Page_watch()	Arranges for notification (of URL) to another agent whenever browser loads new page
Link_watch()	Arranges for notification to another agent when a new page is loaded containing its URL/anchor text pairs
Text	Returns text of current page
Page(direction)	Moves browser scroll-bar in given direction

For the Intelligent Room, we use SodaBot agents as *computational glue* for interconnecting all of the room's components and moving information among them. Initially, we simply duplicated the room's monolithic controller using SodaBot's high-level programming language. Most notably, SodaBot simplified description of room functioning and interaction with remote TCP-based clients and servers by removing networking and hardware details. However, this new room controller, dubbed the *Brain*, was still a computational bottleneck, and we had yet to distinguish between a general behavioral infrastructure for the room (i.e. its core functionality) and the more complex, application specific interactions we built on top of it. This led to the development of the room's current control system, the Scatterbrain, which is the subject of the next section.

Distributed Room Control

The Scatterbrain (Figure 3) is platform on top of which room applications can be layered. In the figure, each circle represents a distinct SodaBot software agent that is wrapped around and interfaced with an external application. (The layer containing these "base applications" is not shown.) Each of the Scatterbrain agents is responsible for a different room function. For example, the *SpeechIn Agent*, runs and interfaces with our speech recognition systems. Once started, *SpeechIn* allows other agents to submit context-free grammars corresponding to spoken utterances they are interested in. As they do, it updates the speech recognition systems to contain the current set of valid utterances. When a sentence is heard by one of the speech systems, *SpeechIn* then notifies those agents who indicated they were interested in it. As another example, the *Netscape Agent* connects to the *Display Agent* to make sure that when web pages are loaded, the browser is actually displayed somewhere in the room where people can see.

The Scatterbrain agents are distributed among 10 different workstations and rely on SodaBot interagent communication primitives to locate and communicate with each other. The lines in the figure represent default interactions the room manifests in all applications, such as having various agents connect with the speech recognition agents and making sure the tracking system notices when

someone comes in the room. Essentially, the Scatterbrain implements the Intelligent Room's reflexes.

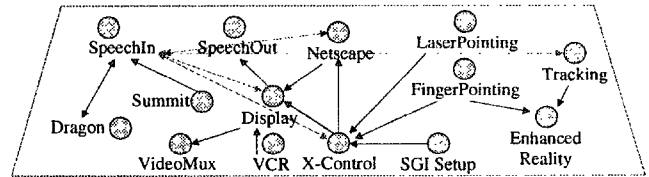


Figure 3 – The Agents of the Scatterbrain

The room no longer has a central controller. A small startup agent runs all of the Scatterbrain agents which then autonomously move to the machines on which they are supposed to run. All the Scatterbrain agents then work together in parallel with different inputs and data being processed simultaneously in different places. This makes the room robust to failure in one of its sub-systems and allows us to restart sections of the room independently. Also, the SodaBot system allows real-time data connections between agents to be broken and resumed invisibly. For example, if the *Tracking Agent* is updating another agent in real-time, either one of them can be stopped and restarted and they will be automatically reconnected.

Layered on top of the Scatterbrain, we created higher-level agents that rely on the Scatterbrain's underlying behaviors. Figure 4 contains the room's intermediate information-level applications such as a *Weather Agent* that can obtain forecasts and satellite maps for particular places. By relying on the previously described interaction, if the *Weather Agent* uses the *Netscape Agent* to display information, it doesn't need to be concerned with insuring the browser is displayed in a place where the user is looking.

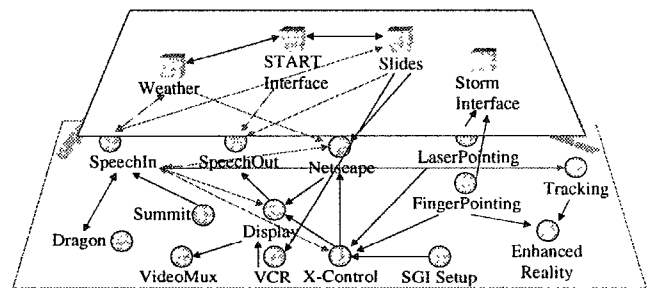


Figure 4 – Information Agents

We then created specific room application agents that relied on the lower-level, general-purpose agents in the room. Figure 5 contains a diagram of several room applications and how they connect to the room's underlying architecture. Note that all of the objects in the figure represent SodaBot software agents and many of

them connect to non-displayed external applications. The next section explores two of the application agent interactions in more detail.

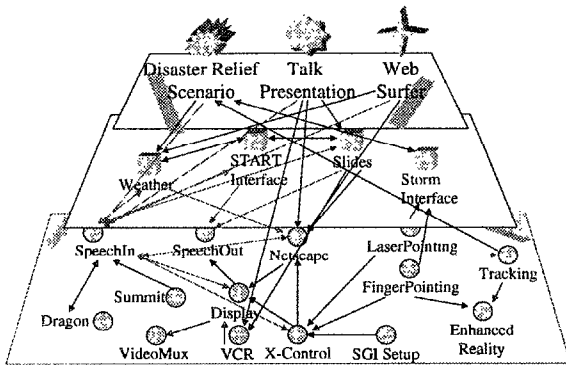


Figure 5 – Intelligent Room Software Agents

Agent Interaction

This section examines how we can get the room to exhibit interesting behavior by layering agents on top of each other. We examine two separate room behaviors and then discuss how they combine to produce greater functionality.

We have a system in the room called *Storm*, used in a disaster relief planning scenario, that can display scalable maps of the Carribean. People can interact with Storm using pointing and speech. For example,

User: *Computer, display Storm on the left projector.*

(User now points at Puerto Rico.)

User: *Computer, zoom in.*

(User now points at San Juan.)

User: *Computer, what is the weather here?*

(The room then displays a weather forecast for San Juan inside a Netscape browser on the other projector.)

To see how this scenario works, we first examine pointing recognition as an example of simple agent interaction. We then look at a more complex scenario from the Tour Guide agent presented earlier.

By default, the room's projectors are set by the *Display Agent* to show portions of the screens of two of our SGI workstations. If someone points someplace close to one of these projected displays, the display's mouse cursor moves to that position. Although this seems like a trivial process, there is a fair amount of effort behind it, as shown in Figure 6. The person moving his finger is reflected in the camera images received by the neural network pointing software. This reconciles the images to produce new pointing information. These new data are passed to the *FingerPointing Agent* which is responsible for handling all such events in the room. By default, the Scatterbrain has all pointing events on the each display sent to an agent

called the *X-Server* that controls the actual SGI workstation generating the display. This X-Server agent then moves the mouse cursor to the appropriate position, which is reflected in the displayed image. However, the Storm Agent overrides this default behavior and redirects pointing events on the Storm display to itself. Upon receipt of a pointing event, it updates the Storm application's internal cursor, which moves intelligently between salient geographical features. For example, pointing near San Juan will cause the Storm program to register the city with the Storm Agent, rather than a point three pixels to its left. Finally, note that the various agents are responsible for translating between the room's many coordinate systems, as shown along the connections.

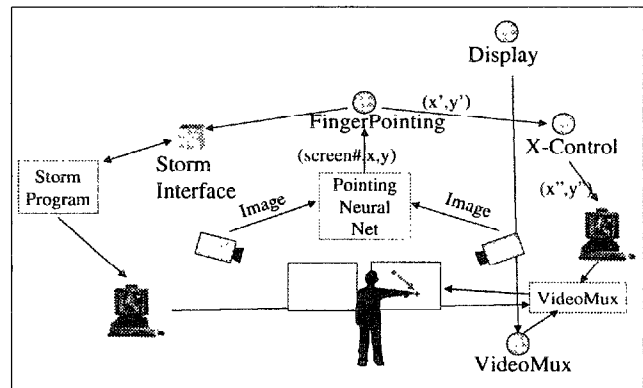


Figure 6 – Pointing in the Room

When someone in the room says *What's the weather here?*, the *SpeechIn Agent* notifies the room's *Disaster Relief Planning Agent* because this utterance is contained in the grammar that agent had registered when first run. The *Storm Agent* is then contacted to determine what geographical entity is closest to where the person was pointing close to the time they asked the question. (Low-level room events are time-stamped by agents in order to facilitate multimodal reconciliation.) This process is shown in Figure 7.

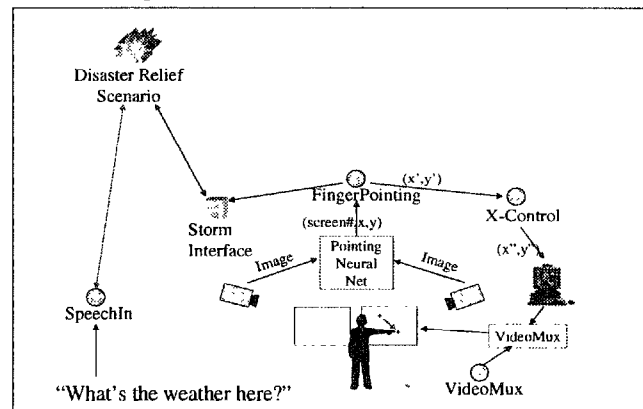


Figure 7 – Multimodal Resolution

Conclusion

Motivated by Minsky's Society of Mind and Brooks' subsumption approach to building robots, we have argued that software agent systems that interact with complex and dynamic worlds are best created from distributed collections of simple agents with parallel, layered architectures.

The complexity of the overall system comes from the interactions of these agents, even though no individual agent is in itself particularly complex and no single agent centralizes the system's control. This approach allows us to build robust, reusable, and behaviorally sophisticated systems that are capable of interacting with the ever-changing real and online worlds. To demonstrate this approach, we presented the Scatterbrain – a distributed collection of software agents that control our laboratory's Intelligent Room.

Acknowledgements

Development of the Intelligent Room has involved the efforts of many people. Professors Tomas Lozano-Perez, Lynn Stein and Rodney Brooks were principally responsible for the room's conception and construction. Mark Torrance led the project during its first year and wrote the room's earliest monolithic controllers. The room's many vision systems are due to the efforts of Jeremy De Bonet, Chris Stauffer, Sajit Rao, Tomas Lozano-Perez, Darren Phi Bang Dang, JP Mellor, Gideon Stein, and Kazuyoshi Inoue. Polly Pook contributed to the design of the room's distributed computation and has worked on modeling the room's functionality as a cognitive process. Josh Kramer wrote large sections of the Scatterbrain and participated in the development of the SodaBot system. All of the above mentioned were also responsible for designing room applications, and many of the above hacked on various room components. Kavita Thomas, along with help from Mark, Polly, and Tomas, configured Victor Zue's speech recognition system. (Jim Glass provided assistance in getting the system running.) Boris Katz and Deniz Yuret provided much support in interfacing with and customizing the START natural language system. Mike Wessler created one of the room's earliest applications and wrote invaluable graphical interfaces for much of the room's hardware.

References

Brooks R. 1985: A Robust Layered Control System for a Mobile Robot, AI Lab Memo 864, Massachusetts Institute of Technology. Cambridge, MA.

Brooks R. 1991: Intelligence without Representation, in Special Volume: Foundations of Artificial Intelligence, Artificial Intelligence, 47(1-3).

Coen, M. 1994. SodaBot: A Software Agent Environment and Construction System. AI Lab Technical Report 1493. Massachusetts Institute of Technology. Cambridge, MA.

Etzioni, O. 1994: Intelligence without Robots, AI Magazine, Winter 1994.

Etzioni, O.; Levy, H.; Segal, R.; and Thekkath, C. 1994. OS Agents: Using AI Techniques in the Operating System Environment. Technical Report 93-04-04. Dept. of Computer Science. University of Washington. Seattle, WA.

Etzioni, O. 1996: Moving Up the Information Food Chain: Deploying Softbots on the World Wide Web, in Proceedings of the Thirteenth National Conference on Artificial Intelligence, AAAI Press/MIT Press, Cambridge, MA, pp.1322-1326, 1996.

Kautz, H.; Selman, B.; Coen, M.; and Ketchpel, S. 1994. An Experiment in the Design of Software Agents. In Proceedings of the Twelfth National Conference on Artificial Intelligence, AAAI Press/MIT Press, Cambridge, MA.

Katz, B. 1990. Using English for Indexing and Retrieving. In *Artificial Intelligence at MIT: Expanding Frontiers*. Winston, P.; and Shellard, S. (editors). MIT Press, Cambridge, MA. Volume 1.

Minsky, M. 1986. *Society of Mind*. New York. Simon and Schuster.

Torrance, M. 1995. Advances in Human-Computer Interaction: The Intelligent Room, In Working Notes of the CHI 95 Research Symposium, May 6-7, 1995, Denver, Colorado.

Zue, V. 1994. Human Computer Interactions Using Language Based Technology, IEEE International Symposium on Speech, Image Processing & Neural Networks, Hong Kong.

Hing-Yan Lee Hiroshi Motoda (Eds.)

PRICAI'98: Topics in Artificial Intelligence

5th Pacific Rim International Conference
on Artificial Intelligence
Singapore, November 22-27, 1998
Proceedings



Springer

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editors

Hing-Yan Lee
Knowledge Lab., Kent Ridge Digital Labs
21 Heng Mui Keng Terrace, Singapore 119613
E-mail: hingyan@krdl.org.sg

Hiroshi Motoda
The Institute of Scientific and Industrial Research
Osaka University
8-1 Mihogaoka, Ibaraki, Osaka 57, Japan
E-mail: motoda@ar.sanken.osaka-u.ac.jp

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Topics in artificial intelligence : proceedings / PRICAI '98, 5th Pacific Rim International Conference on Artificial Intelligence, Singapore, November 22 - 27, 1998. Hing-Yan Lee ; Hiroshi Motoda (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 1998
(Lecture notes in computer science ; Vol. 1531 : Lecture notes in artificial intelligence)
ISBN 3-540-65271-X

CR Subject Classification (1998): I.2

ISBN 3-540-65271-X Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

© Springer-Verlag Berlin Heidelberg 1998
Printed in Germany

Typesetting: Camera ready by author
SPIN 10692922 06/3142 - 5 4 3 2 1 0 Printed on acid-free paper

An Adaptive Agent Oriented Software Architecture^{*}

Babak Hodjat

Christopher J. Savoie

Makoto Amamiya

Department of Intelligent Systems
Graduate School of Information Science and Electrical Engineering
Kyushu University
6-1 Kasugakoen, Kasuga-shi
Fukuoka 816, Japan
http://www_al.is.kyushu-u.ac.jp/~bobby/index.html

Abstract. A new approach to software design based on an agent-oriented architecture is presented. Unlike current research, we consider software to be designed and implemented with this methodology in mind. In this approach agents are considered adaptively communicating concurrent modules which are divided into a white box module responsible for the communications and learning, and a black box which is the independent specialized processes of the agent. A distributed Learning policy is also introduced for adaptability.

Topics: Agent Architectures, Agents Theories.

Keywords: Agent-oriented systems, Multi-Agent Software architectures, Distributed Learning,

1. Introduction

The classic view taken with respect to *Agent Oriented Systems* is to consider each agent an autonomous individual the internals of which are not known and that conforms to a certain standard of communications and/or social laws with regard to other agents [5]. Architectures viewing agents as such have had to introduce special purpose agents (e.g., broker agents, planner agents, interface agents...) to shape the structure into a unified entity desirable to the user [2]. The intelligent behavior of these key agents, with all their complexities, would be vital to the performance of the whole system.

^{*}This paper describes a new adaptive, multi-agent approach to software architecture, currently being investigated as an on going project at Kyushu University.

Another trend in this view is to give the possibility to the agents to query each other's internal knowledge and states through the communications protocols, while at the same time conserving the black-box view of the agents. Inevitably, defining and controlling such issues as conflicts of interest between agents, honesty, helpfulness, and gullibility, have had to be taken into account and dealt with [4]. The most important aspect of this dominant view is that agent architectures are considered to be unifiers of pre-written, separate modules (heterogeneity) [3]. Each of these modules was probably designed without having this higher structure in mind, and is completely different (be it in the code, the machine it is implemented upon, the designer, or the purpose of design). Agent-based Software Engineering was originally invented to facilitate the creation of software able to interpolate in such settings and application programs were written as software agents [6]. On the other hand, methodologies dealing with the internal design of agents tend to view them primarily as intelligent, decision-making beings. In these methodologies, techniques in Artificial Intelligence, Natural Language Processing, machine learning, and adaptive behavior seem to overshadow the agent's architecture, in many cases undermining the main purpose of the agent [7][13].

For instance, one can view agents as reinforcement learning agents with a set of tools to be chosen with respect to environmental senses. Such a view, however well suited for agent learning techniques, may not be readily applied to more algorithmic applications, thus misleading one to assume that such applications should not be or could not be implemented as agents.

In this paper, we wish to present an agent-oriented methodology, which can be universally applied to any software design. The Adaptive Agent Oriented Software Architecture (AAOSA) builds upon and extends the widely accepted object oriented approach to system design. The primary difference sited between Agent-oriented and Object oriented programming has been the language of the interface [6]. In this paper we will suggest an approach in which communication between agents can be done independent of language. This language independent communication will still hold as the main difference with the Object oriented methodology. Another aspect that makes agents more attractive to use in software than objects is their quality of volition. Using AI techniques, adaptive agents are able to judge their results, then modify their behavior (and thus their internal structure) to improve their perceived fitness. First we will clarify our definition of agents, which is somewhat relaxed with respect to the classic definitions. Then the steps by which software should be designed using AAOSA methodology are described. Some suggestions as to how adaptive learning and communication language independence can be achieved are briefly presented next. To clarify the AAOSA methodology, we present an example application in the form of a simple multimodal map program and show some of the resulting features.

2. Our Definition of Agents

Our definition of agents is more in line with the ones given by [12] and [2], and we classify our agents as having the following properties [5]: reactivity, autonomy, temporal continuity, communicative capabilities, team orientation, mobility, learning

(adaptive), and flexibility. The resulting multi-agent system we have in mind is a partially connected one [5]. A direct communication specification-sharing approach is taken here to enhance collaboration. Instead of using assisted coordination, in which agents rely on special system programs (facilitators) to achieve coordination [2], in our approach new agents supply other agents with information about their capabilities and needs. To have a working system from the beginning, the designers preprogram this information at startup. This approach is more efficient because it decreases the amount of communication that must take place, and does not rely on the existence, capabilities, or biases of any other program [6].

Adaptive agents are adaptively communicating, concurrent modules. The modules therefore consist of three main parts: A communications unit, a reward unit, and a specialized processing unit. The first two units we will call the white box and the third the black box parts of an agent (Figure 1). The main responsibilities of each unit follow:

The communications unit: This unit facilitates the communicative functions of the agent and has the following sub-systems:

- *Input of received communication items:* These items may be in a standard agent communication language such as KQML. Later in this paper we will see that only a small subset is needed here.
- *Interpreting the input:* Decides whether the process unit will need or be able to process certain input, or whether it should be forwarded to another agent (or agents). Note that it is possible to send one request to more than one agent, thus creating competition among agents.
- *Interpretation Policy:* (e.g., a table) Determines what should be done to the input. This policy could be improved with respect to the feedback received for each interpretation from the rewards unit. Some preset policy is always desirable to make the system functional from the beginning. In the case of a system reset, the agent will revert to the basic hard-coded startup information. The interpretation policy is therefore comprised of a preset knowledge base, and a number of learned knowledge bases acquired on a per-user basis. A *learning* module will be responsible for conflict resolutions in knowledge base entries with regard to feedback received on the process of past requests. Past requests and what was done with them are also stored until in anticipation of their feedback.
- *Address-Book:* keeps an address list of other agents known to be useful to this agent, or to agents known to be able to process input that can not be processed by this agent. Requests to other agents may occur when:
 - The agent has received a request it does not know how to handle,
 - The agent has processed a request and a number of new requests have been generated as a result.

This implies that every agent have an address and there be a special name server unit present in every system to provide agents with their unique addresses (so that new agents can be introduced to the system at run time). This address list should be dynamic, and therefore adaptive. It may be limited; it may also contain

information on agents that normally send their requests to this agent. In many cases the Address-book could be taken as an extension of the Interpretation Policy and therefore implemented as a single module.

- *Output:* Responsible for sending requests or outputs to appropriate agents, using the Address-book. A confidence factor could be added to the output based on the interpretations made to resolve the input request or to redirect it. We shall see later in the paper that this could be used when choosing from suggestions made by competing agents by output agents.

The rewards unit: Two kinds of rewards are processed by this module: outgoing and incoming. An agent is responsible for distributing and propagating rewards being fed back to it*. This unit will determine what portion of the incoming reward it deserves and how much should be propagated to requesting agents. The interpreter will update its interpretation policy using this feedback. The rewards will also serve as feedback to the Address-book unit, helping it adapt to the needs and specifications of other agents. The process unit could also make use of this feedback.

The rewards may not be the direct quantification of user states and in most cases will be interpretations of user actions made by an agent responsible for that. We will further clarify this point later in the paper.

The process unit: This unit is considered a black box by our methodology. The designer can use whatever method it deems more suitable to implement the processes unique to the requirements of this agent. The only constraints being that the process unit is limited to the facilities provided by the communications unit for its communications with other agents. The process unit also may use the rewards unit to adapt its behavior with regard to the system. Note that each agent may well have interactions outside of the agent community. Agents responsible for user I/O are an example of such interactions. These agents generally generate requests or initiate reward propagation in the community, or simply output results.

The white box module can easily be added to each program module as a *transducer*. According to definition [6] the transducer mediates between the existing program (the process unit) and other agents. The advantage of using a transducer is that it requires no knowledge of the program other than its communication behavior.

We mentioned the process unit as being able to conduct non-agent I/O. It is easy to consider I/O recipients (e.g. files or humans) as agents and make the program redirect its non-agent I/O through its transducer. Other approaches to agentification (wrapper and rewriting) are discussed in [6].

* A special purpose agent is responsible for the interpretation of user input as feedback to individual user requests. This agent will then initiate the reward propagation process.

3. Software design

The software as a whole should be thought of as a society, striving to accomplish a set of requests. The input requests are therefore propagated, processed by agent modules that may in turn create requests to other agents. Again, it is up to the designers to break down the system, as they feel suitable. Hierarchies of agents are possible and agents can be designed to be responsible for the minutest processes in the system. It is advisable that each agent be kept simple in its responsibilities and be limited in the decisions it needs to make to enhance its learning abilities. The overhead of the required units (the white box) should be taken into consideration.

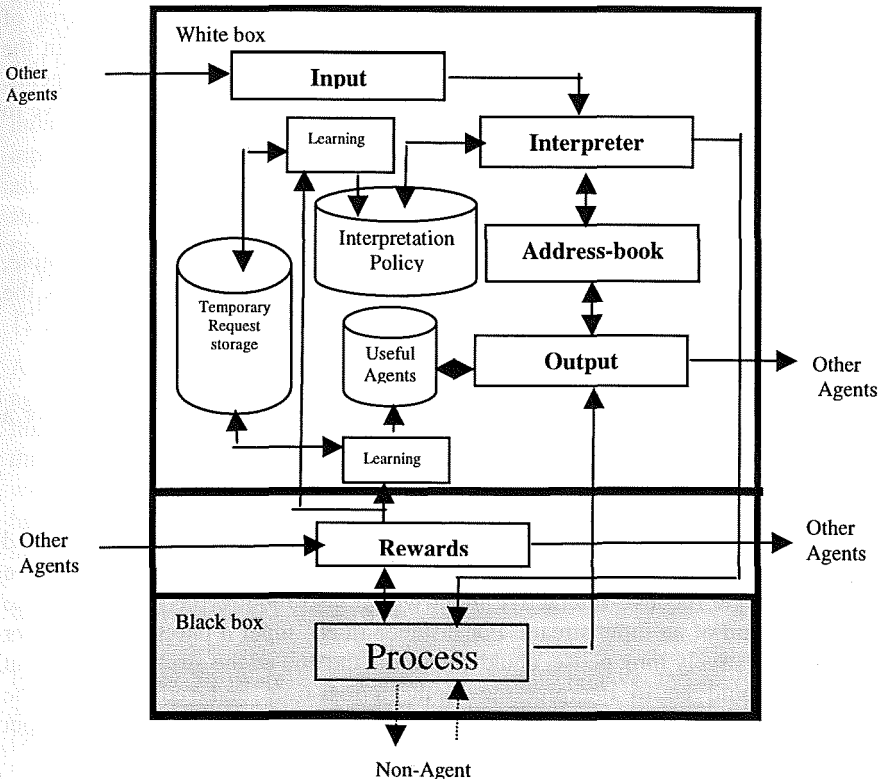


Fig. 1. Each agent is comprised of a black box section (specialties) and a white box section (communications).

Agents can be replaced at run-time with other more complete agents. The replacement can even be a hierarchy or network of new agents breaking down the responsibilities of their predecessor. This feature provides for the incremental design and evaluation of software.

We recommend each agent's responsibilities and actions to be clearly defined at design time. As stated in the previous section, many aspects of the white box units

should also be preset for each agent according to its definition. To have a working system from the beginning, it is also necessary to define the preliminary communication links between the agents at design time. It should be noted that these communications might change through time, for instance in the case of the introduction of newer agents. Thus, the most important phase in the design of software with this methodology will be to determine the participating agents and their capabilities, although the precise details of how they will eventually communicate with each other may not be known at design time.

There are a number of ways by which the designer can limit the changes that his design may undergo in the future or at run time to guarantee a certain degree of functionality for the design. Introduction of new agents could be constrained in the Address Book of critical agents stopping them from passing requests to alien agents. Certain special purpose agents such as the input or output agents could also serve to limit unwanted future changes to the system.

3.1. Special purpose agents

Some special purpose agents may be used depending on the application, for example, agents directly involved with input and output, or an agents which interprets the actions of the user as different levels of reward to different system output* (Figure 2)

Input Agents

Inputs to the system may be from more than one source. In such systems, one, or a network of special purpose input agents should be considered, which:

- Unify inputs from different sources into one request, and/or
- Determine commands that all fall into a single request set.

For example if the user's Natural Language (NL) input is: "Information on this" and the mouse is then pointed at an item on the display, these two inputs should be unified as a single request. Interactive input would also require the input agents to determine the end of an input stream. For instance in NL input a phrase (e.g., Do! or Please!) or a relatively long pause, could determine the end of the stream. A different policy here would be to interpret the input in real-time and redirect it to the appropriate agent. As seen in figure 2, agents can redirect the input back to the input agents once this data is no longer relevant to the responding agent.

Output Agents

This special purpose agent decides which response suggested by various agents should be actuated, thus ensuring competition between agents. The decision may be made based on a combination of different criteria and may depend on a specific request. The criteria may be speed, confidence, or other checks that could in turn be made by quality assurance agents. After the final choice has been made, the output agent will ask the specific suggesting agent to actualize its suggestion, or the decision may be redirected to actuator agents. The output agents may be more than one, thus

* One of these agents may be the reward agent itself, thus tuning itself with the user.

breaking the decision making process into a hierarchy. For instance, competing agents could have an output agent decide between them and the output agents in turn could have a higher-level output agent. Safety mechanisms to ensure the requests will not be stuck in cycles of infinite deadlock loops between the agents could be another responsibility of the output agents. Special purpose safeguard agents could also be used.

Feedback Agents

Any adaptive system needs a reward feedback that tells it how far from the optimum its responses have been. This reward could be explicitly input to the system, or implicitly judged from input responses by the system itself. In the case of implicit rewarding, an agent could be responsible for the interpretation of the input behavior and translating it into rewards. The criteria that could be used depend on the system. For instance in an interactive software application a repeat of a command, remarks indicating satisfaction or dissatisfaction, user pause between requests or other such input could be translated into rewards to different output. The feedback agents could also be more than one depending on the different judgement criteria and a hyper-structure [8] or hierarchy might be designed to create the rewards. One way of propagating the feedback reward through the system would be to be aware of the different output and to pass the interpreted reward to the final (output) layer, which will, in turn, pass it back to the previous agents involved in that particular output.

A name server unit is also required to make possible the dynamic introduction of new agents to the system. Each new agent will have to obtain its name (or address) from this name server so that conflicts do not occur and so agents can be referred to throughout the system. Input requests (commands) to the system should be tagged with the user-id that has issued them because interpretation is done on a per-user basis. Reward fed back to the system should also incorporate the request-id to which the reward belongs.

4. Communication Language

In Agent-Based Software Engineering [6], Agents receive and reply to requests for services and information using a declarative knowledge representation language KIF (Knowledge Interchange Format), a communication language KQML (Knowledge Query and Manipulation Language) and a library of formal ontologies defining the vocabulary of various domains. KQML [4] is a superset of what is needed as a communication language between the AAOSA agents and may well be used effectively. The stress on adaptability eliminates the need for elaborate languages and even a simple message passing protocol between the agents should be sufficient. The main request string may be made of different types of information (e.g., Character strings, voice patterns, images, etc...). Various standard information may be passed along with the main request string including: The originator agent, sender agent, the user initiating this request, request id, and/or a time stamp. The request string itself could be comprised of any item of information ranging from natural language requests from the user, to specialized inter-agent messages. Introductory

messages sent by new agents (introduced to the system at run time), could also be incorporated in the request string, or sent under preset standards such as those provided by KQML. Other specialized (and possibly standardized) information that should be passed includes the reward propagation data.

5. An Example: A multimodal map [1]

Multiple input modalities may be combined to produce more natural user interfaces. To illustrate this technique [1] presents a prototype map-based application for a travel-planning domain. The application is distinguished by a synergistic combination of handwriting, gesture and speech modalities; access to existing data sources including the World Wide Web; and a mobile handheld interface. To implement the described application, a distributed network of heterogeneous software agents was augmented by appropriate functionality for developing synergistic multimodal applications. We will consider a simplified subset of this example to show the differences of the two approaches. A map of an area is presented to the user and she is expected to give view port requests (e.g., shifting the map or magnification), or request information on different locations on the map. For example, a user drawing an arrow on the map may want the map to shift to one side. On the other hand the same arrow followed by a natural language request such as: "Tell me about this hotel." May have to be interpreted differently.

[Cheyer et al 96] use the Open Agent Architecture (OAA) [2] as a basis for their design. In this approach, based on a "federation architecture" [9], the software is comprised of a hierarchy of facilitators and agents. The facilitators are responsible for the coordination of the agents under them so that any agent wanting to communicate with any other agent in the system must go through a hierarchy of facilitators (starting from the one directly responsible for it). Each agent, upon introduction to the system, provides the facilitator above it with information on its capabilities (Figure 3). No explicit provision is given for learning.

An example design based on AAOSA is shown in figure 4. It must be noted here that the design shown in figure 4 is not rigid and communication paths may change through time with the agents adapting to different input requests. The NLP and pointer input agents determine the end of their respective inputs and pass them on to the input regulator. This agent in turn determines whether these requests are related or not. It then passes it down to the agent it considers more relevant to the request. The output agents hierarchically sift the outputs suggested by the shifting, magnification, hotels, restaurants and general information agents. Note that combinations of these output suggestions could also be chosen for actuation. The feedback agent provides the system with rewards interpreted from user input. Some of the differences in the two designs are given below:

- The AAOSA design is much more distributed and modular by nature and many of the processes concentrated in the facilitator agents in figure 3 are partitioned and simplified in figure 4.

- AAOSA is more of a network or hyper-structure [8] of process modules as opposed to the hierarchical tree like architecture in the OAA design.
- AI behavior such as natural language processing and machine learning are incorporated on the architecture rather than introduced as new agents (as is the case with the natural language macro agent in figure 3).

It must be stressed that AAOSA like architectures could be achieved with an OAA if we take each OAA facilitator and its macro agents as one agent and add learning capabilities to each facilitator. Another point worth mentioning is that agents in OAA are usually pre-programmed applications linked together through facilitators. The designers have a lesser say over the software architecture as a whole because they are forced to use what has already been designed, possibly without the new higher-level framework in mind.

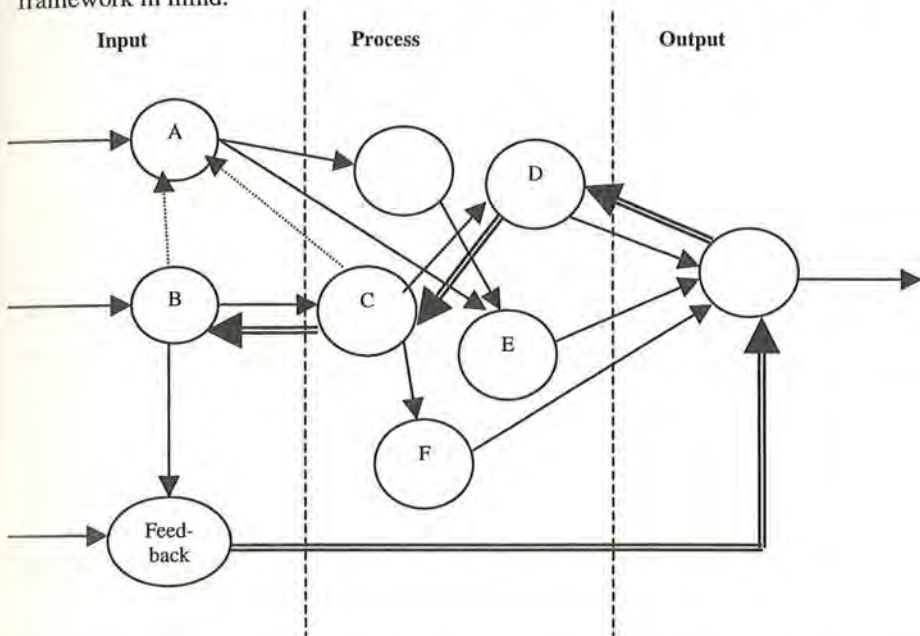


Fig. 2. Example of agent-oriented software architecture. Input agent B may redirect input that does not belong to it to agent A (dotted arrow). This redirection may even happen in later stages (e.g., from C to A). Output suggestions from agents D, E, and F are considered in the output agent and one (in this case D's) is chosen for actuation. The feedback agent uses input directives or indirect behavioral assumptions to calculate a numerical representation of the reward for each output. This reward is then fed back to the agents (e.g., double line arrows).

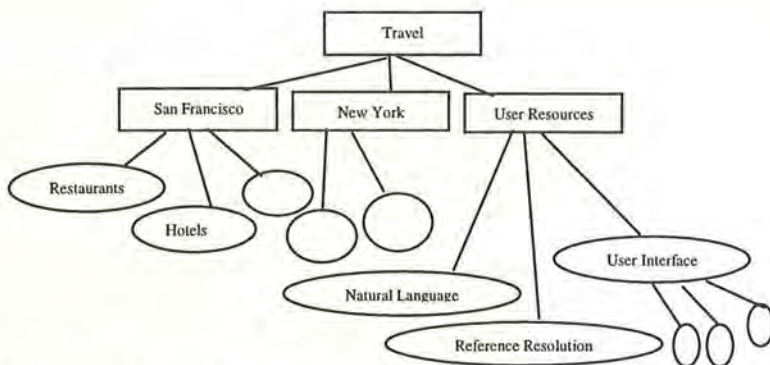


Fig. 3. A structural view of the multimodal map example as designed using OAA in [1]. Boxes represent Facilitators, ellipses represent macro agents and circles stand for modality agents.

6. Interpretation policy

Each agent matches the input pattern with its stored key patterns and finds the closest match and a degree of confidence for it. If this degree of confidence is lower than a threshold, the agent will forward the request to other agents for whom it is confident of their ability to process the request. If no other agent is known with these specifications, either a random action (weighted by the confidence on that action) is chosen, or the request is forwarded to another agent.

In the simplest case the pattern matching process is comprised of checking the presence or absence of certain segments (e.g., words) to determine the course of action that needs to be taken. In more complicated forms patterns should be discovered in the context of the input (e.g., grammar or semantics). This is one reason why if each agent is kept simple in the range of the decisions it needs to make based on its input, the matching and learning process is simplified. A simple pattern matching may be enough to determine the course of action needed. For instance the occurrence of a pointer drag or word patterns including such phrases as "go", or "shift" could cause the map agent in our example (Figure 4) to redirect the request to the shifting agent. Whereas words such as "bigger", "smaller", "magnify", "I can't see" may cause it to send the request to the magnification agent (Figure 5).

Techniques such as those given in [10] could help sort the patterns according to their information value. The nature of the information depends on the application and the agent specializing in it. For example such information as the time between inputs and the loudness or general pattern of the input speech wave could be useful for the feedback agent. The information value of patterns varies depending on the agent (Figure 5).

We will not offer any solutions as to how the interpretation policy of each agent should be stored and updated. Some points that should be taken into consideration while pondering a solution follow:

- It is very important for the interpreter agent to be able to load pre-defined policies at start-up. These are not learned, but hard wired by the engineers. The engineers also determine how much of this initial policy could be undermined through learning.
- It is of equal importance for other agents to be able to contribute to this policy, for instance introducing themselves to the Address-book as possible references.
- The interpretation policy should be dynamic to allow learning of new interpretation rules.
- In many cases the learned policies should be stored on a per-user basis.

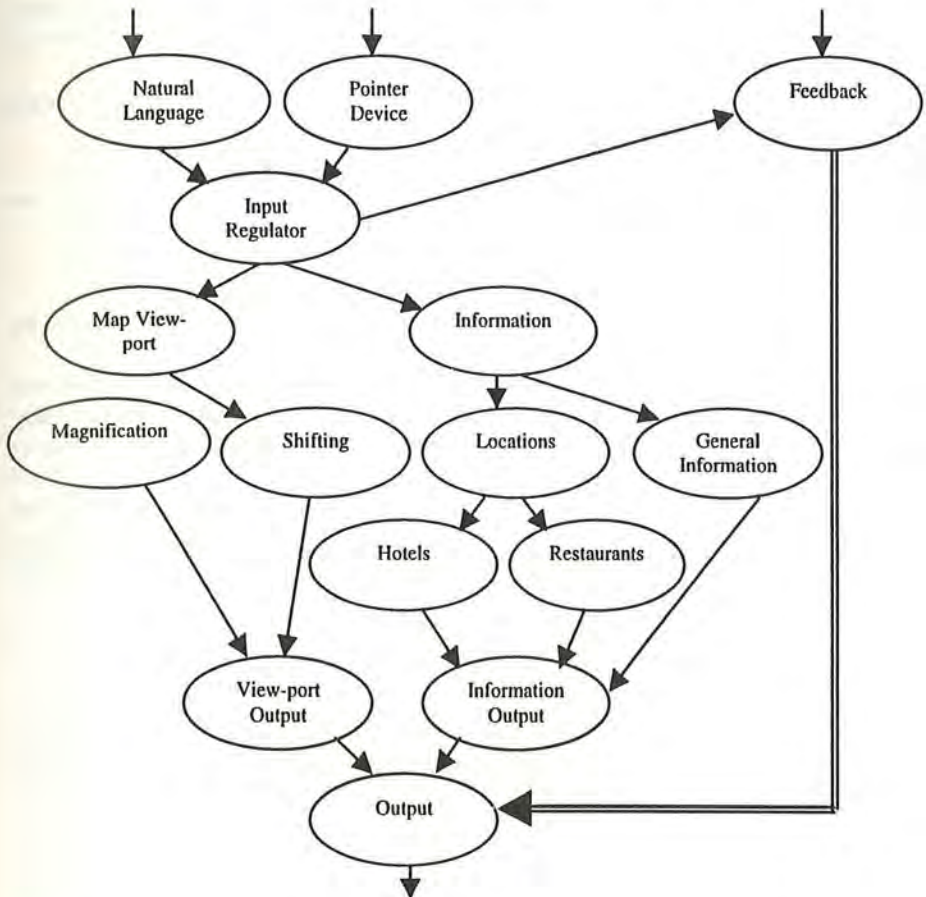


Fig. 4. The multimodal map example as designed based on AAOSA.

7. Learning

Adaptability in AAOSA materializes in three forms:

- The ability of the system to accept new agents at run time,
- The ability of each agent to accept unexpected input or requests,
- The ability of each agent to adapt its behavior according to the feedback it receives (i.e., learning).

Some features of a pattern learning algorithm that may be suitable for AAOSA are briefly mentioned in this section.

Each agent upon the input of a new request goes through the following steps. It must be noted again that the choices mentioned here might be either internal processes or other agents to direct the request to.

- Scan input request for stored patterns estimating a confidence value for each match.
- Choose nearest pattern's choice.
- Keep track of patterns being thrown away in the process of matching as low information patterns [10].
- In case of close ties, choose at random between higher confidence options.
- In case of no reliable match, choose at random between all options*.
- Store request-choice decided upon for adjusting weights and learning until the feedback arrives (delayed reward).

In case of negative reward, patterns in request with highest conflict resolution value should be stored as new decision criteria. These new patterns will be stored according to the user so different users will receive different responses based on their profile in each agent. For example in figure 5 if the input request is slightly changed to: "Shift the *view* to the right", a contradiction will occur. This contradiction could be resolved if the agent identifies the pattern "view" as a higher information value pattern and a new interpretation policy based on the absence or presence of this pattern is conceived.

8. Conclusion

Viewing software as a hyper-structure of Agents (i.e., intelligent beings) will result in designs that are much different in structure and modularization. Some of the benefits of this approach are noted here, some of which are also achievable in object oriented design.

- Flexibility: There is no rigid predetermination of valid input requests.
- Parallelism: The independent nature of the agents creates a potentially parallel design approach.
- Multi platform execution: Agents can run and communicate over networks of computers (on the Internet for instance).
- Runtime addition of new agents and thus incremental development of software.

* Random functions may be weighted according to confidence.

- Software additions by different designers: Different designers can introduce different agents to compete in the software, making this design methodology attractive for commercial applications.

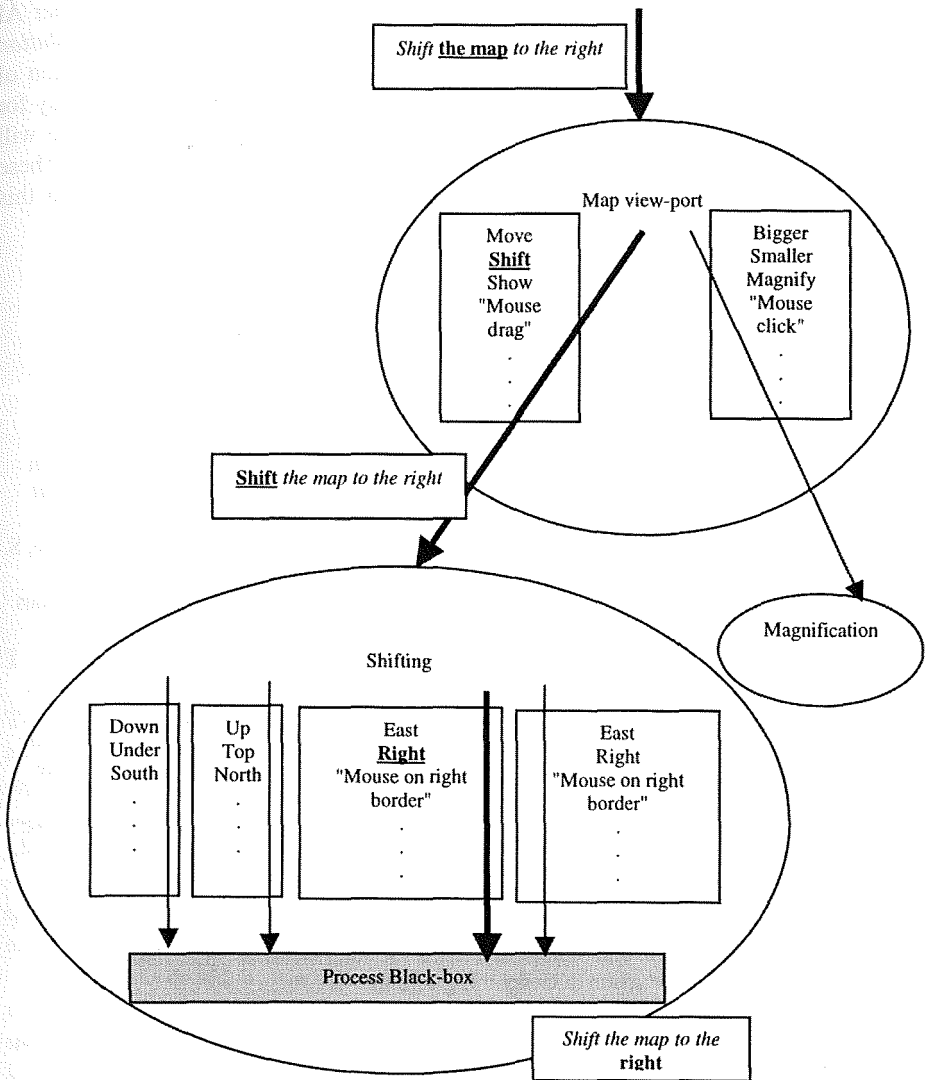


Fig. 5. Each agent needs to identify a small subset of the information in the request and act upon that. This is also an example of distributed Natural Language Processing. Low information (throwaway) patterns (shown in *italic*) vary depending on the agent.

- Reusability of agents.
- Incremental design and evaluation.
- Learning and Intelligence: The distributed nature of learning introduced in this paper suggests a powerful adaptive software design that potentially breaks down an application to a hyper-structure of simple learning modules [8]. Another AI technique that could readily be incorporated into the agents is *artificial evolution* [11]. The mere presence of a reward for each agent makes the introduction of death (removal of an agent from the software) possible. This will make way for other agents, perhaps with better learning techniques, to take over. There will also inevitably be numerous variables to be fine-tuned for each agent. These variables may be thought of as the agent's genes and optimized through this evolutionary process.

References

1. A. Cheyer, L. Julia, Multimodal Maps: An Agent-based Approach, <http://www.ai.sri.com/~cheyer/papers/mmap/mmap.html>, 1996.
2. P. R. Cohen, A. Cheyer, M. Wang, S. C. Baeg, OAA: An Open Agent Architecture, AAAI Spring Symposium, 1994.
3. S. Cranefield, M. Purvis, An agent-based architecture for software tool coordination, in *the proceedings of the workshop on theoretical and practical foundations of intelligent agents*, Springer, 1996.
4. T. Finin, J. Weber, G. Wiederhold, M. Genesereth, R. Fritzson, D. McKay, J. McGuire, S. Shapiro, C. Beck, Specification of the KQML Agent-Communication Language, 1993.
5. S. Franklin, A. Graesser, Is it an Agent or just a Program? A Taxonomy for Autonomous Agents, in: *Proceedings of the Third International Workshop on Agents Theories, Architectures, and Languages*, Springer-Verlag, 1996.
6. M. R. Genesereth, S. P. Ketchpel, Software Agents, Communications of the ACM, Vol. 37, No. 7, July 1994.
7. B. Hayes-Roth, K. Pflieger, P. Lalanda, P. Morignot, M. Balabanovic, A domain-specific Software Architecture for adaptive intelligent systems, IEEE Transactions on Software Engineering, April 1995.
8. B. Hodjat, M. Amamiya, The Self-organizing symbiotic agent, http://www_al.is.kyushu-u.ac.jp/~bobby/1stpaper.htm, 1998.
9. T. Khedro, M. Genesereth, The federation architecture for interoperable agent-based concurrent engineering systems. In *International Journal on Concurrent Engineering, Research and Applications*, Vol. 2, pages 125-131, 1994.
10. R. R. Korfhage, Information Storage and Retrieval, John Wiley & Sons, June 1997.
11. M. Mitchell. An Introduction to Genetic Algorithms. MIT Press, 1996.
12. D. C. Smith, A. Cypher, J. Spohrer, KidSim: Programming Agents without a programming language, Communications of the ACM, Vol. 37, No. 7, pages 55-67, 1994.
13. Y. Shoham, Agent-oriented programming, Artificial Intelligence, Vol. 60, No. 1, pages 51-92, 1993.



US005584024A

United States Patent [19]

[11] Patent Number: **5,584,024**

Shwartz

[45] Date of Patent: **Dec. 10, 1996**

[54] **INTERACTIVE DATABASE QUERY SYSTEM AND METHOD FOR PROHIBITING THE SELECTION OF SEMANTICALLY INCORRECT QUERY PARAMETERS**

FOREIGN PATENT DOCUMENTS

0287310 10/1988 European Pat. Off. G06F 15/40
0387226 9/1990 European Pat. Off. G06F 15/38
63-219034 9/1988 Japan G06F 7/28

[75] Inventor: **Steven P. Shwartz**, Orange, Conn.

OTHER PUBLICATIONS

Wu, "A Knowledge-Based Database Assistant With A Menu Based Natural Language User-Interface" 10 Oct. 1993, IEICI: Trans. Inf. & Syst. V. E76-D N. 10 pp. 1276-1287.

[73] Assignee: **Software AG**, Germany

(List continued on next page.)

[21] Appl. No.: **217,099**

[22] Filed: **Mar. 24, 1994**

Primary Examiner—Wayne Amsbury

Assistant Examiner—Jack M. Choules

Attorney, Agent, or Firm—Howrey & Simon; C. Scott Talbot; Thomas G. Woolston

[51] Int. Cl.⁶ **G06F 17/30**; G06F 17/27

[52] U.S. Cl. **395/604**; 395/922; 395/757; 364/274.2; 364/275.4; 364/283.3; 364/DIG. 1; 364/972.2; 364/974.6; 364/DIG. 2

[58] Field of Search 395/600, 922; 364/419.01, 419.07, 974.6, 972.2, 274.2, 274.7, 275.1, 275.4, 283.3, 282.1

[57] ABSTRACT

A database query system includes a query assistant that permits the user to enter only queries that are both syntactically and semantically valid (and that can be processed by an SQL generator to produce semantically valid SQL). Through the use of dialog boxes, a user enters a query in an intermediate English-like language which is easily understood by the user. A query expert system monitors the query as it is being built, and using information about the structure of the database, it prevents the user from building semantically incorrect queries by disallowing choices in the dialog boxes which would create incorrect queries. An SQL generator is also provided which uses a set of transformations and pattern substitutions to convert the intermediate language into a syntactically and semantically correct SQL query.

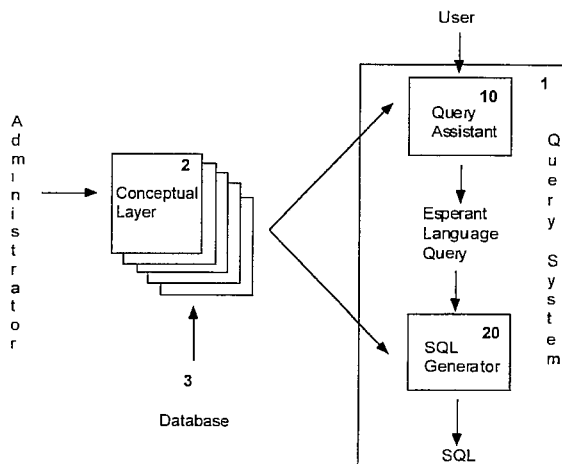
[56] References Cited

U.S. PATENT DOCUMENTS

4,506,326	3/1985	Shaw et al.	364/300
4,688,195	8/1987	Thompson et al.	364/300
4,689,737	8/1987	Grant	364/200
4,736,296	5/1988	Katayama et al.	364/419
4,811,207	3/1989	Hikita et al.	364/200
4,829,423	5/1989	Tennant et al.	364/200
4,839,853	6/1989	Deerwester et al.	364/900
4,914,590	3/1990	Loatman et al.	364/419
4,930,071	5/1990	Tou et al.	364/300
4,931,935	5/1990	Ohira et al.	364/419
4,943,933	7/1990	Miyamoto et al.	364/513
4,974,191	11/1990	Amirghodsi et al.	364/900
4,994,967	2/1991	Asakawa	364/419
5,099,426	3/1992	Carligen et al.	364/419
5,175,814	12/1992	Anick et al.	395/161
5,197,005	3/1993	Shwartz et al.	395/600
5,204,947	4/1993	Bernstein et al.	395/157
5,237,502	8/1993	White et al.	364/419
5,255,386	10/1993	Prager	395/600
5,265,014	11/1993	Haddock et al.	364/419
5,265,065	11/1993	Turtle	395/600
5,349,526	9/1994	Potts, Sr. et al.	364/419.1
5,386,556	1/1995	Hedim et al.	395/600

The intermediate language can represent complex SQL queries while at the same time being easy to understand. The intermediate language is also designed to be easily converted into SQL queries. In addition to the query assistant and the SQL generator, an administrative facility is provided which allows an administrator to add a conceptual layer to the underlying database making it easier for the user to query the database. This conceptual layer may contain alternate names for columns and tables, paths specifying standard and complex joins, definitions for virtual tables and columns, and limitations on user access.

27 Claims, 26 Drawing Sheets



OTHER PUBLICATIONS

- Wu et al, "KDA: A Knowledge-Based Database Assistant With Query Guiding Facility" 5 Oct. 1992, pp. 443-453, IEEE Transactions On Knowledge & Data Eng. V. 4 N. 5.
- Cha, "Kaleidoscope: A Cooperative Menu Guided Query Interface (SQL Version)," 1990, IEEE, Artificial Intelligence Applications.
- Whitaker and Bonnell "Functional Modelling Of Intelligent Systems Using A Blackboard Model" 1992, pp. 1 to 12, The Journal Of Knowledge Eng. V 5, N. 1.
- Winston, P., "Language Understanding," *Artificial Intelligence*, 9:291-334 (1984).
- Rich, E., "Natural Lanugage Interfaces," *Computer*, pp. 39-47 (Sep. 1984).
- Manferdelli, J. L., "Natural Language Interfaces: Benefits, Requirements, State of the Art and Applications," *AI East*, Oct. 1987.
- Schank, R. C., et al, "Inside Computer Understanding: Five Programs Plus Miniatures," 14:354-372, LEA, Publishers, Hillsdale, NJ (1981).
- Hendrix, G., "The Lifer Manual: A Guide to Building Practical Natural Language Interfaces" (Technical Note 138); SRI International, Feb. 1977.
- Hendrix, G., "Human Engineering for Applied Natural Language Processing" (Technical Note 139); SRI International, SRI Project 740D32 CTC, Feb 1977.
- Kao, M., et al, "Providing Quality Responses with Natural Language Interfaces: The Null Value Problem," *IEEE* 14:7, 959-984, Jul., 1988.
- Chapter 6, "Queries," *Building Access Z Applications*, 1995.
- Chapter 8, "Using Query by Example," *Using Access 2 for Windows*, Sp. Ed.
- Chapter 9, "Querying Your Data," *Inside Paradox 5 for Windows*, 1994.
- Cinque, L., et al, "An Expert Visual Query System," *J. Vis. Lang. and Comp.*, 2:101-113 (1991).
- Meng, W., et al. "A Theory of Translation From Relational Queries to Hierarchial Queries," *IEEE Transactions on Knowledge and Data Engineering*, 7:2, 228-245, Apr. 1995.
- Jakobson, G., et al, "CALIDA: A System for Integrated Retrieval from Multiple Heterogeneous Databases," Proceedings of the 3rd Int'l Conf. on Data and Knowledge Bases: Improving Usability and Responsiveness, Jun. 28-30, Jerusalem, Israel.

CUSTOMERS							
CUSTO-MER #	NAME	CITY	STATE	ZIP_CODE	SALES_PERSON#	CREDIT_LIMIT	BALANCE
1	American Butcher Block	New Haven	CT	16516	1	65000	85000
2	Barn Door Furniture	New York	NY	11019	1	50000	75000
3	Bond Dinettes	Boston	MA	22827	1	50000	500
4	Carroll Cut-Rate	Los Angeles	CA	23019	2	50000	0
5	Porch and Patio	San Francisco	CA	24082	3	65000	0
6	Railroad Salvage	Bridgeport	CT	26444	4	30000	750
7	Sheffield Showrooms	Brooklyn	NY	12018	5	50000	12050
8	Spector Furniture	New Bedford	MA	22451	5	50000	100
9	Vista Designs	Stamford	CT	26565	5	30000	4300
10	Milford Furniture	Milford	CT	26460	5	50000	0

FIG. 1A

PRODUCTS							
PRO-DUCT#	NAME	GROUP_ID	TYPE_ID	ABC_CODE	PRICE	VENDOR #	ALT_VENDOR #
1	Executive Desk	100	300	A	4995	1	2
2	Colonial Bedroom Set	100	600	B	3495	1	
3	Children's Bedroom Set	100	300	C	1195	2	3
4	5 Pc. Living Room Set	100	500	A	4895	2	4
5	Crib / Dresser Set	100	400	B	985	3	3
6	Bunk Bed	100	400	C	795	3	
7	3 Pc. Dining Room Set	100	500	A	3995	4	
8	4 Pc. Office Set	200	300	B	2995	4	2
9	Child's Desk	100	400	C	899	5	1
10	4 Pc. Occasional Tables Set	100	500	B	1299	5	1

FIG. 1B

VENDORS		
VENDOR#	NAME	PHONE
1	Chapel Hill Furniture	412-748-2929
2	Barnet Furniture	314-345-6789
3	Basset Inc.	218-324-9288
4	Seal Corporation	509-929-2222
5	Juvenile Warehouse	413-345-5656

FIG. 1C

SALESPEOPLE		
SALESPERSON#	NAME	STATE
1	Paul Williams	NC
2	Bill Smith	KY
3	Wendy Jones	MA
4	Pam Johnson	CT
5	Sam Rogers	NY

FIG. 1D

CODES	
CODE_ID	CODE_TEXT
100	Home Furnishings
200	Office Furnishings
300	Desks
400	Juvenile Furnishings
500	Occasional Furnishings

FIG. 1E

ORDERS							
ORDER #	CUSTO-MER#	ORDER_DATE	ORDER_DOLLARS	FREIGHT_DOLLARS	SALES-PERSON #	SHIP_DATE	STATUS
1000001	1	1992-1-5	36115	124	1	1992-1-15	
1000002	1	1992-4-5	7514	124	1	1992-10-24	
1000003	1	1992-7-5	39540	124	1	1992-7-15	
1000004	2	1992-2-15	47320	124	2	1992-2-24	B
1000005	2	1992-5-15	76195	124	2	1992-5-15	B
1000006	2	1992-8-15	6010	124	2	1992-9-15	
1000007	3	1992-3-25	3225	124	3	1992-3-27	
1000008	3	1992-6-25	12005	124	3	1992-7-2	
1000009	3	1992-9-25	36240	0	3	1992-10-15	
1000010	4	1992-2-5	8115	62	4	1992-2-15	
1000011	4	1992-5-15	20150	124	4	1992-5-24	
1000012	4	1992-8-25	25110	124	4	1992-8-29	
1000013	5	1992-3-4	72115	124	4	1992-3-22	B
1000014	5	1992-6-14	83550	124	4	1992-6-28	
1000015	5	1992-9-24	67015	124	4	1992-10-10	
1000016	6	1992-4-2	12105	124	1	1992-4-10	
1000017	6	1992-7-12	18105	124	1	1992-7-29	
1000018	6	1992-10-22	7350	124	1	1992-11-12	
1000019	6	1992-5-1	2660	124	1	1992-5-15	
1000020	6	1992-8-11	45250	124	1	1992-8-22	
1000021	7	1992-11-11	21500	124	2	1992-11-22	
1000022	7	1992-6-5	6150	62	2	1992-6-25	
1000023	7	1992-8-15	73595	124	2	1992-8-25	
1000024	9	1992-11-16	6100	124	3	1992-11-24	
1000025	9	1992-1-20	3250	124	3	1992-1-24	
1000026	9	1992-4-22	28190	124	3	1992-4-26	
1000027	9	1992-7-25	24250	124	3	1992-7-29	
1000028	10	1992-2-7	3620	124	5	1992-2-27	B
1000029	10	1992-5-17	107620	62	5	1992-5-26	

FIG. 1F

LINE_ITEMS						
ORDER#	LINE #	PRODUCT#	QTY_ ORDERED	QTY_BACK- ORDERED	WAREHOUSE#	
1000001	1	1	2	5	1	
1000001	2	2	0	3	1	
1000002	1	3	2	6	1	
1000003	1	4	0	8	1	
1000004	1	5	0	2	1	
1000004	2	6	3	3	1	
1000004	3	7	0	5	2	
1000004	4	8	2	7	2	
1000004	5	9	0	2	2	
1000005	1	10	2	8	2	
1000005	2	1	2	3	1	
1000005	3	2	9	9	1	
1000005	4	3	4	4	1	
1000005	5	4	0	3	1	
1000006	1	5	0	6	1	
1000007	1	6	0	4	1	
1000008	1	7	0	3	2	
1000009	1	8	0	12	2	
1000010	1	9	0	9	2	
1000011	1	10	0	15	2	
1000012	1	1	0	5	1	
1000013	1	2	0	9	1	
1000013	2	3	0	7	1	
1000013	3	4	4	4	1	
1000013	4	5	5	5	1	
1000013	5	6	7	7	1	
1000014	1	7	0	2	2	
1000014	2	8	0	6	2	
1000014	3	9	0	1	1	
1000014	4	10	0	8	1	
1000014	5	1	0	9	2	

FIG. 1G

LINE_ITEMS						
ORDER#	LINE #	PRODUCT#	QTY_ ORDERED	QTY_BACK- ORDERED	WAREHOUSE#	
1000015	1	2	0	4	2	
1000015	2	3	0	7	2	
1000015	3	4	0	4	2	
1000015	4	5	0	7	1	
1000015	5	6	0	9	1	
1000016	1	7	0	3	1	
1000017	1	8	0	6	2	
1000018	1	9	0	8	2	
1000019	1	10	0	2	1	
1000020	1	1	0	9	1	
1000021	1	2	0	6	1	
1000022	1	3	0	5	2	
1000023	1	4	0	15	2	
1000024	1	5	0	6	1	
1000025	1	6	0	4	1	
1000026	1	7	0	7	1	
1000027	1	8	0	8	1	
1000028	1	9	0	4	2	
1000029	1	10	0	8	2	
1000029	2	1	0	5	1	
1000029	3	2	0	9	1	
1000029	4	3	0	6	1	
1000029	5	4	0	4	1	
1000029	6	5	0	9	1	
1000029	7	6	0	3	2	

FIG. 1G (Continued)

Query Builder		
Tables	Fields	
CUSTOMER.DB		
Sort Order		
Conditions		
OK	Cancel	Help

Figure 2A

Query Builder		
Tables	Fields	
CUSTOMER.DB	NAME STATE BALANCE	
Sort Order		
NAME STATE		
Conditions		
OK	Cancel	Help

Figure 2B

Edit Query

SQL

```
SELECT
  NAME
  STATE
  BALANCE
FROM
  CUSTOMER.DB
ORDER BY
  NAME
  STATE
```

OK Cancel Help

Figure 2C

Conditions

Conditions

CREDIT > 50000

Connector	Field	Operator	Expression
AND		>	

OK Cancel Help

Figure 2D

Query Builder

<p>Tables</p> <div style="border: 1px solid black; padding: 2px;">CUSTOMER.DB</div> <p>Sort Order</p> <div style="border: 1px solid black; padding: 2px;">NAME STATE</div> <p>Conditions</p> <div style="border: 1px solid black; padding: 2px;">CREDIT > 50000</div>	<p>Fields</p> <div style="border: 1px solid black; padding: 2px;">NAME STATE BALANCE</div>
--	---

Figure 2E

Query1		
NAME	STATE	BALANCE
American Butcher Block	CT	85000
Porch And Patio	CA	0

Figure 2F

Query Builder	
<p>Tables</p> <div style="border: 1px solid black; padding: 2px; min-height: 20px;">CUSTOMER.DB</div>	<p>Fields</p> <div style="border: 1px solid black; padding: 2px; min-height: 40px;"> NAME STATE BALANCE </div>
<p>Sort Order</p> <div style="border: 1px solid black; padding: 2px; min-height: 20px;"> NAME STATE </div>	
<p>Conditions</p> <div style="border: 1px solid black; padding: 2px; min-height: 20px;">CREDIT > 50000</div>	
<p>Group By</p> <div style="border: 1px solid black; padding: 2px; min-height: 20px;">SALESPERSON#</div>	<p>Having</p> <div style="border: 1px solid black; padding: 2px; min-height: 20px;">SUM(BALANCE) > \$80,000</div>
<div style="display: flex; justify-content: space-between; margin-top: 10px;"> OK Cancel Help </div>	

Figure 2G

Join Tables		
<p>CUSTOMER.DB</p> <div style="border: 1px solid black; padding: 2px; min-height: 80px;"> CUSTOMER# NAME CITY STATE ZIP_CODE SALESPERSON# CREDIT BALANCE </div>	<div style="border: 1px solid black; padding: 2px; min-height: 80px; text-align: center;"> <> = >= <= > < </div>	<p>ORDER.DB</p> <div style="border: 1px solid black; padding: 2px; min-height: 80px;"> ORDER# CUSTOMER# ORDER_DATE ORDER_DOLLARS FREIGHT_DOLLARS SALESPERSON# SHIP_DATE STATUS </div>

Figure 2H

Natural Language

File Questions Report Column/Table Configure Query Graph Help

Natural Language

: What were the 5 most common defects last month?
What were the 5 defects that occurred the most in June, 1991

<u>Defect</u>	<u>Count</u>
Contamination	1213
Damage in handling	516
Surface finish	423
Bad soldering	315
Cracked board	273

: Show the SQL
List the query.
Query for: What were the 5 most common defects last month?
select repair.def_cd, count(repair.mod_pn)
from repair
where repair.tstr_strt>='6/1/1991'
and repair.tstr_strt<'7/1/1991'
group by repair.def_cd
order by 2 desc;
:

Figure 3A

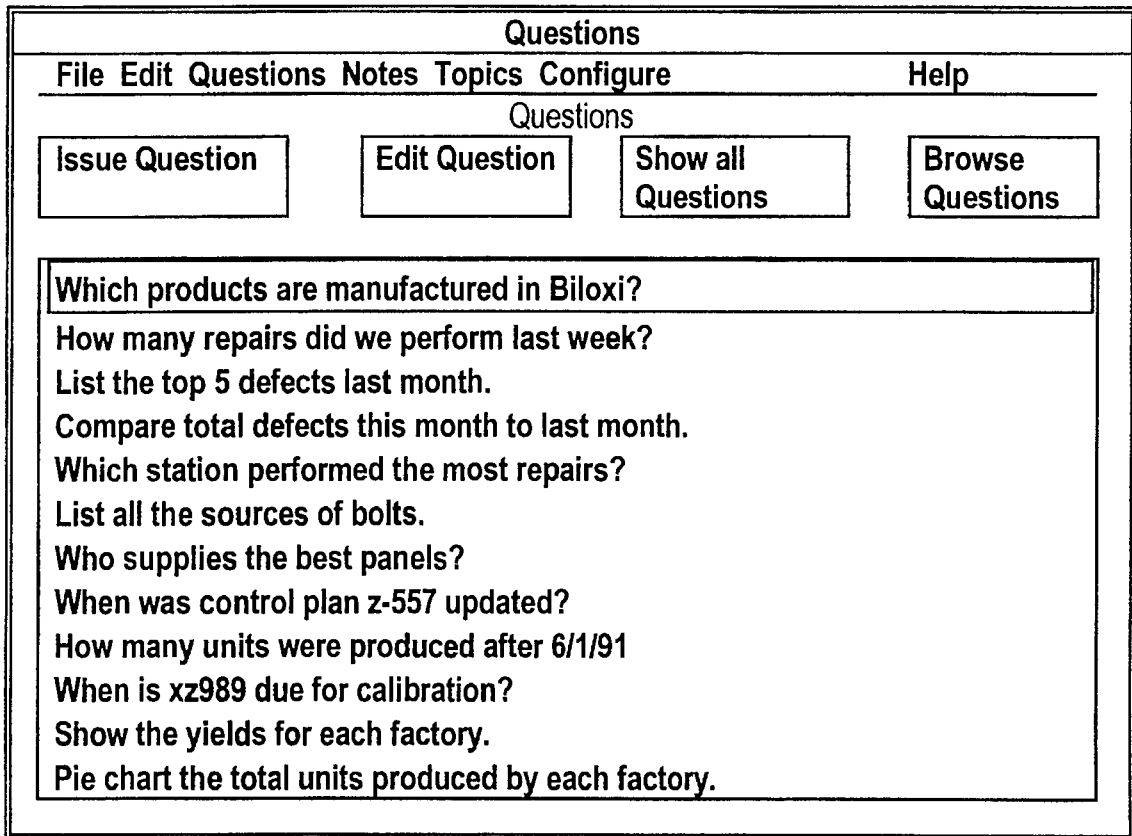


Figure 3B

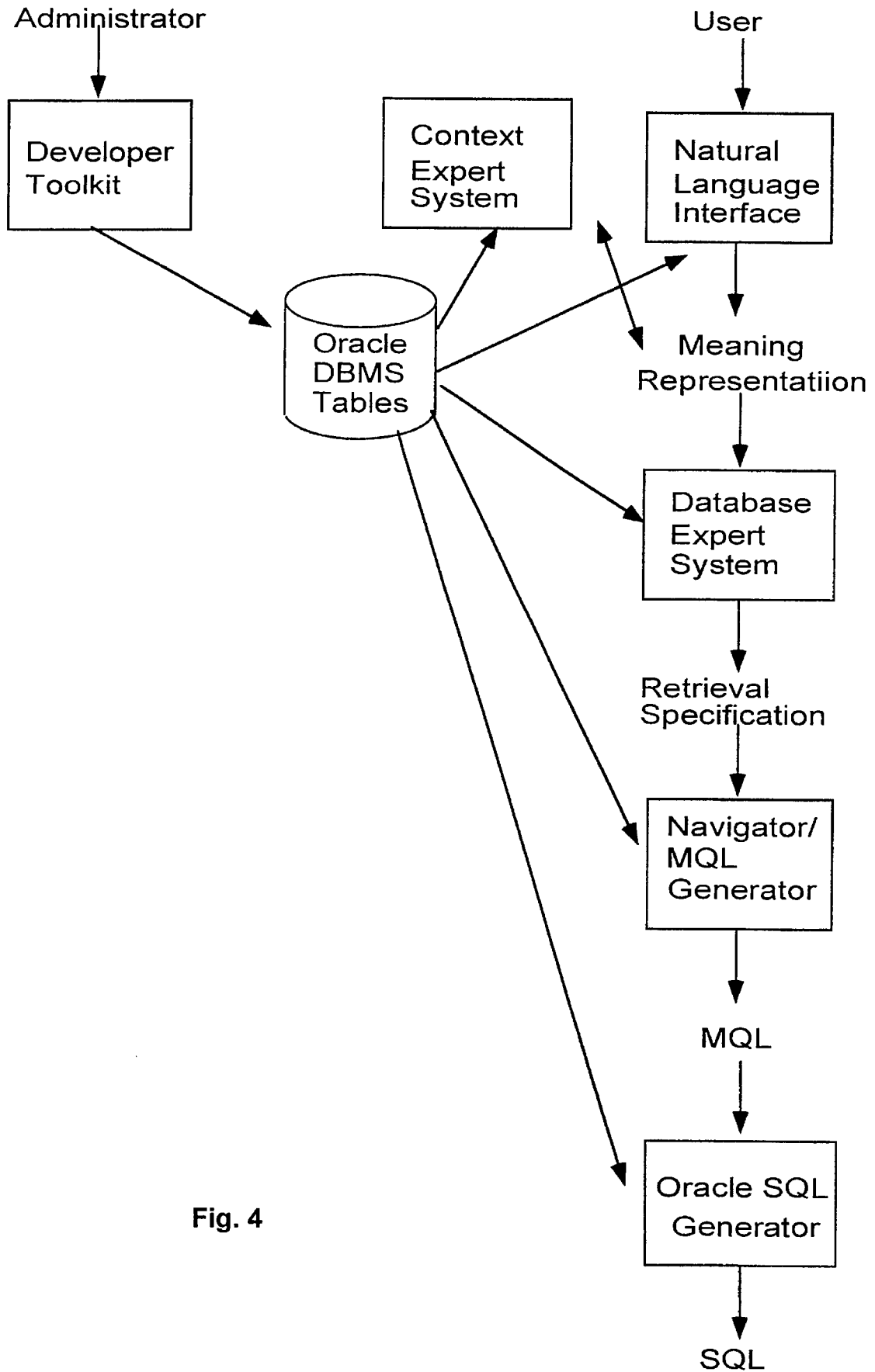


Fig. 4

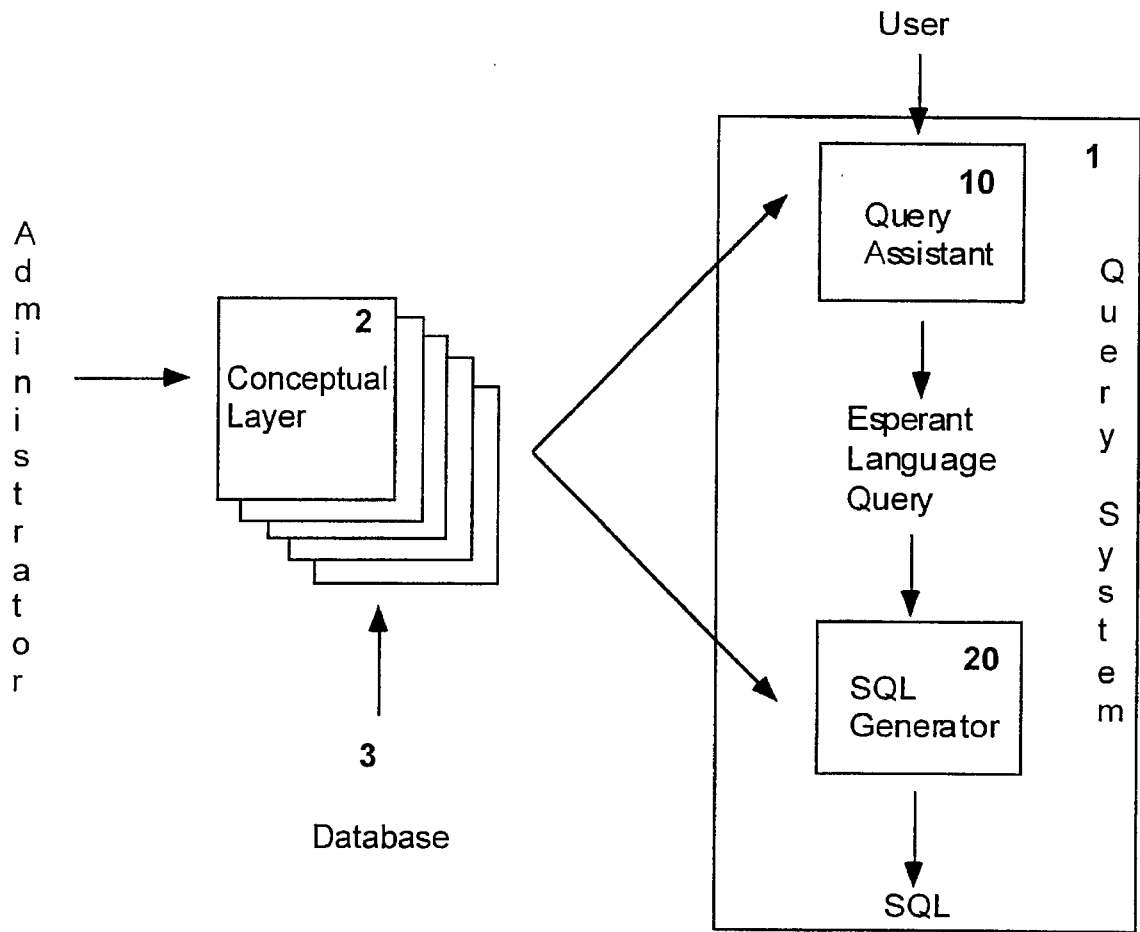


FIG. 5

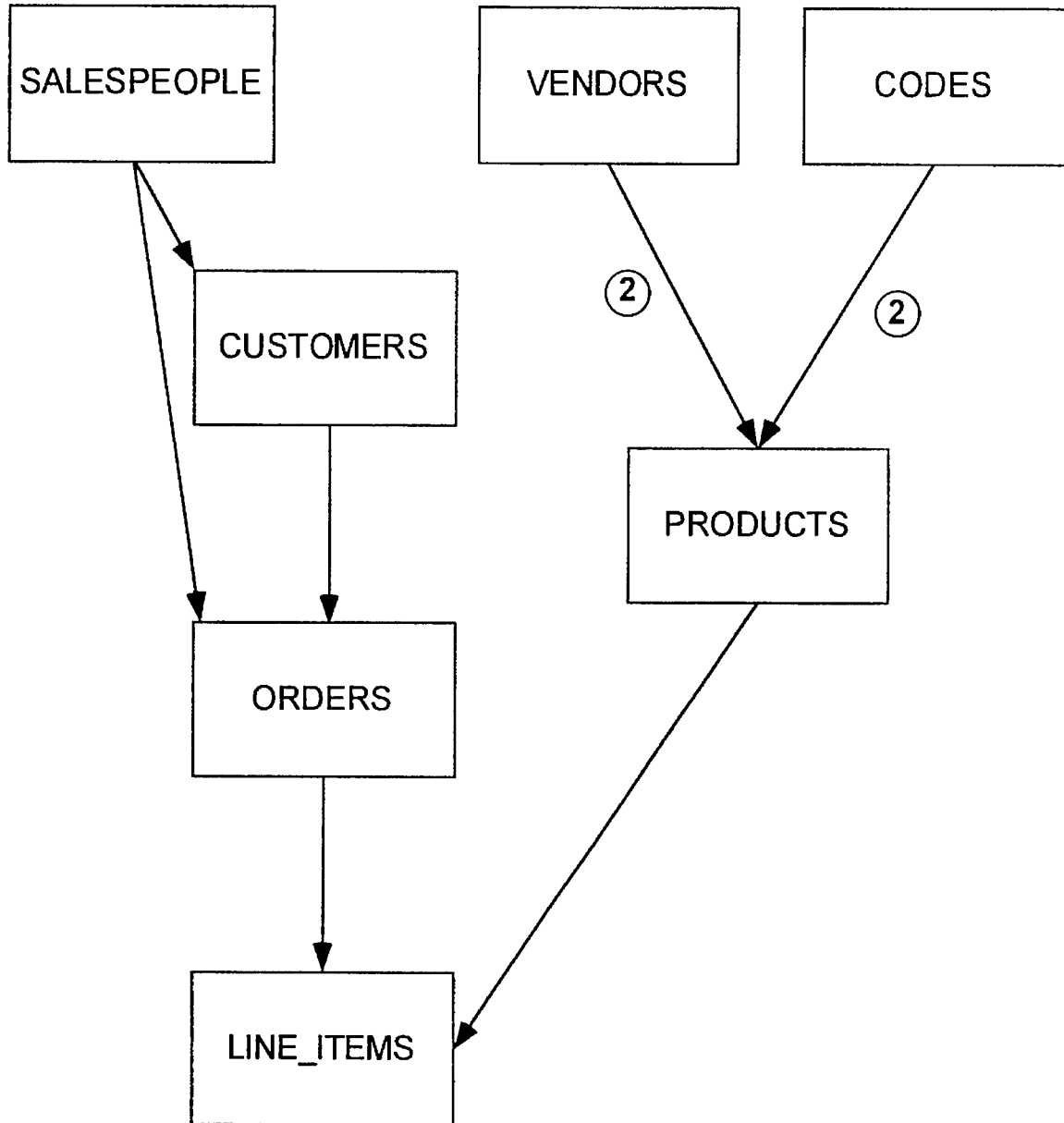


FIG. 6

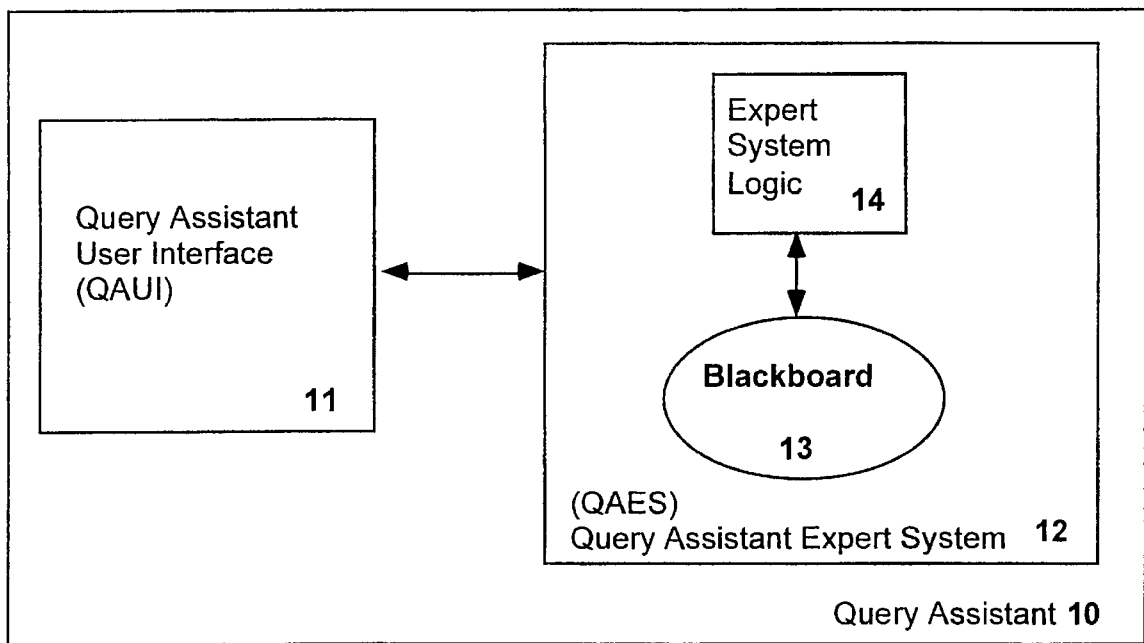


FIG. 7

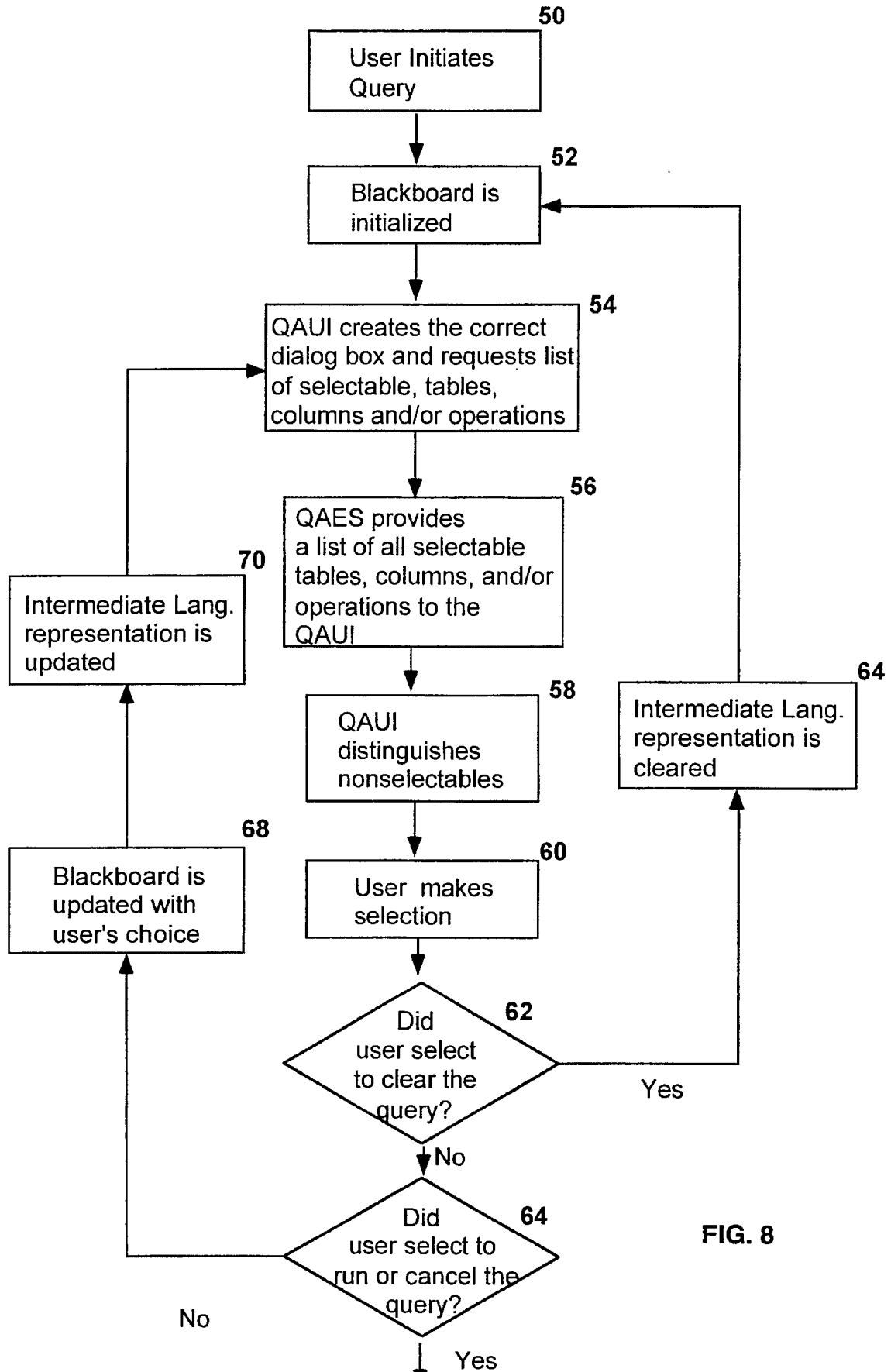
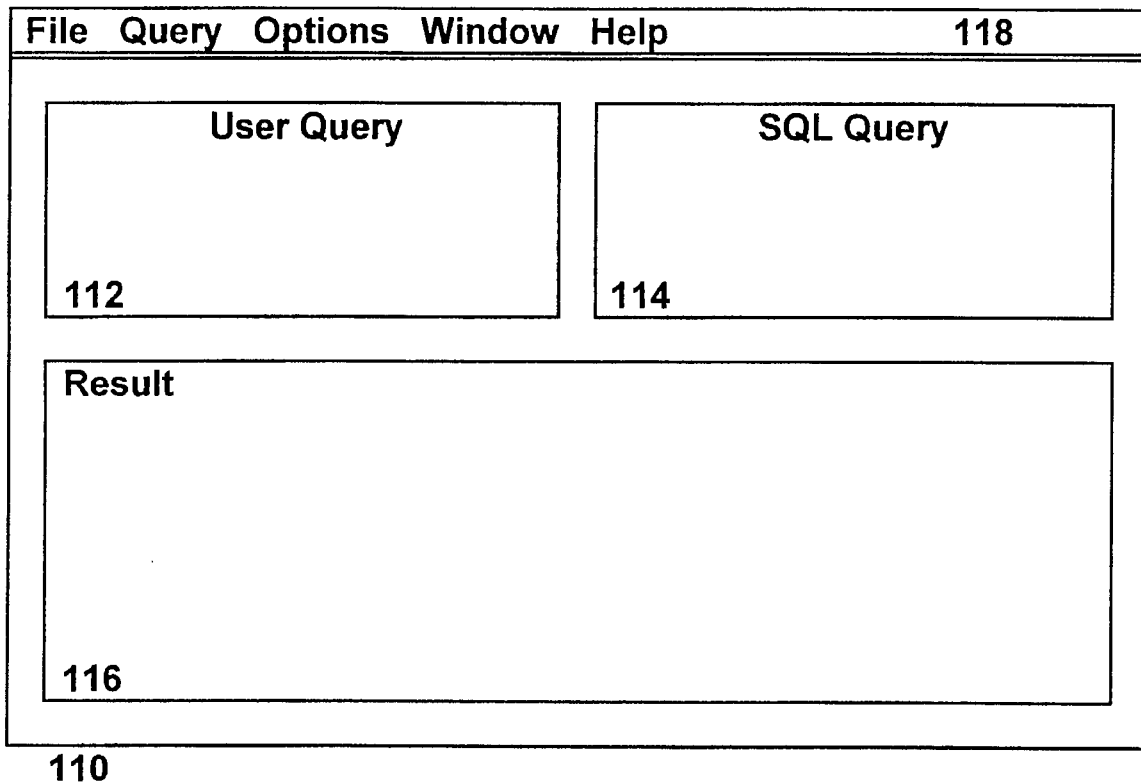


FIG. 8



110

Fig. 9

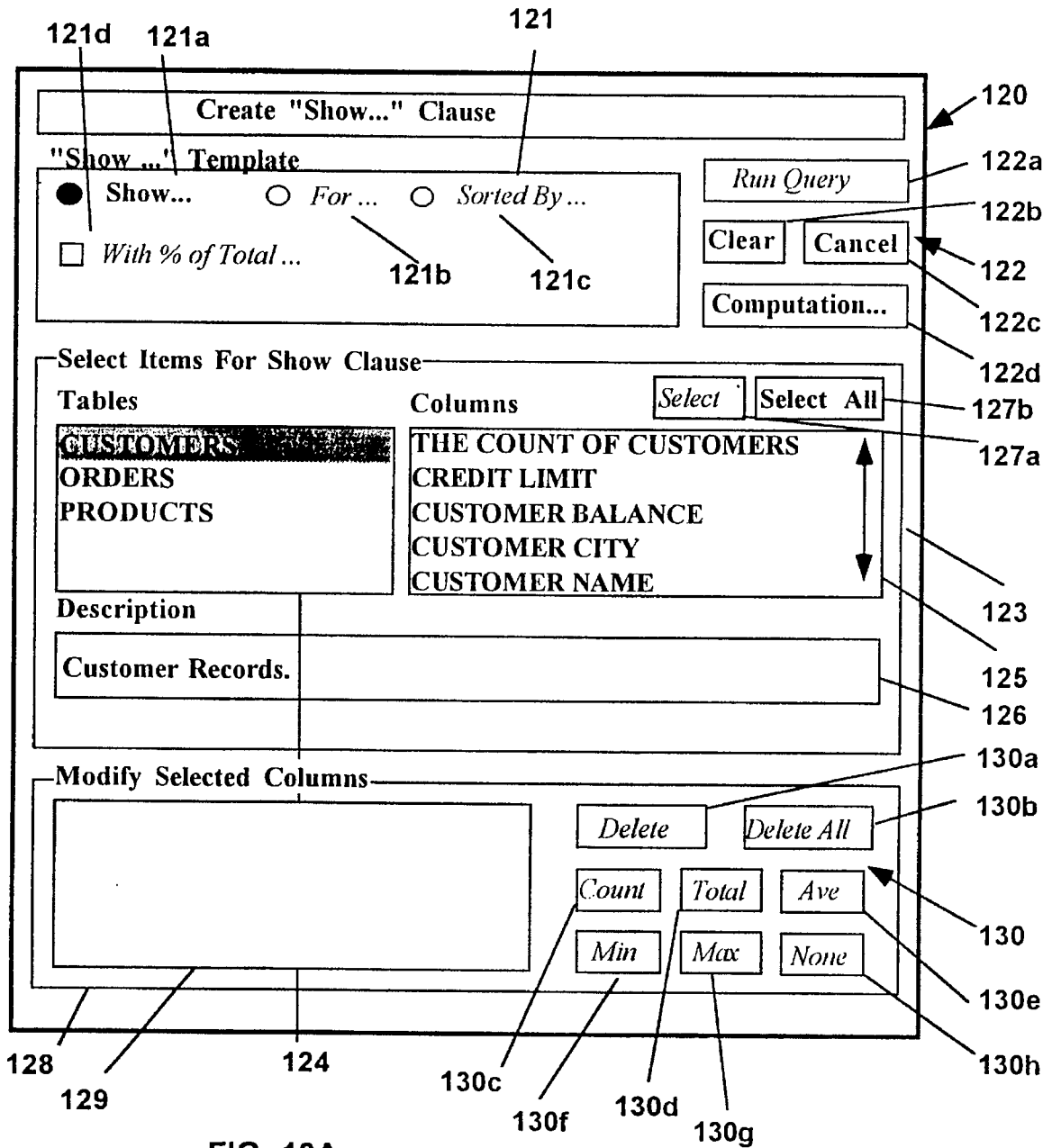


FIG. 10A

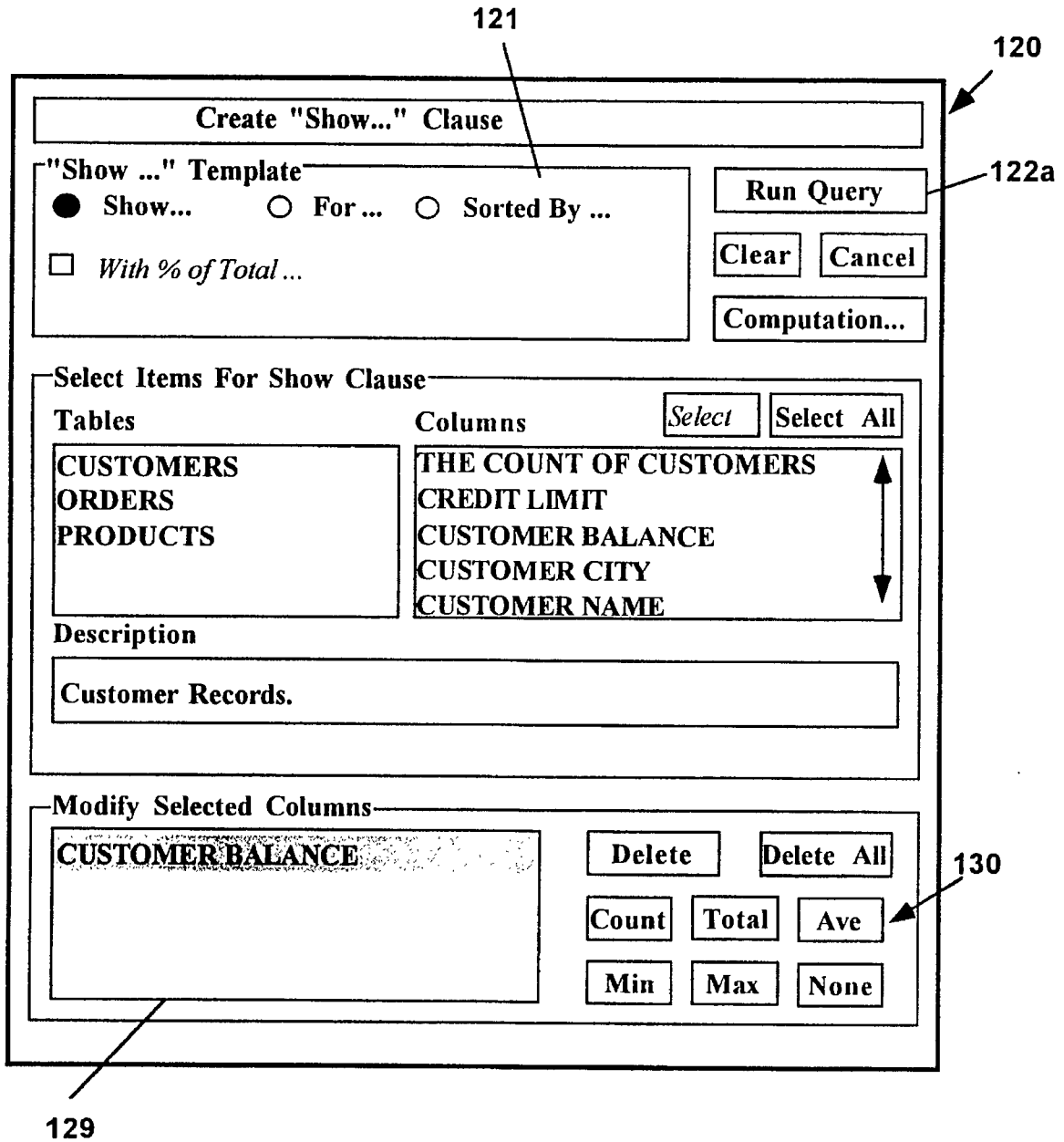


FIG. 10B

Create "Show..." Clause

"Show ..." Template

Show... For ... Sorted By ...

With % of Total ...

Run Query Clear Cancel Computation...

Select Items For Show Clause

Tables	Columns	Select	Select All
CUSTOMERS ORDERS PRODUCTS	THE COUNT OF CUSTOMERS CREDIT LIMIT CUSTOMER BALANCE CUSTOMER CITY CUSTOMER NAME		

Description

The Customer's credit limit.

Modify Selected Columns

THE AVERAGE CUSTOMER BALAN

Delete Delete All

Count Total Ave

Min Max None

120

130b

FIG 10C

Computation

Select a column

<input type="radio"/> Total	<input type="radio"/> Minimum
<input type="radio"/> Average	<input type="radio"/> Maximum

OK
Cancel

Tables	Columns	Select
CUSTOMERS ORDERS PRODUCTS	THE COUNT OF CUSTOMERS CREDIT LIMIT CUSTOMER BALANCE CUSTOMER CITY CUSTOMER NAME CUSTOMER NUMBER	↑ ↓

9	8	7
6	5	4
3	2	1
DEL	0	.

+
-
*
/

()
---	---

Clear

Computation:

135

FIG 10D

Create "Sorted By..." Clause

"Show ..." Template

Show... For ... Sorted By ...

With % of Total ...

Run Query **Clear** **Cancel**

Select Items

Tables	Columns	Select
CUSTOMERS ORDERS PRODUCTS	THE COUNT OF CUSTOMERS CREDIT LIMIT CUSTOMER BALANCE CUSTOMER CITY CUSTOMER NAME	↑ ↓

Select Items Already in the Show Clause

THE AVERAGE CUSTOMER BALANCE

Select

Modify Selected Columns

Delete
Ascending
Decending

← 140

FIG 10E

Create "For..." Clause

"Show ..." Template

Show... For ... Sorted By ...

With % of Total ...

Run Query

Clear **Cancel**

Choose one of the following

THAT HAVE

THAT DO NOT HAVE

Select

Backup

Help

151

150

FIG 10F

Create "For..." Clause

"Show ..." Template

Show... For ... Sorted By ...

With % of Total ...

Run Query

Clear **Cancel**

Select an Item

Total Minimum

Average Maximum

Select

Backup

Computation...

Tables	Columns
CUSTOMERS ORDERS PRODUCTS	THE COUNT OF CUSTOMERS CREDIT LIMIT CUSTOMER BALANCE CUSTOMER CITY CUSTOMER NAME

Description

Customer Records.

160

FIG 10G

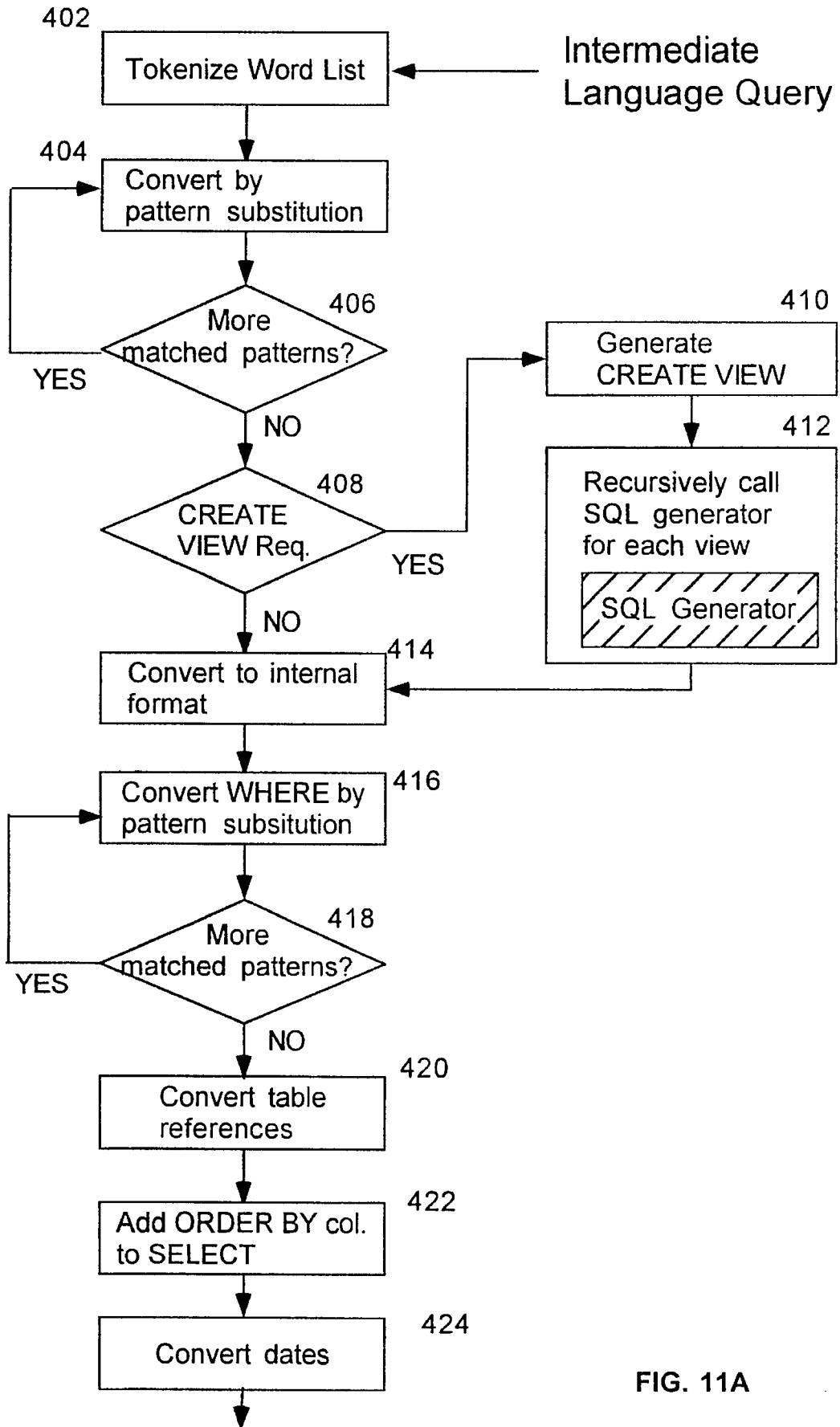
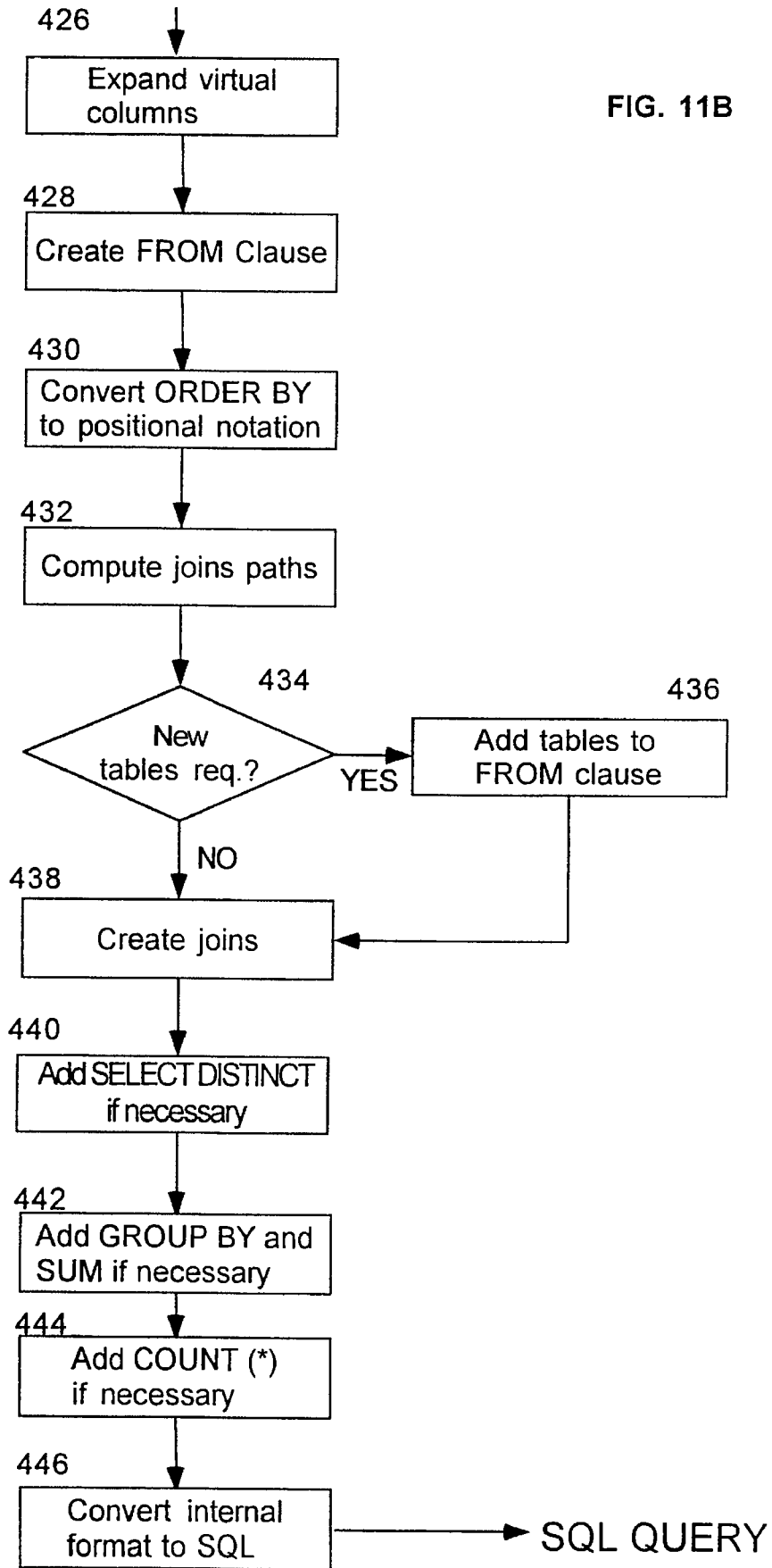


FIG. 11A

FIG. 11B



**INTERACTIVE DATABASE QUERY SYSTEM
AND METHOD FOR PROHIBITING THE
SELECTION OF SEMANTICALLY
INCORRECT QUERY PARAMETERS**

BACKGROUND OF THE INVENTION

The invention relates to a database querying tool, and specifically to a database querying tool which will guide a user to interactively create syntactically and semantically correct queries.

End user workstations are being physically connected to central databases at an ever increasing rate. However, to access the information contained in those databases, users must create queries using a standardized query language which in most instances is Structured Query Language (SQL). Most information system organizations consider it unproductive to try and teach their users SQL. As a result there is an increasing interest in tools that create SQL for the user using more intuitive methods of designating the information desired from the database. These tools are generally called SQL Generators.

Most SQL Generators on the market today appear to hide the complexities of SQL from the user. In reality, these tools accomplish this by severely limiting the range of information that can be retrieved. More importantly, these tools make it very easy for users to get incorrect results. These problems arise out of the reality that SQL is very difficult to learn and use. Existing technologies designed to shield users from the complexities of SQL can be grouped into three categories: point-and-shoot menus; natural language systems; and natural language menu systems. Each of these three categories of product/technology have architectural deficiencies that prevent them from truly shielding users from the complexities of SQL.

Limitations of SQL As An End User Query
Language

SQL is, on the whole, very complex. Some information requirements can be satisfied by very simple SQL statements. For example, to produce from a database a list of customer names and phones for New York customers sorted by zip code, the following SQL statement could be used:

```
(1)  SELECT NAME, PHONE
      FROM CUSTOMERS
      WHERE STATE = 'NY'
      ORDER BY ZIP_CODE
```

In this example, the SELECT command defines which fields to use, the WHERE command defines a condition by which database records are selected, and ORDER BY keywords define how the output should be sorted. The FROM keyword defines in which tables the fields are located. Unfortunately, only a relatively small percentage of information required can be satisfied with such simple SQL

Most information needs, even very simple queries, require complex SQL queries. For example, the SQL statement required to generate a list of orders that have more than two products on backorder, is:

```
(2)  SELECT T1.ORDER#, T1.ORDER_DATE, T1.ORDER_DOLLARS
      FROM ORDERS T1
      WHERE 2 < (
          SELECT COUNT (*)
          FROM PRODUCTS T2, LINE_ITEMS T3
          WHERE T3.QTY_BACKORDERED > 0
          AND T2.PRODUCT# = T3.PRODUCT#
          AND T1.ORDER# = T3.ORDER#)
```

This SQL statement contains two SELECT clauses, one nested with the other. For a user to know that this information requirement needs an SQL query involving this type of nesting (known as a correlated subquery) implies some understanding by the user of the relational calculus. However, except for mathematicians and people in the computer field, few users have this skill. The following are some examples of database queries that require more complex SQL constructs:

GROUP BY: Approximately 75% of all ad hoc queries require a GROUP BY statement in the SQL. Examples include:

Show total sales by division.

Show January sales of bedroom sets to Milford Furniture.

Subqueries: The following are examples of database queries that require subquery constructs which appear as nested WHERE clauses in SQL:

Show customers that have children under age 10 and do not have a college fund.

Show orders that have more than 2 line items on backorder.

HAVING: The following are examples of database queries that require the HAVING construct:

Show ytd expenses by employee for divisions that have total ytd expenses over 15,000,000.

Show the name and manager of salesmen that have total outstanding receivable of more than \$100,000.

CREATE VIEW: The following are examples of database queries that require the CREATE VIEW syntax:

Show ytd sales by customer with percent of total.

What percent of my salesmen have total ytd sales under \$25,000?

UNION: The following are examples of database queries that require the UNION construct:

Show ytd sales for Connecticut salesmen compared to New York salesmen sorted by product name.

Show Q1 sales compared to last year Q1 sales sorted by salesman.

Thus, common information needs require complex SQL that is likely to be far beyond the understanding of the business people that need this information.

3

A greater problem than the complexity of SQL is that syntactically correct queries often produce wrong answers. SQL is a context-free language, one that can be fully described by a backus normal form (BNF), or context-free, grammar. However, learning the syntax of the language is not sufficient because many syntactically correct SQL statements produce semantically incorrect answers. This problem is illustrated by some examples using the database that has the tables shown in FIGS. 1A–G. If the user queries the database with the following SQL query:

```
(3) SELECT CUSTOMERS.NAME, SUM(ORDERS.ORDER_DOLLARS),
      SUM(LINE_ITEMS.QTY_ORDERED)
FROM CUSTOMERS, ORDERS, LINE_ITEMS
WHERE CUSTOMERS.CUSTOMER# = ORDERS.CUSTOMER#
AND ORDERS.ORDER# = LINE_ITEMS.ORDER#
```

The following results are produced:

NAME	SUM(ORDER_DOLLARS)	SUM(QTY_ORDERED)
1 American Butcher Block	119284	22
2 Barn Door Furniture	623585	52
3 Bond Dinettes	51470	19
4 Carroll Cut-Rate	53375	29
5 Milford Furniture	756960	48
6 Porch and Patio	1113400	89
7 Railroad Salvage	85470	28
8 Sheffield Showrooms	101245	26
9 Vista Designs	61790	25

The second column of this report appears to show the total order amount for each customer. However, the numbers are incorrect. In contrast, the following query

```
(4) SELECT CUSTOMERS.NAME, SUM(ORDERS.ORDER_DOLLARS)
FROM CUSTOMERS, ORDERS
WHERE CUSTOMERS.CUSTOMER# = ORDERS.CUSTOMER#
```

produces the correct result:

NAME	SUM(ORDER_DOLLARS)
1 American Butcher Block	83169
2 Barn Door Furniture	129525
3 Bond Dinettes	51470
4 Carroll Cut-Rate	53375
5 Milford Furniture	111240
6 Porch and Patio	222680
7 Railroad Salvage	85470
8 Sheffield Showrooms	101245
9 Vista Designs	61790

Both SQL queries are syntactically correct, but only the second produces correct numbers for the total order dollars. The problem arises from the fact that before performing the selection and totaling functions, the SQL processor performs a cross-product join on all the tables in the query. In the first query above, three tables are used: Customer (a list of customers with customer data); Order (a list of orders with dollar amounts); and Line_Item (a list of the individual line items on the orders). Since the Order table has the total dollars and there are multiple line items for each order, the joining scheme of the SQL processor creates a separate record containing the total dollars for an order for each instance of a line item. When totaled by Customer, this can

4

produce an incorrect result. When the Line_Item table is not included in the query, the proper result is obtained. Unless the users understand the manner in which the database is designed and the way in which SQL performs its query operations, they cannot be certain that this type of error in the result will or will not occur. Whenever a query may utilize more than two tables, this type of error is possible.

Most information systems users would be reluctant to use a database query tool that could produce two different sets of results for what to them is the same information require-

ment (i.e. total order dollars for each customer). Virtually every known database query tool suffers from this shortcoming.

A more formal statement of this problem is that the set of acceptable SQL statements for an information system is much smaller than the set of sentences in SQL. This smaller set of sentences is almost certainly not definable as a context-tree grammar.

Point-And-Shoot Query Tools

Most SQL generator products are “point-and-shoot” query tools. This class of products eliminates the need for users to enter SQL statements directly by offering users a series of point-and-shoot menu choices. In response to the user choices, point-and-shoot query tools create SQL statements, execute them, and present the results to the user,

appearing to hide the complexities of SQL from the user. Examples of this class of product include Microsoft’s Access, Gupta’s Quest, Borland’s Paradox, and Oracle’s Data Query.

Although such products shield users from SQL syntax, they either limit users to simple SQL queries or require users to understand the theory behind complex SQL constructs. Moreover, because they target the context-free SQL grammar discussed above, it is easy and common for users to get incorrect answers. A point-and-shoot query tool is illustrated below with several examples showing a generic interface similar to several popular query tools representative of this genre. The screen of FIG. 2A appears after the user has chosen the customer table of FIG. 1A out of a pick list. This screen shows the table chosen and three other boxes, one for each of the SELECT, WHERE, and ORDER BY clauses of the SQL statement. If the user selects either the “Fields” box or the “Sort Order” box, a list of the fields in the customer table appears. The user makes choices to fill in the “Fields” and “Sort Order” boxes. In this example, the user chooses to display the NAME, STATE, and BALANCE fields, and to sort by NAME and STATE. This produces the screen of FIG. 2B.

At any time, the user can choose to view the SQL statement that is being created as shown in FIG. 2C. There is a one-to-one correspondence between user choices and the

SQL being generated. To fill in the WHERE clause of the SQL statement being compiled, the user chooses the "Conditions" box and fills in the dialog box of FIG. 2D to enter a condition. This produces the completed query design shown in FIG. 2E. The user then chooses the "OK" button to run the query and see the results shown in FIG. 2F.

For queries that involve only a simple SELECT, WHERE, and ORDERBY statement for a single table, a user can readily create and execute SQL statements without knowing SQL or even viewing the SQL that is created.

Unfortunately, only a small proportion of user queries are this simple. Most database queries involve more complex SQL. To illustrate this point, consider a user who wishes to see the same information as in the above example, but to limit the data retrieved to customers of salespersons with total outstanding balance of all the salesperson's customers greater than \$80,000. If the user realizes that this query requires two additional SQL clauses (a GROUP BY clause and a HAVING clause) the query (shown in FIG. 2G) can be readily constructed. However, few users are sufficiently familiar with SQL to do so.

Most point-and-shoot query tools cannot handle other complex SQL constructs such as subqueries. CREATE VIEW and UNION. They offer no way (other than entering SQL statements directly) for the user to create these other constructs. Those products that do offer a way to generate other complex constructs require the user to press a "Subquery" or "UNION" or "CREATE VIEW" button. Of course, only users familiar enough with the relational calculus to know how to break a query up into a subquery or use another complex SQL construct would know enough to press the right buttons.

Additional complexity is introduced when data must be retrieved from more than one table. As shown in FIG. 2H, the user may be required to specify how to join the tables together. The typical user query will involve at least three tables. Problems that can arise in specifying joins include:

- the columns used to join tables may not have the same name;
- the appropriate join between two tables may involve multiple columns;
- there may be alternative ways of joining two tables; and
- there may not be a way of directly joining two tables, thereby requiring joins through other tables not otherwise used in the query.

In summary, point-and-shoot query tools shield users from syntactic errors, but still require users to understand SQL theory. The other critical limitation of point-and-shoot menu products is that they target the context-free SQL language discussed above. A user seeking total order dollars could as easily generate incorrect SQL statement (3) as correct SQL statement (4) above. Thus, these products generate syntactically correct SQL, but not necessarily semantically correct SQL. Only a user that understands the relational calculus can be assured of making choices that generate both syntactically correct and semantically correct SQL. However, most information system users do not know relational calculus. Moreover, when queries require joins, there are numerous way of making errors that also produce results that have the correct format, but the wrong answer.

Natural Language Query Tools

Natural language products use a different approach to shielding users from the complexities of SQL. Natural language products allow a user to enter a request for

information in conversational English or some other natural language. The natural language product uses one mechanism to deduce the meaning of the input, a second mechanism to locate database elements that correspond to the meaning of the input, and a third mechanism to generate SQL.

Examples of natural language products include Natural Language from Natural Language Inc. and EasyTalk from Intelligent Business Systems (described in U.S. Pat. No. 5,197,005 to Shwartz et al.).

FIG. 3A shows a sample screen for a natural language query system which shows a user query, the answer, another query requesting the SQL, and the SQL.

The sequence of interaction is:

- (1) The user types in a free-form English query <"What were the 5 most common defects last month?").
- (2) The software paraphrases the query so that the user can verify its correctness "What were the 5 defects that occurred the most in June, 1991?").
- (3) If there are spelling errors or if the user query contains ambiguities, the software interacts with the user to clarify the query. (not needed in above example).
- (4) The software displays the results.

The attraction of a natural language query tool is that users can express their requests for information in their own words. However, they suffer from several shortcomings. First, they only answer correctly a fraction of the queries a user enters. In some cases, the paraphrase is sufficient to help the user reformulate the query; however, users can become frustrated seeking a formulation that the system will accept. Second, they are difficult to install, often requiring months of effort per application and often requiring consulting services from the natural language vendor. One of the biggest installation barriers is that a huge number of synonyms and other linguistic constructs must be entered in order to achieve anything close to free-form input.

As a compromise, many natural language vendors recommend that, during installation, specific queries are coded and made available to users via question lists. For example, FIG. 3B shows a simple screen containing a list of pre-defined queries. Users can choose to run queries directly from the list or make minor modifications to the query before running it. Of course, the more they change a query, the more likely it is that the natural language system will not understand the query.

To illustrate the operation natural language products, the architecture of the natural language system described in Shwartz, et al. is used as an example. The system architecture is shown in FIG. 4. The Meaning Representation is the focus of Shwartz et al. The Meaning Representation of a query is designed to hold the meaning of the user query, independent of the words (and language) used to phrase the query and independent of the structure of the database.

The same Meaning Representation should be produced whether the user says "Show total ytd sales for each customer?", "What were the total sales to each of my client's this year to date?." or "Montrez les vendes . . ." (French). Moreover, the same Meaning Representation should be produced whether: (1) there is a field that holds ytd sales in a customer table in the database; (2) each individual order must be searched, sorted, and totaled to compute the ytd sales for each customer; or (3) ytd sales by customer is simply not available in the database.

The primary rationale for this architecture is that it provides a many-to-one mapping of alternative user queries onto a single canonical form. Many fewer inference rules are

then needed to process the canonical form than would be needed to process user queries at the lexical level. This topic is addressed in more detail in Shwartz, "Applied Natural Language", 1987.

The NLI (Natural Language Interface) is responsible for converting the natural language query into a Meaning Representation. The Query Analyzer itself contains processes for syntactic and semantic analysis of the query, spelling correction, pronominal reference, ellipsis resolution, ambiguity resolution, processing of nominal data, resolution of date and time references, the ability to engage the user in clarification dialogues and numerous other functions. Once an initial Meaning Representation is produced, the Context Expert System analyzes it and fills in pronominal referents, intersentential referents, and resolves other elliptical elements of the query. See S. Shwartz, for a more detailed discussion of this topic.

The Meaning Representation for the query "Show ytd sales dollars sorted by salesrep and customer" would be:

```
SALES: TIME (YTD), DOLLARS, TOTAL
SALESMAN: SORT(1)
CUSTOMER: SORT(2)
```

Again, this meaning representation is independent of the actual database structure. The Database Expert takes this meaning representation, analyzes the actual database structure, locates the database elements that best match the meaning representation, and creates a Retrieval Specification. For a database that has a table, CUSTOMERS, that contains a column holding the total ytd sales dollars, YTD_SALESS\$, the Retrieval Specification would be:

```
CUSTOMER.YTD_SALES$:
SALESMAN.NAME: SORT(1)
CUSTOMERS.NAME: SORT(2)
```

The Retrieval Specification would be different if the YTD_SALESS\$ column was in a different table or if the figure had to be computed from the detailed order records.

The functions of the NLI (and Context Expert) and DBES are necessary solely because free-form, as opposed to formal, language input is allowed. If a formal, context-free command language was used rather than free-form natural language, none of the above processing would be required. The Retrieval Specification is equivalent to a formal, context-free command language.

The Navigator uses a standard graph theory algorithm to find the minimal spanning set among the tables referred to in the Retrieval Specification. This defines the join path for the tables. The MQL Generator then constructs a query in a DBMS-independent query language called MQL. The SQL Generator module then translates MQL into the DBMS-specific SQL. All of the expertise required to ensure that only syntactically and semantically valid SQL is produced is necessarily part of the MQL Generator module. It is the responsibility of this module to reject any Retrieval Specifications for which the system could not generate syntactically and semantically valid SQL.

Natural Language Menu Systems

A Natural Language Menu System is a cross between a point-and-shoot query tool and a natural language query tool. A natural language menu system pairs a menu interface with a particular type of natural language processor. Rather than allowing users to input free-form natural language, a context-free grammar is created that defines a formal query language. Rather than inputting queries through a command

interface, however, users generate queries in this formal language one word at a time. The grammar is used to determine all possible first words in the sentence, the user chooses a word from the generated list, and the grammar is then used to generate all possible next words in the sentence. Processing continues until a complete sentence is generated.

A natural language menu system will provide a means of ensuring that the user only generates syntactically valid sentences in the sublanguage. However, it can only guarantee that these sentences will be semantically valid for the class of sublanguages in which all sentences are semantically valid. Another difficulty with this class of tool is that it is computationally inadequate for database query. The computational demands of the necessarily recursive algorithm required to run the grammar are immense. Moreover, if the grammar is sufficient to support subqueries, the grammar would probably have to be a cyclic grammar, adding to the computational burden. Finally, the notion of restricting users to a linear sequence of choices is incompatible with modern graphical user interface conventions. That is, users of this type of interface for database query would object to being forced to start with the first word of a query and continue sequentially until the last word of a query. They need to be able to add words in the middle of a query without having to back up and need to be able to enter clauses in different orders.

SUMMARY OF THE INVENTION

The drawbacks of the prior art are overcome by the system and method of the present invention, which hides the complexity of SQL from the user without limiting the range of information that can be retrieved. Most importantly, incorrect results are avoided.

In accordance with the principles of the invention, a Query Assistant is provided that permits the user to enter only queries that are both syntactically and semantically valid (and that can be processed by the SQL Generator to produce semantically valid SQL). The user is never asked to rephrase a query entered through the Query Assistant. Through the use of dialog boxes, a user enters a query in an intermediate English-like language which is i) easily understood by the user. A Query Expert system monitors the query as it is being built, and using information about the structure of the database, it prevents the user from building semantically incorrect queries by disallowing choices in the dialog boxes which would create incorrect queries. An SQL Generator is also provided which uses a set of transformations and pattern substitutions to convert the intermediate language into a syntactically and semantically correct SQL query.

The intermediate language can represent complex SQL queries while at the same time being easy to understand. The intermediate language is also designed to be easily converted into SQL queries. In addition to the Query Assistant and the SQL Generator an administrative facility is provided which allows an administrator to add a conceptual layer to the underlying database making it easier for the user to query the database. This conceptual layer may contain alternate names for columns and tables, paths specifying standard and complex joins, definitions for virtual tables and columns, and limitations on user access.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGS. 1A to 1G are tables of a sample database used in the examples in the specification.

FIGS. 2A to 2H are typical screen displays for a point-and-shoot query tool.

FIG. 3A is a typical screen display for a natural language query tool.

FIG. 3B is a typical screen display for a natural language database query tool with predefined queries. 5

FIG. 4 is a block diagram of the high level architecture of a natural language query tool.

FIG. 5 is a block diagram of the high level architecture of the invention. 10

FIG. 6 is a graphic depiction of the tables in FIGS. 1A-1G and their relationships.

FIG. 7 is a block diagram of the Query Assistant.

FIG. 8 is a flow chart of the flow of control of the Query Assistant User Interface. 15

FIG. 9 is a depiction of the initial screen of the user interface.

FIGS. 10A to 10G are depictions of dialog boxes used to interact with the user to build a query using the Query Assistant. 20

FIGS. 11A and 11B are a flow chart depicting the flow of control of the SQL Generator. 25

DETAILED DESCRIPTION

I. OVERVIEW 30

FIG. 5 shows a high level block diagram of an intelligent query system that embodies the principles of the invention. It is composed of two parts, the Query System 1 and Conceptual Layer 2. Conceptual Layer 2 is composed of information derived from database 3, including table and column information, and information entered by an administrator to provide more intuitive access to the user. Query System 1 uses the information from Conceptual Layer 2 as well as general knowledge about SQL and database querying to limit the user in building queries to only those queries which will produce semantically correct results. 35

Query System 1 is further composed of two main components: Query Assistant 10 and the SQL Generator 20. Users create queries using the menu-based Query Assistant 10 which generates statements in an intermediate query language that take the form of easy to understand sentences. SQL Generator 20 transforms the intermediate language into a target language (in the illustrated embodiment, SQL). To fulfill the requirement that a user never be asked to rephrase (or reconstruct) a query, the expertise concerning what is and what is not a valid SQL query is placed in Query Assistant 10. 40

SQL Generator 20 does not contain this expertise. Although users can pose queries directly to SQL Generator 20, there is no assurance that semantically valid SQL will be produced. It is logically possible to put some of this expertise into SQL Generator 20. However, to assure users that only valid SQL would be generated would require natural language capabilities not presently available. 45

II. CONCEPTUAL LAYER

A database may be composed of one or more tables each of which has one or more columns, and one or more rows. For example: 65

Name	State	Zip
John	VA	22204
Mary	DC	20013
Pat	MD	24312

In this small example, there is one table containing three columns, and three rows. The top row is the column names and is not considered a row in the database table. The term 'row' is interchangeable with the term 'record' also often used in database applications, and 'column' is interchangeable with the term 'field'. The primary distinction between the two sets of terms is that row and column are often used when the data is viewed in a list or spreadsheet table style view, and the terms field and record are used when the data is viewed one record at a time in a form style view.

A database may have more than one table. To this simple example, another table can be added called Purchases which lists purchases made by each Person.

Name	Product	Quantity
John	apple	6
John	orange	4
Mary	kiwi	2
Pat	orange	12
Pat	kiwi	5
Pat	mango	10

Stored along with a database is some structure information about the tables contained within the database. This includes the name of the tables, if there are more than one, the names of the columns of the tables, and the structure of the data stored in the columns. In the above example, the first table is titled (for purposes of this example) "Person" and the second table "Purchases." In the Person table there are three columns: "Name", containing alphanumeric data; "State" containing two characters of alphanumeric data; and "Zip", which may be stored as five characters of alphanumeric data or as numeric data. In the Purchases table there are also three columns: "Name", containing alphanumeric data; Product, containing alphanumeric data; and Quantity, containing numeric data. 50

Also, stored along with the database are the primary keys for each of the tables. In most database systems each row must be uniquely identifiable. One or more columns together create the primary key which when the contents of those columns are combined uniquely identify each row in the table. In the Person table above, the column Name is unique in each row and Name could be the primary key column. However, in the Purchases table, "Name" does not uniquely identify, each row since there are multiple Johns and Pats. In that table, both the "Name" and "Product" columns together uniquely identify each row and together form the primary key. 55

In the above example, there is an implied relationship between the two tables based on the common column title "Name". To determine how many oranges Virginians buy, a user could look in the Person table and find that John is the only Virginian and then go to the Purchases table to find that he bought four oranges. Some database managers explicitly store information about these relationships, including situations where the relationship is between two columns with different names. 60

The example above is very simple, and a user could readily understand what information the database held and

how it was related. However, real world problems are not that simple. Though still rather simplistic compared to the complexity of many real world problems, the example database represented in FIGS. 1A-G begins to show how difficult it might be for a user to understand what is contained in the database and how to draft a meaningful query. This is particularly difficult if the real meaning of the database is contrary to the naming conventions used when building it. For example, the Customer Table of FIG. 1A is not related directly to the Product Table of FIG. 1B even though they both have columns entitled NAME. However, they are related via the path CUSTOMER=>ORDER=>LINE_ITEM<=>PRODUCT (i.e. a customer has orders, an order has line_items, and each line_item has a product).

To shield the user from the complexity of the underlying database, a knowledgeable administrator may define a conceptual layer, which in addition to the basic database structure of table names and keys, and column names and types, also may include: foreign keys, name substitutions, table and column descriptions, hidden tables and columns, virtual tables, virtual columns, join path definitions and non-equijoins.

All of the forms of information that make up the conceptual layer can be stored alongside the database as delimited items in simple text files, in a database structure of their own, in a more compact compiled format, or other similar type of information storage. When a database is specified to be queried Query Assistant 1 has access to the basic structure of the database, which the database manager provides, to aid the user in formulating semantically correct queries. Optionally the user may choose to include the extended set of conceptual information which Query Assistant 1 can then use to provide a more intuitive query tool for the end user.

The conceptual layer information is stored internally in a set of symbol tables during operation. Query Assistant 10 uses this information to provide the user a set of choices conforming to the environment specified by the Administrator, and SQL Generator 20 uses the information, through a series of transformations, to generate the SQL query.

1. Foreign keys

A table's foreign keys define how they relate to other tables. Two tables are joined by mapping the foreign key of one table to the primary key of the second table. A foreign key is defined by the columns within the first table that make up the foreign key, and the name of the second table which can join with the first table by matching its primary key with the first table's foreign key.

In the example above, the Purchases table with the foreign key "Name" can join the Person table with the primary key "Name."

	Purchases		Person	
foreign key	Name	<- - - - -	Name	primary key
	Product		State	
	Quantity		Zip	

If the two tables are joined based on their foreign and primary keys, the following new table is created:

Name	State	Zip	Product	Quantity
John	VA	22204	apple	6
John	VA	22204	orange	4
Mary	DC	20013	kiwi	2
Pat	DC	20013	orange	12
Pat	MD	24312	kiwi	5
Pat	MD	24312	mango	10

Rows from each of the two tables with the same value in their respective "Name" column were combined to create this new table. This is referred to as a One-to-Many relationship. For every one Person row there can be many Purchases rows. A relationship can also be One-to-One, which indicates that for every row in one table, there can be only one related row in another table. Both One-to-Many and One-to-One relationships may be optional or required. If optional, then there may not be a related row in a second table. In the illustrated embodiment, along with the foreign key in the conceptual layer an administrator may designate which of these four types of relationships (i.e. one-to-many, one-to-many optional, one-to-one, one-to-one optional) exists between the tables joined by the foreign key. In some database management systems, it is possible for a table to have multiple primary keys, in which case, the administrator must also designate to which primary key the foreign key is to be joined. FIG. 6 is a graphical representation of the relationships between the tables in FIGS. 1A-1G. Each line represents a relationship between two tables and an arrow at the end of the line indicates a one-to-many optional relationship. The end with the arrow is the "many" end of the relationship. For example, between the SALESPeOPLE and CUSTOMERS tables there is a one-to-many relationship with multiple customers handled by each salesperson. Using the example tables of FIGS. 1A-G and the relationships illustrated in FIG. 6 a definition in the conceptual layer for the foreign keys would be:

Table	Foreign Key	Second Table	Relationship Type	Primary Key
CUSTOMERS	SALESPERSON#	SALESPeOPLE	1-to-many opt.	1
LINE_ITEMS	ORDER#	ORDERS	1-to-many opt.	1
LINE_ITEMS	PRODUCT#	PRODUCTS	1-to-many opt.	1
ORDERS	CUSTOMER#	CUSTOMERS	1-to-many opt.	1
ORDERS	SALESPERSON#	SALESPeOPLE	1-to-many opt.	1
PRODUCTS	GROUP_ID	CODES	1-to-many opt.	1
PRODUCTS	TYPE_ID	CODES	1-to-many opt.	1
PRODUCTS	ALT-VENDOR#	VENDORS	1-to-many opt.	1
PRODUCTS	VENDOR#	VENDORS	1-to-many opt.	1

The above chart represents the foreign key data which may be present in the conceptual layer. The chart is described by way of an example. According to the first row below the headings of the chart, there is a table CUSTOMERS with a foreign key defined by the column SALESPERSON#. This foreign key relates to the first primary key (note the 1 in the primary key column) of the SALESPEOPLE table by a one-to-many optional relationship. In other words, for every row in the SALESPEOPLE table there are zero or more related rows in the CUSTOMERS table according to SALESPERSON#.

The (2) next to the lines between CODES and PRODUCTS and VENDORS and PRODUCTS in FIG. 6 indicates that there are actually two one-to-many relationships between those tables. This can be seen in the foreign key chart above. There are two sets of foreign keys linking PRODUCTS to VENDORS and PRODUCTS to CODES.

2. Name substitution

Name substitution is the process by which a table's or columns name as defined in the database structure is substituted with another more intuitive name for presenting to the user. This is particularly useful when dealing with a database management system which only provides limited naming capabilities (i.e. only one word). This process serves two primary purposes. First it allows an administrator to make the information available to the user in a given database more readily understandable, and second, it can be used to distinguish columns from different tables which have the same name, but are not related (i.e. column "Name" in the CUSTOMERS table (FIG. 1A) and column "Name" in the PRODUCTS table (FIG. 1B)). In addition, it is possible to provide plural and singular names for tables.

For example, using the table in FIG. 1A it is possible to define the singular and plural names for the table as CUSTOMER and CUSTOMERS, and to rename the fields to provide more guidance to the user and distinguish conflicts as follows:

Column	New Name
CUSTOMER#	Customer Number
NAME	Customer Name
CITY	Customer City
STATE	Customer State
ZIP CODE	Customer Zip Code
SALESPERSON#	Salesperson Number
CREDIT_LIMIT	Credit Limit
BALANCE	Customer Balance

3. Table and column descriptions

Descriptions of the various tables and columns can be included in the conceptual layer to provide better understanding for the user. For example, The CUSTOMER table may have an associated description of "Records containing address and credit information about our customers." Then when the user highlights or otherwise selects the CUSTOMER table while building a query, the description will appear on a status line of the user interface or something similar. The same type of information can be stored for each of the columns which display when the columns are highlighted or otherwise selected for possible use in a query.

4. Hidden tables and columns

In the design of a database, it is often necessary to add columns that are important in relating the database tables but that are not used by the end user who will be forming queries on the database. For example, the SALESPERSON# column in the tables of FIGS. 1A, 1D, and 1E are not important to

the end user, who need only know that Paul Williams is the salesperson for American Butcher Block and Barn Door Furniture. The end user need not know that his internal number for use in easily relating the tables is 1. Accordingly, as part of the conceptual layer, an administrator can hide certain columns so that the user cannot attempt to display them or use them in formulating a query. When a column is hidden, it can still be used to join with another table. This same techniques can be used to prevent end users from displaying private or protected data, and to shield the user from the details of the database which might be confusing and unnecessary.

In some cases, there are tables which are used to link other tables together or are unimportant to the end user. Therefore, as part of the conceptual layer, an administrator can also hide certain tables so that the user cannot attempt to display them or use them in formulating a query. A hidden table, however, can still be used by the query system to perform the actual query—it is just a layer of detail hidden from the end user. In addition, as described in more detail below, when virtual column and table techniques are used, columns may be included, for display to the end user, as elements of other tables. By hiding the original columns and/or tables, the administrator can, in effect, move a column from one table to another.

When designating elements that an end user can include in generating a semantically correct query, the Query Assistant will not designate the hidden tables and columns.

5. Virtual tables

Virtual tables are constructs that appear to the user as separate database tables. They are defined by the Administrator as a subset of an existing database table limited to rows that meet a specific condition. Initially, the virtual table has all the fields within the actual table upon which it is based, but it only contains a subset of the records. For example, the Administrator could define the virtual table BACKORDERS which includes all the records from the ORDERS table where the Status field contains the character "B". Then, when a user queries the BACKORDERS table, the user would only have access to those orders with backorder status.

The Administrator defines the virtual table according to a condition clause of the target language (in this case, SQL). In the above example, the table BACKORDERS would be defined as "ORDERS WHERE ORDERS.STATUS='B'". In this way, the SQL generation portion of the virtual table is accomplished by a simple text replacement. Similarly, the condition could be stored in an internal representation equivalent to the SQL or other target language condition.

It is possible to define a virtual table without the condition clause. In that case, a duplicate of the table on which it is based is used. However, the Administrator can hide columns and add virtual columns to the virtual table to give it distinct characteristics from the table upon which it is based. For example, a single table could be split in two for use by the end user by creating a virtual table based on the original and then hiding half of the columns in the original table, and half of the columns in the virtual table.

6. Virtual columns

The conceptual layer may also contain definitions for virtual columns. Virtual columns are new columns which appear to the user to be actual columns of a table. Instead of containing data, the values they contain are computed when a query is executed. It is possible to add fields which perform calculations or which add columns from other tables. There are six primary uses for virtual columns:

(1) Moving/copying items from one table to another. Often due to various database design factors, there are more tables in the physical database than in the user's conceptual model. In the example in FIGS. 1A-1G, an end user might not consider orders as being multiple rows in multiple tables as is required with the LINE-ITEM, ORDER distinction of the example. The Administrator can specify in the conceptual layer that the user should see the field of LINE_ITEM (i.e. product, qty_ordered, qty_backordered, warehouse, etc.) as being part of the order table. Columns can be moved from one table to another with only one limitation that a primary key/foreign key relationship exist between the table the column is being moved from and the table the column is being moved to. These relationships are indicated in FIG. 6 as the lines with the arrows.

(2) Creating a virtual column defined by a computation. Virtual columns can be created by an administrator which are computations on existing columns in a table. For example, we could add a TURNAROUND column to the ORDERS table of FIG. 1F defined as "SHIP_DATE—ORDER_DATE". This would allow a user to easily create a query which asked to show what the turnaround time was for orders without having to actually specify the calculation in the query.

(3) Creating a virtual column defined using DBMS specific functions. The target language of the Data Base Management System (DBMS) being used may have specific formatting or other data manipulation operations which could be used to present information to the user in a particular way. Even though the SQL Generator is designed to produce SQL, implementations of SQL differ from DBMS to DBMS.

By the addition of a Lookup function, explicit joins can be defined in order to add columns from other tables or instances of the same table. The Lookup function can be used to define a virtual column and takes as parameters: a foreign key column (which is a foreign key column for the table where the virtual column is being placed, or a base table if it is a virtual table); and a reference column (which is a column in the table that the foreign key references). The remaining three uses employ this function to avoid complexities which are not addressed by current query systems.

(4) Eliminating complexity caused by alternate foreign keys. Tables often have multiple ways of joining, represented by alternate foreign keys. This can be a source of confusion for the user. For example, using the tables of FIGS. 1A-1G, the PRODUCT table (FIG. 1B) has two foreign keys, VENDOR# and ALT_VENDOR#. To aid the user in accessing the database, the Administrator would define virtual columns within the PRODUCT table for Vendor Name and Alternate Vendor Name, so that it appears to the user that they can easily find the vendor's names without resorting to looking in multiple tables for the information. However, this would generally confuse a query system because there are two foreign keys for use in joining the tables. By using the Lookup function for each of the Vendor Name and Alternate Vendor Name virtual columns, different foreign key joins can be specified for each of the columns, giving the user both the vendor and alternate vendor names. The definition of the virtual columns would be:

Table	Virtual Column	Type	Definition
PRODUCTS	VNAME	A	Lookup(PRODUCT.VENDOR#, VENDOR.NAME)
PRODUCTS	AVNAME	A	Lookup(PRODUCT.ALT_VENDOR#, VENDOR.NAME)

(5) Eliminating complexity caused by code tables

This is a special case of the alternate foreign keys case (4) above. Many databases have a code table whose purpose is to store the name and other information about each of several codes. The tables themselves only contain the code identifications. If a single table has multiple code columns which use the same table for information about the codes there is a potential for the alternate foreign key problem. The user instead of asking for products with "status_id='007' and type_id='002'" would prefer to ask for products with "status='open' and type='wholesale'". Using the Lookup scheme, two virtual columns for the textual status and type can be added to the products table.

(6) Eliminating complexity caused by self-referencing tables

For example, each employee in an employee table may have a manager who himself is an employee—the manager column refers back to the employee table. Using the Lookup function, virtual columns for each employees managers name, salary, etc. can be added to the employee table. To perform the actual query, a self join will be required. Using the employee table example for a table called EMP and a column MGR being a foreign key relating to the EMP table, the virtual column definitions would be:

Table	Virtual Column	Type	Definition
EMP	MNAME	A	Lookup(EMP.MGR, EMPLNAME)
EMP	MSAL	N	Lookup(EMP.MGR, EMPSAL)

7. Join path definitions

In certain circumstances it is possible to join two tables by multiple paths. For example, in the tables shown in FIGS. 1A-1G, the SALESPEOPLE table can be joined with the ORDERS table by two different paths. This is easiest to see in FIG. 6. By following the direction of the arrow, SALESPEOPLE are connected directly to ORDERS or they can be connected to ORDERS via CUSTOMERS. In a query of the database, the manner in which the join is performed yields different results with different meanings.

(1) If SALESPEOPLE is joined directly with the ORDERS table, the result will indicate which salesperson actually processed the order.

(2) If SALESPEOPLE is joined to ORDERS via CUSTOMERS, the result will indicate the current salesperson for the customer on the order.

The Administrator can add to the conceptual layer a set of join paths for each pair of tables if desired. If multiple join paths are defined, a textual description of each join is also included. When the query is being generated, the system will prompt the user for which type of join the user prefers in an easy to understand manner. In the above example, when a user creates a query which joins the SALESPEOPLE and ORDERS tables the Query Assistant will generate the following dialog box:

Please clarify your query by indicating which of the following choices best characterizes the data you wish displayed:

1. Use the salesperson that actually processed the order.
2. Use the current salesperson for the customers on the order where the text in the choices is defined by the Administrator and correlates with the join path taken and used by the system.

If no join paths are defined for a given pair of tables, the shortest path is used by the system when creating a query. This can be determined by using a minimal spanning tree algorithm or similar techniques commonly known in the art.

8. Non-equi joins

The table joins discussed in the preceding examples have been equi joins. They are called equi joins because the two tables are combined or joined together based on the equality of the value in a column of each table (i.e. SALESPERSON#=SALESPERSON#, however, the column names need not be the same). In the illustrated embodiment, the Administrator can also provide in the conceptual layer definitions for non-equi join relationships between tables which will join rows from two different tables when a particular condition is met. For example, another table ORDTYPE could be added to the example of FIG. 1A-1G that provides different classifications for orders of dollar amounts in different ranges:

Low	High	Type
0	10000	Small
10000	50000	Medium
50000	1000000	Huge

Using a non-equi join, a record from the ORDERS table could be joined with a record from the ORDTYPE table when ORDER_DOLLARS<=LOW (from ORDTYPE table)<=HIGH. Instead of an equality relationship, there is a relationship based on a range of values.

The Administrator codes the non-equi join as an SQL condition. For the above example, the Administrator would specify that ORDERS should be joined with ORDTYPE "Where ORDERS.ORDER_DOLLARS>=ORDTYPE.Low AND ORDERS.ORDER_DOLLARS<ORDTYPE.High". It will also become evident that the procedures for specifying non-equi joins could be implemented in a manner similar to Query Assistant 10 to ensure correctness. The condition is stored in SQL so as to be directly used during the conversion from the intermediate language to the target language SQL. However, it is obvious to an artisan that the condition could be stored in an internal representation equivalent to the SQL condition or other target language.

III. QUERY ASSISTANT

FIG. 7 shows a block diagram of the Query Assistant 10. It has two components: The Query Assistant User Interface (QAUI) 11 and the Query Assistant Expert System (QAES) 12. QAUI 11 performs the functions of displaying the current state of the query to the user and providing a set of choice to the user for constructing a semantically correct query.

B. Query Assistant User Interface (QAUI)

QAUI 11 interacts with the user and QAES 12 to formulate a query in the intermediate language. Through the interface, the user initiates a query, formulates a query, runs a query, and views the results. FIG. 8 shows the basic flow of control of QAUI 11. When a user initiates a query at step

50, QAUI 11 calls QAES 12 to initialize the blackboard at step 52, then, in steps 54-58, continuously presents to the user a set of choices based on the current context in the query, as limited by the rules in QAES 12. After the user makes a selection at step 60, the system QAUI 11 determines whether the user selected to clear the query (step 62), and, if not, whether the user selected to run or cancel the query (step 66), the blackboard is updated at step 68 and an intermediate language representation is updated at step 70. This continues until the user either clears the query (at step 62, in which case the intermediate language representation is cleared at step 64) or cancels or runs the query (at step 66), in which case the appropriate action is taken.

FIG. 9 shows the initial screen 110 presented by QAUI 11 to the user. Initial screen 110 has four areas: User Query window 112 (where the query in the intermediate language is built up); SQL Query window 114 (where the SQL equivalent of the User Query is displayed after the User Query is formulated); Result window 116 (where the result is displayed after the SQL Query is applied to the Database Management System); and menu bar 118 (providing access to a set of drop down menus that allow the user to select databases, select conceptual layers, interface to report generators, save and load queries, clear the current query, run the current query, set system defaults, etc.).

The user can invoke Query Assistant 10 by selecting it from a drop down menu under the Query heading. This brings up a query selection menu listing the various types of queries Query Assistant 10 can handle. This is based on the range of queries the intermediate language is capable of handling and the query generation routines built into Query Assistant 10. Optionally, the administrator can limit the types of queries which the user can use on a given database by so specifying in the conceptual layer. If the user is limited to a single kind of query, then the query list is bypassed. In the illustrated embodiment, the query selection menu includes:

- Show . . .
- What percent of . . . have . . .
- Compare . . . against . . .
- Show . . . as a percentage of . . .

The "Show . . ." query, covers approximately 99% of all queries and is the basic command to view certain columns or calculations thereon. The other queries are special types for percentage and comparison calculations. The type of result desired is obvious from the query excerpt in the display. This is in part due to the design of the intermediate language to make difficult query concepts easy to understand.

When the user selects the "Show . . ." query, the Create Show Clause dialog box 120 (shown in FIG. 10A) is displayed. This is the primary means for interaction between the user and the Query Assistant. For purposes of illustration in the figures, items that can be selected by the user are in bold face, and items which cannot be selected are in italics. Other ways of distinguishing selectable items include: 'graying out' unselectable items by displaying them in lighter shades or different color: or inhibiting the display of non-selectable items so that only selectable items are displayed. The selections status (whether or not an item can be selected) is specified either by QAUI 11 or by a call to QAES 12. Procedural rules are governed by the QAUI and expert system rules which define the selectable tables, columns, and operations are governed by QAES 12. Procedural rules include, but are not limited to:

1. conditions or sort order on a query, cannot be specified until something for display has been specified:

2. an individual column cannot be selected until it is highlighted;

3. a query cannot be run before something has been entered; and

4. items cannot be deleted until there is at least one item to delete.

The section designation window **121** of Create Show Clause dialog box **120** allows the user to designate what section or clause of the query is being entered. Window **121** includes Show, For, Sorted By, and With % of Total sections **121a-d**, respectively. The user need not designate the sections in any specific order except that at least one column must be designated to be shown before the other clauses can be specified. However, the user may move back and forth between the sections. For example, a user may specify one column to show, then fill in For section **121b**, return to designate more columns to be shown, then designate a Sorted By column by selecting Sorted By section **121c**, etc.

The control section **122** of dialog box **120** includes a set of selection buttons **122a-d** by which the user can direct the system to run the query, clear the query, cancel creating a query, and create a column to show a computation. Computations are discussed in more detail below.

In item selection window **123**, the user can select tables and columns as specified in the conceptual layer, including any virtual tables or columns and any name substitutions. Any hidden tables or columns are hidden. Item selection window **123** includes table selection window **124**, column selection window **125**, description box **126**, and Select and Select All buttons **127a** and **127b**. For purposes of example, FIG. **10A** uses the tables of FIGS. **1A-1G** with several tables hidden, the columns renamed, and a generated column "THE COUNT OF CUSTOMERS" defined in the virtual layer as a Count on the table CUSTOMERS. By moving the highlighted bar from table to table in table selection window **124**, the list of available columns for the highlighted table is displayed in column selection window **125**. The Select and Select All buttons **127a**, **127b** allow the user to select a column to show. Description box **126** shows a description for the highlighted table or column if a description is present in the conceptual layer.

The user can modify selected columns in the column modification window **128**. Columns selected for the Show clause are listed in display window **129**. The user can apply aggregate computations (i.e. count, total, average, minimum, and maximum) to the selected columns or unselect them via aggregate buttons **130a-h**.

After the user makes a selection (of a table in table selection window **124** or a column in column selection window **125**), QAUI **11** communicates with QAES **12** to update blackboard **13** and to request a new set of allowable selections. In addition, User Query window **112** of initial screen **110** is updated to reflect the query at that point in the intermediate language. If the selection made by the user causes certain items to become selectable or nonselectable, dialog box **120** is updated to reflect that. For example, FIG. **10B** shows dialog box **120** after the user has selected the CUSTOMER BALANCE column of the CUSTOMERS table to display and has further selected to modify the column (indicated by the column being shown in display window **129**). In response, QAUI **11** has modified dialog box **120** in several ways. First, aggregate buttons **130a-h** are now selectable. QAES **12** has informed QAUI **11** that these buttons can be selected based on the determination that CUSTOMER BALANCE is numeric and that placing an aggregate on it would not create a semantically incorrect query. Had the user selected CUSTOMER NAME instead,

QAES **12** would only have made Count button **130c** and None button **130h** selectable since the other types of aggregates require a numeric column. Also, For and Sorted By sections **121b**, **121c**, in section designation window **121** are now selectable, as is the Run Query command **122a** in control section **122** since the Show section has something to show. User Query window **112** of initial screen **110** would now contain the string "SHOW CUSTOMER BALANCE".

FIG. **10C** shows the state of dialog box **120** after the user has asked to find the average of CUSTOMER BALANCE (via Average button **130e**) and is preparing to select another column for display. Since the average aggregate has been placed on a numeric column, all the rows will be averaged together. Therefore, no joins to one-to-many tables are allowed and only other numeric columns which can be similarly aggregated can be selected. This has been determined by QAES **12** upon request by QAUI **11** and can be seen in dialog box **120** where all other tables and all non numeric columns have been made nonselectable. Had there been a virtual numeric column from another table present it also would not be selectable since a join is not allowed. If the user selects CREDIT LIMIT, QAUI **11** will be notified by QAES **12** that an aggregate is required and will put up a dialog box requesting which aggregate the user would like to use.

The user may also ask to see results that are actually computed from existing columns. In that case, the user can select Computation button **122d**. This selection causes QAUI **11** to display computation dialog box **135**, shown in FIG. **10D**. Computation dialog box **b** allows the user to build computations of the columns. QAUI **11** requests of QAES **12** which tables, columns and operations are selectable here as well. The state of computation dialog box **b** as shown in FIG. **10D** is as it would be at the start of a new query. However, all non-numeric fields are not selectable since computations must occur on numeric columns. This rule is stored in QAES **12**.

When the user selects Sort By section **121e** of section designation window **121**, QAUI **11** displays Sorted By dialog box **140**, shown in FIG. **10E**. This dialog box is very similar to Create Show Clause dialog box **120**. As with the other dialog boxes, QAUI **11** works with QAES **12** to specify what columns can be selected by the user for use in the "Sort By . . ." section. Note, the generated column THE COUNT OF CUSTOMERS is not selectable since it is actually an aggregate computation that cannot be used to sort the results of the query.

The For section, which is used to place a condition on the result, is more procedural in nature. If the user selects For section **121b** in section designation window **121**, QAUI **11** presents the user with a series of Create For Clause dialog boxes of the form shown in FIG. **10F**, which provide a list of available choices in the creation of a "For . . ." clause. FIG. **10F** shows the first Create For Clause dialog box **150**. A list of available choices is presented in choice window **151**. The displayed list changes as the user moves through the For clause. In dialog box **150**, the user can select to place a condition to limit the result of a query to rows "THAT HAVE" or "THAT HAVE NOT" the condition. When the user is required to enter a column in formulating the condition, the second Create For Clause dialog box **160**, shown in FIG. **10G**, is displayed, with the tables, columns and operations designated as either selectable or nonselectable in a manner similar to the prior dialog boxes. In this way the user builds a condition clause from beginning to end. The With percent of total section is simply a flag to add "WITH PERCENT OF TOTAL" to the end of the query. This

provides a percent of total on a numeric field for every row in the result, if the query is sorted by some column. The other three types of queries have similar sections which are handled by QAUI in a similar way:

1. What percent of . . . have . . . queries have three sections, the "What percent of . . ." section, the "With . . ." section and the "Have . . ." section. In the "What percent of . . ." section the user is requested to select any table in the database as seen through the conceptual layer. Both the "With . . ." and "Have . . ." section ask for conditions as in the "For . . ." clause mentioned above.

2. Compare . . . against . . . queries have the following sections: "Compare . . .", "Against . . .", "Sort By . . .", and two "For . . ." sections. The query compares two numeric columns or computations which can have a condition placed on them in their respective "For . . ." sections. Also the result can be sorted similarly to the "Show . . ." query. QAUI 11 handles each section similarly to the "Show . . .", "For . . .", and "Sort By . . ." sections discussed above, with additional conditions placed on what can be selected set by QAES 12.

3. Show . . . as a percentage of . . . is treated the same by QAUI 11 as the Compare . . . query above except that "Compare" is replaced with "Show", and "against" is replace with "as a percentage of". This query is a special kind of comparison query.

The sections of the queries relate to the various portions of the target language SQL, however the actual terms such as "Show", "Compare", "That Have", etc. are a characteristic of the intermediate language used. As discussed more fully below, the intermediate language is designed in terms of the target language. Therefore QAUI 11 is designed with the specific intermediate language in mind in order to guide the user in creating only semantically correct queries.

B. Query Assistant Expert System (QAES) QAES 12 is called by QAUI 11 to maintain the internal state information and to determine what are allowable user choices in creating a query. Referring to FIG. 7, QAES 12 contains Blackboard 13 and Query Expert 14 which, based on the state of Blackboard 13, informs QAUI 11 what the user can and cannot do in formulating a semantically correct query. Query Expert 14 provides QAUI 11 access to the blackboard and embodies the intelligence which indicates, given the current state of Blackboard 13, what choices are available to the user for constructing a semantically correct query.

1. Blackboard

A blackboard is a conceptual form of data structure that represents a central place for posting and reading data. In the present invention. Blackboard 13 contains information about the current state of the system. As a query is being formulated by the user, Blackboard 13 is modified to reflect the selections made by the user.

Within the listed variables. Blackboard 13 maintains the following information:

- whether or not a query is being created.
- the type of query (Show, what % of, etc.)
- the current clause (Show, For, Subquery, Sorted By, etc.)
- the current set of choices of what can be selected by the user (for backup capability)
- the set of tables selected for each of the current clause, whole query, and any subqueries
- the table involved in a Count operation, if any (there can only be one)
- the table involved in an aggregate operation (there can only be one)

the table involved in a computation operation (there can only be one)
 the base table / virtual table relationship for any virtual table columns
 a string defining each condition clause (i.e. For, With, Have)

To access and manipulate the data, the following routines are provided:

- Initialize Blackboard (This sets all of the variable to zero or null state prior to the start of a query)
- Set Query Type
- Set Current Clause
- Backup current set of selectable tables, columns, and operations.
- Restore backup of set of selectable tables, columns, and operations.
- Add table to set of tables selected for each of the current clause, whole query, and any subqueries
- Remove table from set of tables selected for each of the current clause, whole query, and any subqueries
- Read list of tables selected for each of the current clause, whole query, and any subqueries
- Read/Write/Clear table involved in Count operation
- Read/Write/Clear table involved in aggregate operation
- Read/Write/Clear table involved in computation operation
- Read/Write any base table <-> virtual table relationship for any virtual columns
- Add/Remove text from string containing the whole intermediate language query and each condition clause (i.e. For, With, Have)

Possible methods for physical implementation of the blackboard include, but is not limited to, a set of encapsulated variables, database storage, or object storage, each with appropriate access routines.

After the user makes each selection in the process of building a query, Blackboard 13 is updated to reflect the current status of the query and Query Expert 14 can use the updated information to determine what choice the user should have next.

1. Query Expert

Query Expert 14 utilizes information stored on Blackboard 13 and information from the conceptual layer about the current database application to tell QAUI 11 what are the available tables, columns, and operations that the user can next select when building a query. Query Expert 14 makes this determination through the application of a set of rules to the current data. Although the rules used by the system are expressed in the illustrated embodiment by a set of If . . . Then . . . statements, it should be evident to the artisan that the rules may be implemented procedurally, through a forward or backward chaining expert system, by predicate logic, or similar expert system techniques.

Query Expert 14 examines each table and each column in each table to determine whether it can be selected by the user based on the current state of the query. Query Expert 14 also determines whether a selected table and/or column can be used in aggregate or computation operations. In addition, during the creation of a conditional "For . . ." clause, Query Expert 14 addresses further considerations.

Query Expert 14 uses some similar and some different rules during construction of each of the sections of a query. For each section of the "Show . . ." query described above, the rules employed by the Query Expert to designate what tables, columns, and operations the user can select in generating a semantically correct query are set forth below.

1. The "Show . . ." section. Below is a set of rules used to determine what tables, columns and operations are select-

23

able by the user. The term "current clause" used within the rules refers to the entire Show . . . query. The current clause becomes important in the other three types of queries which have two separate sections used in comparisons. Each of those sections are separate clauses for the purpose of the rule base.

TABLES: For each Table(x) in the database, if the following Table rules are all true, then the table is selectable. A rule of the form If . . . Then TRUE has an implied Else FALSE at the end, and there is similarly an implied Else TRUE after an If . . . Then FALSE. A table which is hidden, according to the conceptual layer, is not presented to the user for selection, but is processed by the rules in case virtual columns in non hidden tables are based on the hidden tables. If the hidden table cannot be selected, then any virtual table relying upon it cannot be selected.

Rule 211

IF the current clause is empty; Table(x) is a table already included in the current clause; there is only one other table in the current clause, and it can be joined with Table(x); or more then one table already exists in the current clause, and adding the new table results in a navigable set (There is a single common most detailed table between the tables.)
Then TRUE

24

-continued

Table(x) only through a more detailed table then there is a conflict)
Then FALSE
Rule 214

IF Table(x) is following an aggregate command, and (there are no tables in the query; there is another aggregate present and either Table(x) already has an aggregate operation applied or has a one-to-one relationship with the already aggregated table; or there is not another aggregate present, and Table(x) is the most detailed table in the query (in one-to-many relationships, the many side is more detailed then the one side)
Then TRUE

COLUMNS: For all Column(x) in a Table(x), if all the following Column rules are not false, the columns are selectable, else they are not.

Rule 221

IF Table(x) is not selectable
Then FALSE

Rule 222

IF Column(x) is a virtual column; and the table on which the column is based is not selectable
Then FALSE

Rule 223

IF There exists an aggregate on a Column(y); Column(y) is based on the same table as Column(x) or is based on a table with a one-to-one relationship with the table on which column(x) is based; and Column(x) is non-numeric
Then FALSE

-continued

Rule 212

IF Table(x) is the base table of a virtual table with a condition clause; and the virtual table has already been selected
Then FALSE

Rule 213

IF There is an aggregate being performed in the current clause; and Table(x) conflicts with the table that the aggregate is being applied to (If Table(x) is more detailed then the aggregate table or is joinable with

45

COMPUTATIONS: The same rules apply to the use of tables and columns in computations accept for additional Computation rule 231, on Column(x) which must be true.

Rule 231

IF Column(x) is selectable; and Column(x) is numeric.
Then TRUE

55

AGGREGATE OPERATIONS: For each aggregate operation Aggregate(x) (i.e., count, total, min, max, average) to be selectable for a selected column, Column(y), the following Aggregate rules must be true.

Rule 241

IF Column(y) is numeric; or Column(y) is non-numeric and Aggregate(x) = COUNT
Then TRUE

Rule 242

IF There exists an aggregate on a Column(z); Column(z) is based on the same table as Column(y) or is based on a table

with a one-to-one relationship with the table Column(y) is based on.
 Then TRUE
 Rule 243
 IF applying Aggregate(x) to Table(x) would cause a conflict with another table in the current clause (If Table(x) is less detailed than another table in the current clause then there is a conflict)
 Then FALSE

SPECIAL RULE: Special rules **251** is applied when a Column(x) is selected for display. Special rule **251** requires the user to enter an aggregate operation on the selected column.

¹⁰ the user may: clear the query (which will clear the blackboard and start over); backup a step (which will undo the last choice made); or run the query (if at an appropriate point). The pseudocode is shown to cover a certain set of condition

Rule 251
 IF There exists an aggregate on a Column(y);
 Column(y) is based on the same table as Column(x) or is based on a table with a one-to-one relationship with the table Column(x) is based on;
 Column(x) is selectable; and
 Column(x) is selected.
 Then Column(x) must have an aggregate applied, and the QAUI, at the direction of the QAES, will request one from the user.

2. The "Sort By . . ." section. No computations or aggregates are allowed on the columns selected to sort the query by. Otherwise, the rules are similar to the "Show . . ." section, with some minor changes. Table rules **210** are the same, while Computation, Aggregate, and Special rules **231**, **240**, and **251** are not applied. Finally, Column rule **223** is not applied since the "Sort By . . ." section helps define a grouping order and will cause the aggregates to group by the Sort By columns. Therefore, even though an aggregate is already applied, the Sort By column cannot, and should not, be aggregated. It does not matter whether the column is numeric or non-numeric provided the table is selectable.

²⁵ clause types, however, it will be apparent to the artisan how additional condition types can readily be added. According to the pseudocode, QAUI **11** calls the For_Clause_Top_Level_Control procedure to initiate the creation of a For clause. The Choose_entity function below is used to select a table or column and is described in more detail below.

Procedure For_Clause_Top_Level_Control **310** is a loop that creates conjunctive portions of the For clause until there are no more ANDs or ORs to be processed.

```

Procedure For_Clause_Top_Level_Control
  Repeat
    ITEM = Choose_Entity ("Entity, NMD")
    Display [THAT HAVE, THAT HAVE NOT] to the user and get a response
    RESULT = Call Make_For_Clause
    Fix parentheses if necessary
    If RESULT is no AND or OR selected, Then
      Display [AND , OR] to the user and get a response
      (user can also choose to switch to a different clause or run a query)
    Until RESULT says the last item selected is not an AND or OR and the user has
      switched to a different clause selected to run a query (i.e., "For . . ." or "Sort By . . .")
  End Procedure
  
```

3. The "For . . ." section is somewhat different from the previous two sections. Upon selecting a For condition, the user is led through a series of dialog boxes providing a set of choices to continue. Different rules apply at different points in the process of building a condition. Therefore, the knowledge as to what items are selectable by a user is contained in two forms. First, there is a procedural list of instructions, which direct the user in building a For clause, and second, there is a set of rules that are applied at specific times to tables, columns, and operations in a similar manner to the Show and Sort By clauses.

Pseudocode representations of the procedural knowledge used in directing a user to create a For clause is shown below. Although not explicitly stated in the pseudocode, after every selection made by the user, Blackboard **13** is updated, and the current query is updated for display to the user. Also, at any time during the creation of the For clause,

⁵⁰ Function Make_For_Clause **320** handles some special types of For clauses by itself and sends the general type of For clause constructs (i.e. constraints) to function Make_Constraint **330**. The result of the function is an indication as to whether there is a pending And or Or in the query.

```

Function Make_For_Clause
  CHOICE = Display ['Place a condition', 'other conditions', '('] to the user and get a
response
  If CHOICE = '(' Then
    RESULT = Call Make_For_Clause
    Add ')' to query
    Return RESULT
  ElseIf CHOICE = 'Place a condition' Then
    Return the result of Make_Constraint ("Attribuc, NMD", False)
  ElseIf CHOICE = 'other conditions' Then
    CHOICE = Display [>,<,<=>,>=<=>, '... FOR EVERY ...', '... OF ANY ...', 'EVERY ...']
to user and get a response
    If CHOICE is from the set [>,<,<=>,>=<=>] Then
      Get a value
      ITEM = Choose_Entity ("Entity, Detail, ForTable")
      CHOICE = Display [THAT HAVE, THAT HAVE NOT, AND, OR] to user and get
response
      If CHOICE is from the set [THAT HAVE, THAT HAVE NOT] Then
        Return the result of CALL Make_For_Clause
      ElseIf CHOICE is from the set [AND, OR] Then
        Return an And or Or is selected
      Endif
    ElseIf CHOICE is from the set {'... FOR EVERY ...', '... OF ANY ...'} Then
      ITEM1 = Choose_Entity ("Entity, Detail, ForTable")
      ITEM2 = Choose_Entity ("Entity, NLD,NOTENT")
      Return no And or Or selected
    ElseIf CHOICE = 'EVERY ...' Then
      Return the result of Make_Constraint ("Attribute, Detail, ForTable", True)
    Endif
  Endif
End Function

```

Function Make_Constraint 330 is the heart of the cre- 30 part of the For clause is already present when this function
ation of the "For" clause. It takes as parameters some QAES is called. The result of the function is an indication whether
rule parameters and a flag which indicates whether every or not the last thing selected was an AND or Or.

```

Function Make_Constraint (RULE_PATTERN, EVERY_CLAUSE_FLAG)
  ITEM = Choose_Entity (RULE_PATTERN)
  If ITEM is numeric Then SELECTION includes [<,>,<=>,>=<=>, Between]
  Endif
  If ITEM is alphanumeric Then
    SELECTION includes [<,>,<=>,>=<=>, Between, Begins with, Ends with, Contains]
  Endif
  If ITEM is a date Then
    SELECTION includes [Specific date, Since, Before, Between]
  Endif
  SELECTION = SELECTION + [is Blank, Is Not Blank]
  CHOICE = Display SELECTION to user and get response
  If CHOICE is from the set [Is Blank, Is Not Blank] Then
    Return no And or Or selected.
  ElseIf CHOICE is numeric or alphanumeric 'Between' Then
    Get before value limited to proper data type
    Got after value limited to proper data type
    Return no And or Or selected.
  ElseIf CHOICE is from the set [Begins with, Ends with, Contains] Then
    Get alphanumeric value
    Return no And or Or selected.
  ElseIf CHOICE is from the set [Specific date, Since, Before] Then
    Get date value
    Return no And or Or selected.
  ElseIf CHOICE is date 'Between' Then
    Get first date
    Get second date
    Return no And or Or selected.
  ElseIf CHOICE is from the set [<,>,<=>, >=<=>] Then
    If EVERY_CLAUSE_FLAG is TRUE Then
      Get value
      Return no And or Or selected.
    Else CHOICE = Display ['Enter a value', 'Place a condition', Total, Average,
Maximum, Minimum] to user and get response
    If CHOICE = 'Enter a value' Then
      Get value
      Return no And or Or selected.
    ElseIf CHOICE = 'Place a condition' Then
      ITEM = Choose_Entity (RULE_PATTERN)
      CHOICE = Display [OF, AND, OR] to user and get response

```


-continued

```

If CHOICE = OF Then
  ITEM = Choose_Entity ("Entity, OFENT")
  CHOICE = Display [THAT HAVE, THAT HAVE NOT, AND, OR] to
    user and get response
  If CHOICE is from the set [THAT HAVE, THAT HAVE NOT]
    Then
      Return the result of CALL Make-For-Clause
    ElseIf CHOICE is from the set [AND , OR] Then
      Return an And or Or is selected
    Endif
  ElseIf CHOICE is from the set [AND, OR] Then
    Return an And or Or is selected
  Endif
Elseif CHOICE is from the set [Total, Average, Maximum, Minimum] Then
  If CHOICE is from the set [Total, Average] Then
    ITEM = Choose_Entity("Numeric Attribute, OFENT, NMD")
  Else
    ITEM = Choose_Entity("Affribute, OFENT, NMD")
  Endif
Endif
ITEM = Choosc_Entity ("Entity, OFENT")
CHOICE = Display [THAT HAVE, THAT HAVE NOT, AND, OR] to user and get
  response
If CHOICE is from the set [THAT HAVE, THAT HAVE NOT] Then
  Return the result of CALL Make For Clause
Elseif CHOICE is from the set (AND, OR) Then
  Return an And or Or is selected
Endif
Endif
Endif
End Function

```

In the above pseudocode, the function Choose_Entity is not defined. This function uses the second type of knowl-
edge. The function is called with a set of parameters, and
based on those parameters, the user is asked to choose either
a table or column. As with the other clauses, this information
is presented to the user in a manner to distinguish which
choices the user can make. QAES 12 determines what
selection the user may make by applying a set of rules to the
tables and columns as in the other clauses.

There is an additional element, however, in the rule base
for the For clause. The rule base is expanded to include
special circumstances which are specified by a parameter
Choose_Entity. The parameter, in the pseudocode, takes the
form of a list of conditions in a string separated by commas.
There are two types of condition, those which inform QAES
12 what type of dialog item the user will be selecting and
therefore what type of dialog box to display, and those which
are conditions in the rules. Types of item parameters include:

Entity—indicates that the user needs to select a table;

Attribute—indicates that the user needs to select a col-
umn; and

Numeric Attribute—indicates that the user can only select
a numeric column.

If the user needs to select a table, then the rules will not
be applied to the columns, since they will not be displayed.
The condition type parameters are:

NMD The table can be No More Detailed then any other
table in the current clause

NLD The table can be No Less Detailed Then any other
table in the current clause

Detail The table must be the most detailed table in the
current clause

ForTable The table must be Identical or one-to-one with
any tables in the For clause

OFENT The table must be Identical or one-to-one with
any table in current clause

NOTENT The table must not be identical or one-to-one
with any table in current clause.

In a one-to-many relationship, the table on the many side
of the relationship is more detailed then the table on the one
side. As discussed previously, the term "current clause"
actually refers to the entire query in a Show . . . query. The
current clause becomes important in the other three types of
queries which have two separate section used in compari-
sons. Each of those sections are separate clauses for the
purpose of the rule base. The rule base used in the For clause
is set out below.

TABLES: Table rules 211–214 are applied. In addition,
the following Table Parameter rules are applied.

Rule 311

IF Parm contain "NMD"; and
a less detailed table then Table(x) is already in the current clause

Then FALSE

Rule 312

IF Parm contains "NLD"; and
a more detailed table then Table(x) is already in the current clause

Then FALSE

Rule 313

IF Parm contains "Detail"; and
(there is a more detailed table then Table(x) in the current clause;
Table(x) already exists in the current clause; or

-continued

```

a table with a one-to-one relationship with Table(x) exists in the current
clause and it is aggregated)
Then FALSE
Rule 314
IF Parm contains "ForTable";
Table(x) doesn't exist in the For clause; and
Table(x) does not have a one-to-one relationship with any table in the For
clause
Then FALSE
Rule 315
IF Parm contains "OFENT";
Table(x) doesn't exist in the current clause; and
Table(x) does not have a one-to-one relationship with any table in the
current clause
Then FALSE
Rule 316
IF Parm contains "NOTENT"; and
(Table(x) exists in the current clause; or
Table(x) has a one-to-one relationship with any table in the current
clause)
Then FALSE

```

COLUMNS: In addition to Column rules 221–223, the following Column Parameter rule is applied.

```

Rule 321
IF Parm contains "Numeric Attribute";
Column(x) is non-numeric
Then FALSE

```

Computation rule 231 is also applied.
4. "With percent of Total" check box. The user can add this phrase to the end of a query if two things are true: (1) the last item in the Show clause was numeric; and (2) there is a sort specified in the Sort By clause.

The same set of rules are used in the equivalent sections of the other three query types as discussed in the earlier section on QAUI 11. These queries are considered two clause queries with each clause represented by the ellipses in the queries. Blackboard 13 is set to the current clause, and the rules which refer to current clauses use the clause being built by the user. The rules are primarily the same as the Show . . . query with the following caveats:

1. In the What percent of . . . have . . . queries, the "What percent of . . ." section is limited to one table in the database, so there are no real rules applied. The "With . . ." and "Have . . ." sections are then the same as the "For . . ." section in the Show . . . query.

2. In the Compare . . . against . . . queries, the "Compare . . ." and "Against . . ." sections are limited to a numeric columns, including aggregates and computations. Also in each of the sections, there can be only one column, aggregate or computation. The "For . . ." and "Sort By . . ." clause use the same rule sets as those in the Show . . . queries.

3. In the Show . . . as a percentage of . . . queries, the "Show . . ." and "as a percentage of . . ." sections are limited to a numeric columns, including aggregates and computations. Also in each of the sections, there can be only one column, aggregate or computation. The "For . . ." and "Sort By . . ." clause use the same rule sets as those in the Show . . . queries.

To illustrate how Query Assistant 10 prevents the user from having the opportunity to formulate the incorrect query involving the three tables CUSTOMERS, ORDERS and LINE_ITEMS from FIGS. 1A, 1F, and 1G, respectively, with the relations shown in FIG. 6, the steps that the user would take to attempt the incorrect query are described. First, the user would invoke Query Assistant 10 and select a Show . . . query. At this point all of the tables and columns

would be selectable since nothing has yet been selected, however, the Run Query box is not selectable, and the other sections of the query are not selectable until there is something in the Show section. Next, the user would select the column Name in the CUSTOMERS table for display. Again, after applying the rules, all of the tables and columns are selectable. This is indicated by the rule base because all tables can be joined with CUSTOMERS, and there has not been an aggregate defined. Next, the user would select Order Dollars from the ORDERS table to display. All tables and columns are still selectable for the same reason.

Next, the user would select to modify Order Dollars. After applying the rules, QAES 12 would indicate that any of the aggregates can be applied to Order Dollars since Order Dollars is numeric and there are no other aggregates. Next, the user would select a Total on Order Dollars. After applying the rules, QAES 12 would determine that the LINE_ITEMS, PRODUCT, CODE, and VENDOR tables are no longer selectable because of Table rule 213. Also, only numeric columns are selectable in the ORDERS table and they must be aggregated as dictated by Column rule 223 and Special rule 251.

Finally, columns in the CUSTOMERS table are selectable, but cannot be aggregated because of Aggregate rule 242. The user is not allowed to select the LINE_ITEMS table once an aggregate is placed on ORDER DOLLARS so the incorrect query cannot be formulated. Similarly, if a column in the LINE_ITEMS table had been selected prior to placing the Total on Order Dollars, Order Dollars could not be aggregated because of Aggregate rule 243.

More complex queries are handled in the same way. After each user selection, QAES 12, through QAUI 11, provides a list of available choices based on knowledge it has about databases and the information it contains in the conceptual layer. It applies rules to all applicable tables, columns, and operations to determine the list of selectable items. During the creation of a For clause, a procedural component is introduced, but the method of operation is substantially the same.

IV. INTERMEDIATE LANGUAGE

As discussed earlier, because the set of semantically valid queries cannot be described by a context-free grammar, a grammar is not given for the intermediate language, and one is not used to create user choices. Rather, the intermediate

language is defined by the templates and choices presented to the user by Query Assistant 10. The templates are screen definitions whose contents (i.e. picklist contents and active vs. inactive elements) are governed by QAES 12. Condition clause generation is driven by a separate module in QAES 12. The definition of the intermediate language is precisely those queries that can be generated by Query Assistant 10.

The design of the intermediate language, however, is driven from the bottom (SQL Generator 20) and not the top (Query Assistant 10). The architecture of Query System 1 is designed to minimize the amount of processing by SQL Generator 20 by making the intermediate language as similar as possible to the target language (SQL) while providing a more easily understandable set of linguistic constructs. Building upon the design of the language, Query Assistant 10 is built to implement the production of semantically correct queries using the linguistic constructs of the language, which in turn can further simplify the design of SQL Generator 20.

In natural language systems, the problem lies in converting a representation of a natural language to a target language, such as SQL. In the present invention, conversion of the intermediate language to a target language is straightforward because the intermediate language was designed to accommodate this process. The intermediate language is designed by starting with the target language (the illustrated embodiment, SQL), and making several modifications.

First, the grouping and table specification constructs, which in SQL are specified by the GROUP BY, HAVING, and FROM clauses respectively, are deleted, so that the user need not specify them. Rather, this information can be inferred readily from the query. For example, if the user selects a column for display, the table from which the column comes needs to be included in the table specification (i.e. FROM clause). When a user selects to view columns in a table without selecting a primary key, the user would likely want to see the column results grouped together, so that like values are in adjacent rows of the output and duplicates are discarded. This is specified in the GROUP BY clause of SQL, but it can be inferred. In SQL, the HAVING clause is a special clause which operates as a WHERE clause for GROUP BY items. This is also readily inferred from which columns are grouping columns and if they have specified conditions.

Second, join specifications are deleted from the condition clause. SQL requires an explicit definition of the joins in its WHERE clause (i.e. WHERE CUSTOMER.CUSTOMER#=ORDER.CUSTOMER#). This information can be inferred or specifically requested when creating a query if necessary, but it does not form a part of the intermediate language query.

Third, specific and readily understandable patterns are defined for each type of subquery supported by the intermediate language. For example, the English pattern "MORE THAN 2<category>" can be defined to have a specific SQL subquery expansion.

Fourth, the remainder of the target language is replaced with easily understandable words or phrases that, when strung together, form a comprehensible sentence. For example, using SQL, "SELECT" is replaced with "Show". "WHERE" is replaced with "For", "ORDER BY" is replaced with "Sorted By" and so on.

Finally, synonyms are provided for various words, phrases and constructs. Target language constructs may look differently in the intermediate language depending on the

type of query to be formed if the query is to be an easily understood sentence. This also allows the user multiple ways of specifying concepts, including, but not limited to: dates (i.e. Jan. 1, 1994 v. 01/01/94. etc.), ranges (between x and y, >x and <y, last month, etc.); and constraints (>, Greater Than. Not Less Than or Equal).

V. SQL GENERATOR

Given the design of the intermediate language described above, SQL Generator 20 need only perform two basic functions. First, it needs to infer the implicit portions of the query that are explicitly required in the target language, such as the GROUP BY clause in SQL, or the explicit joins in the WHERE clause. This information is easily inferred because of the design of the intermediate language. Second, SQL Generator 20 needs to resolve synonyms and transform the more easily understood intermediate language into the more formal target language through a series of transformations by pattern substitution. It is this set of patterns that give the intermediate language its basic look and feel.

Internally, the intermediate language has a component that is independent of the database application, and a component that is specific to the database application. The application independent component is represented by the sets of patterns used in the pattern substitution transformations and the set of routines used to infer implicit information. The application specific component is represented by the conceptual layer which contains information used in both basic functions of SQL Generator 20.

SQL Generator 20 has no expertise concerning what is and is not a semantically valid query in the intermediate language. If the user bypasses Query Assistant 10 to directly enter a query using the syntax of the intermediate language, the user can obtain the same incorrect results that can be obtained with conventional point-and-shoot query tools.

A. Flow of Control

FIGS. 11A and 11B depict a flowchart of the flow of control of SQL Generator 20. Each of the steps is described in detail below. SQL Generator 20 applies a series of transformations to the intermediate language input to produce a resulting SQL statement. If one of these transformations fails, an error message will be displayed. An error message is only possible if the input query is not part of the intermediate language (i.e. was not, and could not be generated by Query Assistant 10). SQL Generator 20 takes as input an intermediate language query and produces an SQL statement as output by applying the steps described below.

In step 402, the intermediate language query is tokenized, i.e. converted to a list of structures. Each structure holds information about one token in the query. A token is usually a word, but can be a phrase. In this step, punctuation is eliminated, anything inside quotes is converted to a string, and an initial attempt is made to categorize each token.

In steps 404 and 406, the intermediate language query is converted to an internal lexical format. The conversion is done via successive pattern matching transformations. The clause is compared against a set of patterns until there are no more matched patterns. The lexical conversion pattern matcher is discussed in more detail below. The resulting

internal format differs from the intermediate language format in several ways:

- (a) Synonyms within the intermediate language for the same construct are resolved to a single consistent construct (i.e., "HAS" and "HAVE" become "HAVE").
- (b) Synonyms for tables and columns are resolved, utilizing the names specified in the conceptual layer and by converting column names to fully qualified column names in the form of Table__Name. Column__Name
- (c) Dates are converted to a Julian date format
- (d) Extraneous commas and ANDs (except those in condition clauses) are deleted
- (e) Condition clauses are transformed to match one or more predefined WHERE clause patterns stored as ASCII text in an external file
- (f) Special symbol are inserted to demarcate the beginning and middle of "what percent of . . .", "show . . . as a % of . . .", and "compare . . ." queries.
- (g) A special symbol is inserted to demarcate the object of every FOR clause.
- (h) Certain words designated as Ignore words are eliminated. (i.e. The, That, etc.)

When no further patterns can be matched, control transfers to step 408, where it is determined whether CREATE VIEW statements are necessary. If so, in steps 410 and 412, SQL Generator 20 is called recursively as a subroutine to generate the required views. As the view is generated the recursive call to SQL Generator 20 is terminated. CREATE VIEW (an SQL construct) is required for queries in the intermediate language which call for percentage calculations or otherwise require two separate passes of the database (i.e. comparisons). The types of queries that Query Assistant 10 can produce that require a CREATE VIEW statement are of a predetermined finite set, and SQL Generator 20 includes the types of queries which require CREATE VIEW generation. An example type of query where it is required is "Compare X against Y" where X and Y are independent queries that produce numeric values. Within each recursive call to SQL Generator 20, pattern matching is conducted to resolve newly introduced items. Control then passes to step 414.

In step 414, the internal lexical format is converted into an internal SQL format, which is a set of data structures that more closely parallel the SQL language. The internal SQL format partitions the query into a sets of strings conforming to the various SQL statement elements including: SELECT columns, WHERE clauses, ORDER BY columns, GROUP columns, Having clause flag, FROM table/alias pairs, JOINS. In this step, the SELECT, and ORDER BY sections are populated, but WHERE clauses are maintained in lexical format for processing at the next step. The other elements are set in the following steps if necessary.

In the ensuing steps 416 and 418, the lexical WHERE phrases are compared with a set of patterns stored in an external ASCII text file. If a match is found, a substitution pattern found in the external file is used for representing the structure in an internal SQL format. In this way, the WHERE clause is transformed from the intermediate language to the internal SQL format equivalent.

If any table references have been introduced into the internal SQL structure as columns, they are converted to column references in step 420. This can occur on queries like

"show customers". Virtual table references are also expanded in this step using the conceptual layer information to include the table name and the virtual table condition, if present, which is added to the internal structure.

If there are any columns in the ORDER BY clause that are not in the SELECT, they are added to the SELECT in step 422. In step 424, Julian dates are converted to dates specified in appropriate SQL syntax. Next, in step 426, virtual columns are expanded into the expressions defined in the conceptual layer by textual substitution. This is why virtual column expressions are defined according to SQL expressions or other expressions understood by the DBMS. In this step, the expression of a virtual column will be added to the WHERE clause—a Lookup command will simply mac another join condition in the WHERE clause.

In step 428, the FROM clause of the SQL statement is created by assigning aliases for each table in the SELECT and WHERE clauses, but ignoring subqueries that are defined during the pattern matching of steps 416 and 418. In step 430, the ORDER BY clause is converted from column names to positional references. Some SQL implementations will not accept column names in the ORDER BY clause—they require the column's position in the SELECT clause (i.e. 1, 2 etc.). This step replaces the column names in the ORDER BY clauses with their respective column order numbers.

In step 432, the navigation path is computed for required joins. This is done using a minimal spanning tree as described above. This is a technique commonly used for finding the shortest join path between two tables, but other techniques will work equally well. If additional tables are required then they are added. Also, by default, the shortest join path is created. However, if the user designated a different join path which was predefined by the administrator and put in the conceptual layer, that path is used. If it is determined in step 434 that new tables are required, they are added in step 436 to the FROM clause. Then, in step 438, the WHERE clause join statements are created in the internal SQL structure.

In step 440, SELECT is converted to SELECT DISTINCT, if necessary. This is required if the query does not include the primary key in the Show clause of the query and there are only non-numeric columns (i.e. "Show Customer State and Customer City"). The primary keys are defined as Customer Number and Customer Name in the CUSTOMERS table. SELECT DISTINCT will limit the query to distinct sets and will group the results by columns in the order listed. Using SELECT alone will result in one output line for every row in the table.

In step 442, the GROUP BY clause is added to internal SQL (if necessary), as are any inferred SUMs. This is required if the query does not include the primary key in the Show clause of the query and there are numeric columns (i.e. "Show Customer State and Customer Balance"). The primary key is the Customer Number in the CUSTOMERS table and here Customer Balance is a numeric field. What the user wants is to see the balance by state. Without including Group By and SUM, there will be a resulting line for every row in the CUSTOMERS table. This step places any numeric fields in SUM expression and places all non-numeric fields in a GROUP BY clause. For example, the above query would produce the following SQL.

```
(5)  SELECT T1.STATE, SUM (T1.BALANCE)
      FROM CUSTOMERS T1
      GROUP BY T1.STATE
```

In step 444, COUNTs are converted to COUNT (*), if necessary. This is required, as an SQL convention, where the user requests a count on a table. For example, the query

```
{ } or
[ ] a single phrase
~ optional
!???x variable that matches anything, x is a number.
!ENTx table variable which matches any table name, x is a number in case of
multiple tables in the pattern.
!ATTx column variable which matches any column name, x is a number in
case of multiple columns in the pattern.
!VALx value variable which matches any numeric value, x is a number for
multiple values in the pattern.
!FUNCTIONx function variable which matches a function that can be
applied to a column (i.e. SUM, AVG, etc.), x is a number for multiple
functions in the pattern.
```

“Show The Count of Customers” produces the SQL code 25

```
(6)  SELECT COUNT (*)
      FROM CUSTOMERS T1
```

Finally, in step 446, the internal SQL format is converted into textual SQL by passing it through a simple parser. 30

B. Pattern Matching

Steps 404 and 416 transform the intermediate language query using pattern matching and substitution techniques. These steps help to define the intermediate language more than any other steps. By modifying these pattern/substitution pairs the intermediate language could take on a different look and feel using different phrases. Accordingly, Query Assistant 10 would need to be able to produce those phrases. Further, by adding patterns, the user can be given more ways of entering similar concepts (when not using Query Assistant 10), and more types of subqueries can be defined. For every new type of subquery defined as a pattern, the Where clause generation function of Query Assistant 10 would need to be modified to provide the capability. 35

Two types of patterns used in SQL Generator 20. The first, used in step 404, is a simple substitution, while the second, used in converting Where clauses in step 416, is more complex because it can introduce new constructs and subqueries. 40

1. Lexical conversion pattern matching

In the lexical conversion pattern matching of step 404, a text string of the query is compared to a pattern, and if a substring of the query matches a pattern, the substring is replaced with the associated substitution string. Patterns take the form of: 45

PRIORITY SUBSTITUTION<-PATTERN

PRIORITY is a priority order for the patterns which takes the form of #PRIORITY-? with ? being a whole number greater than or equal to 0. This provides an order for the pattern marcher with all #PRIORITY-0 patterns being processed before #PRIORITY-1 patterns and so on. Within a given priority, the patterns are applied in order listed. If the pattern does not begin with a priority, it has the same priority 50

as the most recently read pattern (i.e. all patterns have the same priority until the next priority designation)

PATTERN is what is compared against the query to find a match. Textual elements which match directly with words in the query along with the following key symbols: 5

SUBSTITUTION is the replacement text. Every instance of !???, !ENTx, !ATTx, or !VALx is replaced with the table, column or value bound to the variable in the PATTERN.

As an example, the pattern “PRIORITY-0 AND THAT HAVE <- {[AND HAVE] [AND HAS]}” indicates that the phrases “AND HAVE” and “AND HAS” are synonyms for the phrase “AND THAT HAVE” and will be accordingly substituted. The brackets signify phrases. The braces signify multiple synonyms. The “#PRIORITY-0” entries define the pattern as having a priority of 0 so that this rule would apply before any priority 1 rules. etc. 55

Another example pattern is “(!ATT1>=!???1 and !ATT1<=!???2)<- {[!ATT1 BETWEEN !???1 AND !???2] [!ATT1 FROM !???1 TO !???2]}”. In this case the pattern would match substrings of the form of “BALANCE BETWEEN 10000 AND 50000” or “BALANCE FROM 10000 TO 50000” and would substitute it with “BALANCE >=10000 AND BALANCE<=50000”. As is evident from the form of the patterns, the intermediate language which is understandable to the SQL Generator can be simply varied by changing these patterns or adding new patterns to recognize different words or phrases. 60

The set of patterns used in step 404 of the illustrated embodiment (i.e., for one instance of an intermediate language) is shown below. 65

```

Pattern 501
#PRIORITY-0 AND THAT HAVE <-- {[AND HAVE][AND HAS]}
Pattern 502
#PRIORITY-0 AND THAT DO NOT HAVE <-- {[AND DO NOT HAVE][AND DOES NOT HAVE]}
Pattern 503
#PRIORITY-0 WHERE NOT <-- {WITHOUT [THAT DO NOT HAVE][THAT DOES NOT HAVE]}
Pattern 504
#PRIORITY-0 HAVE NOT <-- {[DO NOT HAVE][DOES NOT HAVE]}
Pattern 505
#PRIORITY-2 !ENT1, !ENT2 <-- {!ENT1 !ENT2}
Pattern 506
#PRIORITY-5 !ENT1, !ATT1 <-- {!ENT1 !ATT1}
Pattern 507
#PRIORITY-0 PCT_TOTAL <-- [WITH {% PERCENT PERCENTAGE} OF TOTAL ]
Pattern 508
#PRIORITY-0 !ATT1 PCT_TOTAL <--
    [!ATT1 WITH {% PERCENT PERCENTAGE} OF TOTAL SUBTOTAL]
Pattern 509
#PRIORITY-0 WHAT_PERCENT_BEGIN <-- [WHAT {% PERCENT PERCENTAGE} ~ OF]
Pattern 510
#PRIORITY-0 AS_PCT_MIDDLE <-- [AS A {% PERCENT PERCENTAGE} OF]
Pattern 511
#PRIORITY-0 COMPARE_BEGIN <-- COMPARE
Pattern 512
#PRIORITY-2 OFENTITY!! !ENT1 WHERE
    <-- [FOR {!ENT1 THAT HAVE}
        [!ENT1 THAT HAS]
        [!ENT1 CANTFOLLOW XDATE]]
Pattern 513
#PRIORITY-2 OFENTITY!! "ENT1 WHERE <-- {[WHERE !ENT1 HAVE][WHERE !ENT1 HAS]}
Pattern 514
#PRIORITY-2 !ATT1 = !VAL1 <-- [!ATT1 = "!VAL1 "]
Pattern 515
#PRIORITY-2 XDATE MTH !MTH1 ENDPT <-- !MTH1
Pattern 516
#PRIORITY-2 XDATE MTH !VAL1 DAY !VAL2 CYR !VAL3 ENDPT <-- [!VAL1!/VAL2!/VAL3]
Pattern 517
#PRIORITY-2 XDATE MTH XDATE MTH !MTH1 DAY !VAL1 CYR !VAL2 ENDPT <-- [!MTH1 !VAL1 ~, !VAL2]
Pattern 518
#PRIORITY-2 XDATE MTH !MTH1 CYR !VAL1 ENDPT <-- [!MTH1 ~, !VAL1]
Pattern 519
PRIORITY-2 XDATE XDATE RDAY 0 ENDPT <-- TODAY
Pattern 520
PRIORITY-2 XDATE RDAY -1 ENDPT <-- YESTERDAY
Pattern 521
PRIORITY-2 XDATE RWEK 0 ENDPT <-- [THIS WEEK]
Pattern 522
#PRIORITY-2 XDATE RWEK -1 ENDPT <-- [LAST WEEK]
Pattern 523
#PRIORITY-2 XDATE RMTH 0 ENDPT <-- {[THIS MONTH] MTD}
Pattern 524
#PRIORITY-2 XDATE RMTH -1 ENDPT <--[LAST MONTH]
Pattern 525
#PRIORITY-2 XDATE RCYR 0 ENDPT <--[THIS YEAR ] YTD}
Pattern 526
#PRIORITY-2 XDATE ROTR 0 ENDPT <--[THIS QUARTER]
Pattern 527
#PRIORITY-2 XDATE RQTR -1 ENDPT <-- [LAST QUARTER]
Pattern 528
#PRIORITY-2 XDATE RQTR - !VAL1 ENDPT <-- [!VAL1 QUARTERS AGO]
Pattern 529
#PRIORITY-2 XDATE RQTR - !VAL1 POINT2 RQTR -1 ENDPT <-- [LAST !VAL1 QUARTERS]
Pattern 530
#PRIORITY-2 XDATE RCYR -1 ENDPT <-- [LAST YEAR]
Pattern 531
#PRIORITY-2 XDATE RDAY - !VAL1 ENDPT <-- [!VAL1 DAYS AGO]
Pattern 532
#PRIORITY-2 XDATE RDAY - !VAL1 POINT2 RDAY -1 ENDPT <-- [LAST !VAL1 DAYS]
Pattern 533
#PRIORITY-2 XDATE RWEK - !VAL1 ENDPT <-- [!VAL1 WEEKS AGO]
Pattern 534
#PRIORITY-2 XDATE RWEK - !VAL1 POINT2 RWEK -1 ENDPT <--[LAST !VAL1 WEEKS]
Pattern 535
#PRIORITY-2 XDATE RMTH - !VAL1 ENDPT <-- [!VAL1 MONTHS AGO]
Pattern 536
#PRIORITY-2 XDATE RMTH - !VAL1 POINT2 RMTH -1 ENDPT <--[LAST !VAL1 MONTHS]
Pattern 537
#PRIORITY-2 XDATE RCYR - !VAL1 ENDPT <--[!VAL1 YEARS AGO]
Pattern 538
#PRIORITY-2 XDATE RCYR - !VAL1 POINT2 RCYR -1 ENDPT <-- [LAST !VAL1 YEARS]

```

-continued

```

Pattern 539
#PRIORITY-2 !ATT1 XDATE !VAL1 -1 <- [!ATT1 >= XDATE !VAL1 !VAL2]
Pattern 540
#PRIORITY-2 !ATT1 XDATE !VAL1 -1 <- [!ATT1 {SINCE >} XDATE !VAL1 !VAL2]
Pattern 541
#PRIORITY-2 !ATT1 XDATE -1 !VAL1 <- [!ATT1 <= XDATE !VAL1 !VAL2]
Pattern 542
#PRIORITY-2 !ATT1 XDATE -1 !VAL1 <- [!ATT1 {BEFORE <} XDATE !VAL1 !VAL2]
Pattern 543
#PRIORITY-2 !ATT1 XDATE !VAL1 !VAL2 <- [!ATT1 = XDATE !VAL1 !VAL2]
Pattern 544
#PRIORITY-2 !ATT1 XDATE !VAL1 !VAL4
<- [!ATT1 BETWEEN XDATE !VAL1 !VAL2 AND XDATE !VAL3 !VAL4]
!ATT1 FROM XDATE !VAL1 !VAL2 TO XDATE !VAL3 !VAL4
Pattern 545
#PRIORITY-2 SUM <- {TOTAL {SUM OF}}
Pattern 546
#PRIORITY-2 COUNT<- [HOW MANY]
Pattern 547
#PRIORITY-2 AVG <- {AVERAGE AVE}
Pattern 548
#PRIORITY-2 MIN <- MINIMUM
Pattern 549
#PRIORITY-2 MAX <- MAXIMUM
Pattern 550
#PRIORITY-2 !!FUNCTION (!ATT1) <- [!FUNCTION !ATT1]
Pattern 551
#PRIORITY-1 SELECT COUNT <- {COUNT FIRSTWORD}
Pattern 552
PRIORITY-2 !!FUNCTION1 (!!FUNCTION2 (!ATT1))
<- [!!FUNCTION1 !!FUNCTION2 !ATT1]
{!ATT1 !!FUNCTION1 !!FUNCTION2}
Pattern 553
PRIORITY-0 SELECT COUNT <- {COUNT FIRSTWORD}
Pattern 554
PRIORITY-2 COUNT (DISTINCT!ATT1) <- [COUNT !ATT1]
Pattern 555
PRIORITY-2 COUNT (!ENT1) <- [COUNT !ENT1]
Pattern 556
!ENT1 WHERE <-[!ENT1 {FOR [THAT HAVE][THAT HAS] HAVING}]
Pattern 557
#PRIORITY-2 !ATT1 WHERE !ATT1 <- [WHERE !ATT1 WHERE !ATT1]
Pattern 558
#PRIORITY-2 WHERE !ATT1 <- [WHERE !ATT1 WHERE]
Pattern 559
#PRIORITY-2 !ENT1 WHERE <- [!ENT1 THAT {HAVE HAS}]
Pattern 560
#PRIORITY-2 (!ATT1 >= !???1 AND !ATT1 <= !???2) <-
[!ATT1 BETWEEN !???1 AND !???2] <-
[!ATT1 FROM !???1 TO !???2]}
Pattern 561
#PRIORITY-2 SELECT <- [{WHERE IS DO AM WERE ARE WAS WILL HAD HAS HAVE DID
DOES CAN I LIST SHOW GIVE PRINT DISPLAY OUTPUT FORMAT PLEASE RETRIEVE
CHOOSE FIND GET LOCATE COMPUTE CALCULATE HOW WHOSE DO WHAT WHO WHEN
HOW WHOSE [WHAT {IS ARE}]} FIRSTWORD]
Pattern 562
NOT NULL <-[!IS NOT NULL][!IS NOT BLANK]
Pattern 563
NULL <- [IS BLANK]
Pattern 564
=<- IS
Pattern 565
#PRIORITY-2 <><- {NEQ!= [NOT EQUAL ~TO]}
Pattern 566
><- {OVER GREATER [GREATER THAN][MORE THAN] ABOVE
[NOT LESS THAN OR EQUAL ~ TO]}
Pattern 567
#PRIORITY-2 >= <- {[GREATER THAN OR EQUAL ~ TO][GT OR EQ ~TO][AT LEAST] =>
[NOT LESS THAN][GTE ~ TO][MORE THAN OR EQUAL ~ TO]}
Pattern 568
#PRIORITY-2 < <-[!LESS [LESS THAN] I BELOW UNDER [NOT MORE THAN OR EQUAL ~ TO]
]
Pattern 569
#PRIORITY-2 <= <- [!LESS THAN OR EQUAL ~ TO][LT OR EQ ~ TO][AT MOST] =<
[NOT MORE THAN][LTE ~ TO]}
Pattern 570
#PRIORITY-2 = <- [!EQUAL ~ TO]}
Pattern 571
ORDERBY <- [- AND {BY [SORTED BY]}]
Pattern 572

```

```

#PRIORITY-5 ORDERBY <— [ORDER BY]
Pattern 573
#PRIORITY-0 DESC <— {(DESCENDING [IN {DECREASING DESCENDING} ORDER])}
Pattern 574
#PRIORITY-0 ASC <— {ASCENDING [IN {INCREASING ASCENDING} ORDER]}
Pattern 575
#PRIORITY-0 THATBEGINWITH !?? <— [BEGINS WITH !???]
Pattern 576
#PRIORITY-0 THATENDWITH !??? <— [ENDS WITH !???]
Pattern 577
#PRIORITY-0 THATCONTAIN !??? <— [CONTAINS !???]
Pattern 578
#PRIORITY-0 THATSOUNDLIKE !??? <— {~ THAT SOUNDS LIKE !???}
Pattern 579
HAVE <— HAS
Pattern 580
#PRIORITY-5 WHERE <— {HAVING [THAT HAVE][THAT HAS]}
Pattern 581
#PRIORITY-5 >= 1 <— {{FOR OF}{ANY SOME}}
Pattern 582
#PRIORITY-5 SELECT <— [SELECT EVERY]
Pattern 583
#PRIORITY-2 !ENT1 WHERE EVERY <— [!ENT1 EVERY]
Pattern 584
EVERY <— {ALL EACH}
Pattern 585
!ENT1 WHERE <— [!ENT1 {ARE FOR WITH WHICH HAVE [THAT HAVE][THAT HAS] HAVING}
Pattern 586
#PRIORITY-2 !ATT1 EVERY !ENT1 <— [EVERY !ATT1 ~ OFENTITY!! !ENT1]
Pattern 587
>= <— {SINCE FOLLOWING AFTER}
Pattern 588
<= <— {BEFORE[PRIOR TO]PRECEEDING}
Pattern 589
#PRIORITY-2 WHERE <— [WHERE WHERE]

```

When SQL Generator 20 is initiated, the patterns above are read from an external text file. The patterns are stored in a structure which, for each pattern, contains a priority, the pattern as a text string, and the substitution as a text string. Construction and operation of binding pattern matchers are well known in the art and any of the many techniques that provide the aforementioned capabilities is within the scope of this invention.

2. Conversion of lexical WHERE to internal SQL format

In step 416, a set of patterns is used to convert the Where clause of a query into SQL. These patterns help expand the type of queries the intermediate language can handle, and often map to SQL structures which require subqueries. By adding additional patterns, the intermediate language can be expanded to represent more types of complex SQL queries.

Similarly to the prior pattern matching step, this step compares a text string of the Where clause of a query against a pattern. The Where clause is compared from left to right with the patterns. When there is a match, the matched substring is removed from the Where clause string, and the internal SQL format is supplemented according to the defined substitution. This proceeds until the Where clause string is empty. Patterns are applied in a pre-specified order. Patterns take the form of:

PATTERN[] SUBSTITUTION **

PATTERNS are similar to those in the prior pattern matcher accept that, since there has already been a pass through the prior pattern matcher, there is no need for the { } symbols or the [] phrase symbols. Here, [] and ** are simply symbols used to mark the different portions of the pattern and replacement. In addition to the !??x, !ENTx, !ATTx, !VALx, and !FUNCTIONx binding variables, there are also !NUM_CONSTRx which are numeric constraint

variables that match any numeric constraint (i.e. >, <, <=, >=, <>), and NUM_ATTx which match numeric columns.

SUBSTITUTION contains the elements to be added to the internal SQL structure, including SELECT tables, FROM table/alias pairs, WHERE clauses, JOINs. There are also several keywords used in the substitution section.

BIND !ENTx The pattern matcher successively matches the Where clause string against the patterns removing portions that have been matched and then matching the remainder. This binds the table held in the binding variable !ENTx to the variable LAST-ENTITY for use in matching the rest of the Where string.

LAST-ENTITY This contains the last table found in the most recently matched pattern prior to the current pattern match. Thus, this is the last table that is matched in the last pattern that was matched. A pattern can set what the LAST-ENTITY will be for the next matched pattern by using the BIND command. At the start of the pattern matcher, it is set to the last table processed by the SQL generator.

ADD_TO_SELECT !ATTx or !NUM_ATTx This specifies to add the column in the !ATTx or !NUM_ATTx variable to the SELECT clause of the internal SQL representation.

!ALIASx In SQL, the FROM clause defines the table from which information comes, and if there is more than one table it generally requires that an alias be assigned to the table for use by the other clauses. The general convention is for an alias to be of the form Tx where x is a number. For example a FROM clause will typically have the format "FROM Customer T1, Order T2" where T1 and T2 are aliases. The SELECT clause may then have the format "SELECT NAME.T1, ORDER_DOLLAR.T2". This prevents confusion if columns from different tables have the same names.

45

When !ALIASx is encountered in the substitution, an alias is generated for storage in the SQL structure. Since there are generally multiple aliases, x is a number. For each different x, a different alias is generated.

PKT !ENTx This returns either the table in !ENTx or the base table if !ENTx contains a virtual table defined in the conceptual layer.

PK !ENTx This returns the Primary Key of the table in !ENTx variable.

COL At this stage, columns stored in !ATTx and !NUM__ATTx variables are fully qualified in the form of table.column. COL !ATTx returns the column portion.

TABLE TABLE !ATTx or !NUM__ATTx returns the table portion of the fully qualified column name.

As an example, refer to the following pattern:

```

/* customers with orders of any product */
1 !ENT1 >= 1 !ENT2 [ ]
2 FROM PKT LAST-ENTITY !ALIAS1
3 WHERE EXISTS
4     SELECT*
5     FROM PKT!ENT2 !ALIAS2
6     WHERE EXISTS
7     SELECT *
8     FROM PKT !ENT1 !ALIAS3
9     JOIN !ENT2 !ALIAS2
10    JOIN LAST-ENTITY !ALIAS1
11 BIND !ENT2 **
    
```

Line 1 contains the pattern to match—it will match a string containing “Table_ref1 >=1 Table_ref2” where the table_refs are names of tables or virtual tables stored in the

46

conceptual layer. The string is put in this format during the prior pattern matching and substitution in step 404. Line 2 creates a FROM statement with the base table of the last table referenced by SQL Generator 20 before matching this pattern and create an alias. Line 5 creates a FROM clause with the base table of !ENT2 with an alias distinct from the prior alias. Line 8 is similar to lines 2 and 5. Lines 9 and 10 specify the Joins in internal SQL that will be required. The tables specified in lines 8 and 9 with their respective aliases need to be joined to the table specified in the FROM clause in line 8. Finally, in line 11, !ENT2 is bound as the LAST-ENTITY for any further pattern matching.

Therefore, if the clause being matched contains “ORDERS>=1 PRODUCTS” and the last table referenced as stored in LAST-ENTITY is CUSTOMERS, the resulting internal SQL format would contain:

```

FROM CUSTOMERS T1
WHERE EXISTS
20  SELECT*
    FROM PRODUCTS T2
    WHERE EXISTS
    SELECT*
    FROM ORDERS T3
25  JOIN PRODUCTS T2
    JOIN CUSTOMERS T1
    BIND PRODUCTS**
    
```

The set of patterns employed in step 416 of the illustrated embodiment (i.e., for one instance of an intermediate language) is shown below.

5,584,024

47

48

```
FROM CUSTOMERS T1
WHERE EXISTS
  SELECT *
  FROM PRODUCTS T2
  WHERE EXISTS
    SELECT *
    FROM ORDERS T3
    JOIN PRODUCTS T2
    JOIN CUSTOMERS T1
  BIND PRODUCTS **
```

10

Pattern 601

```

15 /* order date = january 1, 1993 */
   !ATT1 XDATE !VAL1 !VAL2 []
   FROM TABLE !ATT1 !ALIAS1
   WHERE !ALIAS1 . COL !ATT1 XDATE !VAL1 !VAL2 **

```

Pattern 602

```

20 /* balance between 100 and 500 */
   !NUM-ATT1 BETWEEN !VAL1 AND !VAL2 []
   FROM TABLE !NUM-ATT1 !ALIAS1
   WHERE !ALIAS1 . COL !NUM-ATT1 BETWEEN !VAL1 AND !VAL2 **

```

25

Pattern 603

```

   /* balance > 500 */
   !NUM-ATT1 !NUM-CONSTR1 !VAL1 []
   FROM TABLE !NUM-ATT1 !ALIAS1
30 WHERE !ALIAS1 . COL !NUM-ATT1 !NUM-CONSTR1 !VAL1 **

```

Pattern 604

```

   /* customers with orders of every product */
   !ENT1 WHERE EVERY !ENT2 []
35 FROM PKT !LAST-ENTITY !ALIAS1
   WHERE NOT EXISTS
     SELECT *
     FROM PKT !ENT2 !ALIAS2
     WHERE NOT EXISTS
40     SELECT *
     FROM PKT !ENT1 !ALIAS3
     JOIN !ENT2 !ALIAS2
     JOIN !LAST-ENTITY !ALIAS1
   BIND !ENT1 **
45

```

Pattern 605

/* customers with orders of any product */

!ENT1 >= 1 !ENT2 []
FROM PKT LAST-ENTITY !ALIAS1

5 WHERE EXISTS

SELECT *
FROM PKT !ENT2 !ALIAS2
WHERE EXISTS10 SELECT *
FROM PKT !ENT1 !ALIAS3
JOIN !ENT2 !ALIAS2
JOIN LAST-ENTITY !ALIAS1

BIND !ENT2 **

15 **Pattern 606**

/* salesmen that have at least 1 order */

>= 1 !ENT1 []
FROM PKT LAST-ENTITY !ALIAS1
WHERE EXISTS20 SELECT *
FROM PKT !ENT1 !ALIAS2
JOIN LAST-ENTITY !ALIAS1

BIND !ENT1 **

25 **Pattern 607**

/* salesmen that have at least 2 orders */

!NUM-CONSTR1 !VAL1 !ENT1 []
FROM PKT LAST-ENTITY !ALIAS1
WHERE REVERSE !NUM-CONSTR1 !VAL130 SELECT COUNT (*)
FROM PKT !ENT1 !ALIAS2
JOIN LAST-ENTITY !ALIAS1

BIND !ENT1 **

35 **Pattern 608**

/* customers that have every order_date since january 1 */

EVERY !ATT1 XDATE !VAL1 !VAL2 []
FROM PKT LAST-ENTITY !ALIAS1
WHERE NOT EXISTS40 SELECT *
FROM TABLE !ATT1 !ALIAS2
JOIN LAST-ENTITY !ALIAS1
WHERE NOT !ALIAS2 . COL !ATT1 XDATE !VAL1 !VAL2 **45 **Pattern 609**

/* salesmen that have every order_amount between 10 and 50 */

EVERY !ATT1 BETWEEN !VAL1 AND !VAL2 []
FROM PKT LAST-ENTITY !ALIAS1
WHERE NOT EXISTS50 SELECT *
FROM TABLE !ATT1 !ALIAS2
JOIN LAST-ENTITY !ALIAS1
WHERE !ALIAS2 . COL !ATT1 NOT BETWEEN !VAL1 AND !VAL2 **

Pattern 610

```

/* salesmen that have every order_amount > 50 */
EVERY !ATT1 !NUM-CONSTR1 !VAL1 []
FROM PKT LAST-ENTITY !ALIAS1
5 WHERE NOT EXISTS
  SELECT *
  FROM TABLE !ATT1 !ALIAS2
  JOIN LAST-ENTITY !ALIAS1
  WHERE !ALIAS2 . COL !ATT1 REVERSE-PROPER !NUM-CONSTR1 !VAL1 **
10

```

Pattern 611

```

/* salesmen that have every cust_name that
* sounds like smith */
EVERY !ATT1 THATSOUNDLIKE !??? []
15 FROM PKT LAST-ENTITY !ALIAS1
WHERE NOT EXISTS
  SELECT *
  FROM TABLE !ATT1 !ALIAS2
  JOIN LAST-ENTITY !ALIAS1
20 WHERE SOUNDEX ( !ALIAS2 . COL !ATT1 )
  <> SOUNDEX ( ' NOSPACESON !??? ' NOSPACESOFF ) **

```

Pattern 612

```

/* salesmen that have every cname that contains s */
25 EVERY !ATT1 THATCONTAIN !??? []
FROM PKT LAST-ENTITY !ALIAS1
WHERE NOT EXISTS
  SELECT *
  FROM TABLE !ATT1 !ALIAS2
30 JOIN LAST-ENTITY !ALIAS1
  WHERE NOT !ALIAS2 . COL !ATT1 LIKE ' NOSPACESON %
  !??? % ' NOSPACESOFF **

```

Pattern 613

```

35 /* salesmen that have every cname that begins with s */
EVERY !ATT1 THATBEGINWITH !??? []
FROM PKT LAST-ENTITY !ALIAS1
WHERE NOT EXISTS
  SELECT *
40 FROM TABLE !ATT1 !ALIAS2
  JOIN LAST-ENTITY !ALIAS1
  WHERE NOT !ALIAS2 . COL !ATT1 LIKE ' NOSPACESON !??? % '
  NOSPACESOFF **

```

Pattern 614

```

45 /* salesmen that have every cname that ends with s */
EVERY !ATT1 THATENDWITH !??? []
FROM PKT LAST-ENTITY !ALIAS1
WHERE NOT EXISTS
  SELECT *
50 FROM TABLE !ATT1 !ALIAS2
  JOIN LAST-ENTITY !ALIAS1
  WHERE NOT !ALIAS2 . COL !ATT1 LIKE ' NOSPACESON % !??? '
  NOSPACESOFF **
55

```

Pattern 615

```

/* salesmen that have every cname = smith */
EVERY !ATT1 = !??? []
FROM PKT LAST-ENTITY !ALIAS1
5 WHERE NOT EXISTS
  SELECT *
  FROM TABLE !ATT1 !ALIAS2
  JOIN LAST-ENTITY !ALIAS1
  WHERE !ALIAS2 . COL !ATT1 <> 'NOSPACESON !???'
10 NOSPACESOFF **

```

Pattern 616

```

/* every order where state = ct */
EVERY !ENT1 WHERE []
15 FROM PKT LAST-ENTITY !ALIAS1
  WHERE NOT EXISTS
    SELECT *
    FROM PKT !ENT1 !ALIAS2
    JOIN LAST-ENTITY !ALIAS1
20 BIND !ENT1 **

```

Pattern 617

```

/* salary > salary of at least 1 employee */
!NUM-ATT1 !NUM-CONSTR1 !NUM-ATT2 >= 1 !ENT1 []
25 FROM TABLE !NUM-ATT1 !ALIAS1
  WHERE ANY
    SELECT *
    FROM TABLE !NUM-ATT2 !ALIAS2
    WHERE !ALIAS1 . COL !NUM-ATT1 >
30 !ALIAS2 . COL !NUM-ATT2
  BIND !ENT1 **

```

Pattern 618

```

/* salary > salary of at least 6 employees */
!NUM-ATT1 !NUM-CONSTR1 !NUM-ATT2 !NUM-CONSTR2 !VAL1 !ENT1 []
35 FROM TABLE !NUM-ATT1 !ALIAS1
  WHERE REVERSE !NUM-CONSTR2 !VAL1
    SELECT COUNT (*)
    FROM TABLE !NUM-ATT2 !ALIAS2
40 WHERE !ALIAS1 . COL !NUM-ATT1 !NUM-CONSTR1
  !ALIAS2 . COL !NUM-ATT2
  BIND !ENT1 **

```

Pattern 619

```

/* salary > salary of the manager of that employee */
!NUM-ATT1 !NUM-CONSTR1 !NUM-ATT2 !ATT1 !ENT1 []
45 FROM TABLE !NUM-ATT1 !ALIAS1
  WHERE !NUM-ATT1 !NUM-CONSTR1 ALL
    SELECT !NUM-ATT2
50 FROM TABLE !NUM-ATT2 !ALIAS2
  WHERE !ALIAS1 . COL !ATT1 = !ALIAS2 . PK !ENT1
  BIND !ENT1 **

```

Pattern 620

```

/* salary > salary of employees [having name='smith'] */
!NUM-ATT1 !NUM-CONSTR1 !NUM-ATT2 WHERE []
55 FROM TABLE !NUM-ATT1 !ALIAS1
  WHERE !NUM-ATT1 !NUM-CONSTR1 ALL
    SELECT !NUM-ATT2
60 FROM TABLE !NUM-ATT2 !ALIAS2 **

```

Pattern 621

```

/* salary > salary employees [having name='smith'] */
!NUM-ATT1 !NUM-CONSTR1 !NUM-ATT2 !ENT1 []
FROM TABLE !NUM-ATT1 !ALIAS1
5 WHERE !NUM-ATT1 !NUM-CONSTR1 ALL
      SELECT !NUM-ATT2
      FROM TABLE !NUM-ATT2 !ALIAS2
      BIND !ENT1 **

```

Pattern 622

```

/* salary > salary of every employee [having state=ct] */
!NUM-ATT1 !NUM-CONSTR1 !NUM-ATT2 EVERY !ENT1 []
FROM TABLE !NUM-ATT1 !ALIAS1
15 WHERE !NUM-ATT1 !NUM-CONSTR1 ALL
      SELECT !NUM-ATT2
      FROM TABLE !NUM-ATT2 !ALIAS2
      BIND !ENT1 **

```

Pattern 623

```

20 /* salary > average salary of employees [having name='smith'] */
!ATT1 !NUM-CONSTR1 !!FUNCTION1 ( !ATT2 ) !ENT1 []
FROM TABLE !ATT1 !ALIAS1
WHERE !ATT1 !NUM-CONSTR1
      SELECT !!FUNCTION1 ( !ATT2 )
25 FROM TABLE !ATT2 !ALIAS2
      BIND !ENT1 **

```

Pattern 624

```

30 /* salary > average salary of all employees [having state = ct] */
!ATT1 !NUM-CONSTR1 !!FUNCTION1 ( !ATT2 ) EVERY !ENT1 []
FROM TABLE !ATT1 !ALIAS1
WHERE !ATT1 !NUM-CONSTR1
      SELECT !!FUNCTION1 ( !ATT2 )
      FROM TABLE !ATT2 !ALIAS2
35 BIND !ENT1 **

```

Pattern 625

```

/* salesman that have the same state */
SAME !ATT1 []
40 FROM TABLE !ATT1 !ALIAS1
WHERE !ALIAS1 . COL !ATT1 IN
      SELECT !ATT1
      FROM TABLE !ATT1 !ALIAS2
      WHERE !ALIAS1 . PK LAST-ENTITY <>
45 !ALIAS2 . PK !LAST-ENTITY **

```

Pattern 626

```

/* salesmen that have no orders */
NO !ENT1 []
50 FROM PKT LAST-ENTITY !ALIAS1
WHERE NOT EXISTS
      SELECT *
      FROM PKT !ENT1 !ALIAS2
      JOIN LAST-ENTITY !ALIAS1
55 BIND !ENT1 **

```

Pattern 627

```

/* balance > 500 */
!NUM-ATT1 !NUM-CONSTR1 !VAL1 []
60 FROM TABLE !NUM-ATT1 !ALIAS1
WHERE !ALIAS1 . COL !NUM-ATT1 !NUM-CONSTR1 !VAL1 **

```

Pattern 628

```

/* balance > credit limit */
!NUM-ATT1 !NUM-CONST1 !NUM-ATT2 []
FROM TABLE !NUM-ATT1 !ALIAS1
5 WHERE !NUM-ATT1 !NUM-CONST1 !NUM-ATT2 **

```

Pattern 629

```

/* balance > (credit limit * 10) */
!NUM-ATT1 !NUM-CONST1 !COMP1 []
10 FROM TABLE !NUM-ATT1 !ALIAS1
WHERE !NUM-ATT1 !NUM-CONST1 !COMP1 **

```

Pattern 630

```

/* (balance*5) > (credit limit * 10) */
15 !COMP1 !NUM-CONST1 !COMP2 []
WHERE !COMP1 !NUM-CONST1 !COMP2 **

```

Pattern 631

```

/* balance > sum(order$) */
20 !NUM-ATT1 !NUM-CONST1 !!FUNCTION1 ( !NUM-ATT2 ) []
FROM TABLE !NUM-ATT1 !ALIAS1
WHERE !NUM-ATT1 !NUM-CONST1 !!FUNCTION1 ( !NUM-ATT2 ) **

```

Pattern 632

```

25 /* sum(order$) > balance */
!!FUNCTION1 ( !NUM-ATT1 ) !NUM-CONST1 !NUM-ATT2 []
FROM TABLE !NUM-ATT1 !ALIAS1
WHERE !!FUNCTION1 ( !NUM-ATT1 ) !NUM-CONST1 !NUM-ATT2 **

```

Pattern 633

```

30 /* sum(order$) > avg(freight) */
!!FUNCTION1 ( !NUM-ATT1 ) !NUM-CONST1 !!FUNCTION2 ( !NUM-ATT2 ) []
FROM TABLE !NUM-ATT1 !ALIAS1
WHERE !!FUNCTION1 ( !NUM-ATT1 ) !NUM-CONST1 !!FUNCTION2
35 ( !NUM-ATT2 ) **

```

Pattern 634

```

/* customer names that sound like ab */
40 !ATT1 THATSOUNDLIKE !??? TESTSOUNDEX []
FROM TABLE !ATT1 !ALIAS1
WHERE SOUNDEX ( !ATT1 )
= SOUNDEX ( ' NOSPACESON !??? ' NOSPACESOFF ) **

```

Pattern 635

```

45 /* customer names that contain ab */
!ATT1 THATCONTAIN !??? []
FROM TABLE !ATT1 !ALIAS1
WHERE !ALIAS1 . COL !ATT1 LIKE ' NOSPACESON % !??? % ' NOSPACESOFF **

```

Pattern 636

```

50 /* customer names that begin with ab */
!ATT1 THATBEGINWITH !??? []
FROM TABLE !ATT1 !ALIAS1
WHERE !ALIAS1 . COL !ATT1 LIKE ' NOSPACESON !??? % ' NOSPACESOFF **
55

```

Pattern 637

```

/* customer names that end with yz */
!ATT1 THATENDWITH !??? []
FROM TABLE !ATT1 !ALIAS1
60 WHERE !ALIAS1 . COL !ATT1 LIKE ' NOSPACESON % !??? ' NOSPACESOFF **

```


Pattern 638

```
/* cust.cnum = ord.cnum */
!ATT1 !NUM-CONSTR1 !ATT2 []
FROM TABLE !ATT1 !ALIAS1
5 WHERE !ATT1 !NUM-CONSTR1 !ATT2 **
```

Pattern 639

```
/* state = ct */
!ATT1 = !??? []
10 FROM TABLE !ATT1 !ALIAS1
WHERE !ALIAS1 . COL !ATT1 = ' NOSPACESON !??? ' NOSPACESOFF **
```

Pattern 640

```
/* ( ytd_sales - ytd_cost ) between 100 and 500 */
!COMP1 BETWEEN !VAL1 AND !VAL2 []
15 WHERE !COMP1 BETWEEN !VAL1 AND !VAL2 **
```

Pattern 641

```
/* ( ytd_sales - ytd_cost ) > 500 */
20 !COMP1 !NUM-CONSTR1 !VAL1 []
WHERE !COMP1 !NUM-CONSTR1 !VAL1 **
```

Pattern 642

```
/* customer number = 100 */
25 !ALPHA-ATT1 !NUM-CONSTR1 !VAL1 []
FROM TABLE !ALPHA-ATT1 !ALIAS1
WHERE !ALIAS1 . COL !ALPHA-ATT1 !NUM-CONSTR1 ' NOSPACESON !VAL1 '
NOSPACESOFF **
```

Pattern 643

```
/* customer names > aa */
!ATT1 !NUM-CONSTR1 !??? []
FROM TABLE !ATT1 !ALIAS1
WHERE !ALIAS1 . COL !ATT1 !NUM-CONSTR1 ' NOSPACESON !??? ' NOSPACESOFF **
35
```

Pattern 644

```
/* names that are null */
!ATT1 NULL []
FROM TABLE !ATT1 !ALIAS1
40 WHERE !ALIAS1 . COL !ATT1 IS NULL **
```

Pattern 645

```
/* names that are not null */
!ATT1 NOT NULL []
45 FROM TABLE !ATT1 !ALIAS1
WHERE NOT ( !ATT1 IS NULL ) **
```

Pattern 646

```
/* same as product 100 */
50 = !VAL1 []
WHERE PK LAST-ENTITY = !VAL1 **
```

Pattern 647

```
/* where salesman have */
55 !ENT1 WHERE []
BIND !ENT1 **
```

Pattern 648

```
/* of salesmen */
60 OF-ENTITY!! !ENT1 WHERE []
BIND !ENT1 **
```

When SQL Generator 20 is initiated, the patterns are read from an external text file. The patterns are stored in a structure which, for each pattern contains, the pattern string and the substitution string. Construction and operation of binding pattern marchers are well known in the art and any of the many techniques that provide the aforementioned capabilities is within the scope of this invention. The pattern marcher is recursively called when it encounters nested Where clauses in the case of parentheticals.

C. Join Path

Steps 432-436 call for the computation of join paths, the addition of any new tables to the FROM clause, and inclusion of the explicit joins in the WHERE clause. The computation of the join paths will produce the shortest join path between two tables unless the administrator has defined alternate join paths in the conceptual layer for the user to choose from. With a database structure as shown in FIG. 6, where the direction of the arrows show primary key -> foreign key relationships, the shortest join path can be readily computed as follows.

First, a table of primary key tables is constructed, foreign key tables following all primary key -> foreign key links, the next table in the join chain if not the foreign key table, and the number of joins it takes to get from the primary key table to the foreign key table. This table can be constructed using the foreign key information from the conceptual layer, or by querying the user as to the relationships among the tables.

Second, the navigable paths are computed for the tables to be joined by following the primary key -> foreign key pairs in the table. The Least Detailed Table (LDT) common to the primary key -> foreign key paths of the two tables to join is then found. The LDT is the table up on the graph. In a one-to-many relationship, the one is the least detailed table. If through multiple paths, there are multiple LDTs, the table where the sum of the number of joins is the least is selected. If the number of hops from table to table is equal, it is particularly appropriate for the administrator to define the join paths for user selection. If nothing is defined, one of the paths is arbitrarily chosen. Finally, the join path can be computed by following the primary key -> foreign key relations down to the LDT and then, if necessary, backwards following foreign key -> primary key back up to the second table of the join if neither table is the common LDT.

Using the above procedure, the following table can be constructed for the database of FIG. 6.

Primary table	Foreign table	Next Table	Number of Joins
SALESPEOPLE	CUSTOMERS		1
SALESPEOPLE	ORDERS		1
SALESPEOPLE	LINE_ITEMS	ORDERS	2
CUSTOMERS	ORDERS		1
CUSTOMERS	LINE_ITEMS	ORDERS	2
ORDERS	LINE_ITEMS		1
VENDORS	PRODUCTS		1
VENDORS	LINE_ITEMS	PRODUCTS	2
CODES	PRODUCTS		1
CODES	LINE_ITEMS	PRODUCTS	2
PRODUCTS	LINE_ITEMS		1

To find the join path from SALESPEOPLE to ORDERS given the above table, the navigable paths are first computed. For SALESPEOPLE, there are two navigable paths, [SALESPEOPLE CUSTOMER ORDERS LINE_ITEMS] and [SALESPEOPLE ORDERS LINE_ITEMS]. For ORDERS, there is one path [ORDERS LINE_ITEMS]. The common LDT for SALESPEOPLE and ORDERS using

either of the paths found for SALESPEOPLE is ORDERS. since there are two paths from SALESPEOPLE to ORDERS, we calculate the number of hops to be one using the [SALESPEOPLE ORDERS] path and two using the [SALESPEOPLE CUSTOMERS ORDERS] path. Without any path definitions in the conceptual layer, the SQL generator will use the shorter path.

As another example, to find the join path from ORDERS to PRODUCTS, the navigable paths are first computed in the same way. This yields the path [ORDERS LINE_ITEMS] for ORDERS, and [PRODUCTS LINE_ITEMS] for PRODUCTS. The common LDT for these paths is LINE_ITEMS. Following the table from ORDERS to LINE_ITEMS and then back up to PRODUCTS, the join path [ORDERS LINE_ITEMS PRODUCTS] is computed. This technique is one of several well know in the art and calculation of the join path is not limited to this technique in the present invention.

In the last example above, the LINE_ITEMS table is introduced in creating the join path. Step 436 adds any new tables introduced in the process of calculating the join path to the FROM clause in the internal SQL structure. Also included is an alias for the new table.

SQL requires the joins to be explicitly provided in the WHERE clause, and step 436 implements this. The primary and foreign key columns are stored in the conceptual layer either by the administrator or by Query System 1 after querying the user. Using the information, the following statement can be included in the WHERE clause to express the join of the above example if the alias for ORDERS, PRODUCTS and LINE_NUMBERS is T1, T2, and T3: "WHERE T1.PRODUCT# = T2.PRODUCTS# AND T2.PRODUCT# = T3.PRODUCT#"

D. Example Conversion of the Intermediate Language to SQL

The example below shows the steps in the conversion of a query in the intermediate language of the form "SHOW CUSTOMER NAME FOR CUSTOMERS THAT HAVE ORDERS OF ANY PRODUCT SORTED BY CUSTOMER CITY" to SQL code. First, in step 402, the query is tokenized into individual units, here marked by < >

```
<SHOW> <CUSTOMER NAME> <FOR> <CUSTOMERS>
<THAT HAVE> <ORDERS> <OF ANY> <PRODUCT>
<SORTED BY> <CUSTOMER CITY>
```

These are the words and phrases made into tokens. This distinction continues, but for purposes of the following steps, the < > around the tokens are not included. Next, in steps 404 and 406, the query is applied against the first set of patterns. The above query matches patterns 512, 556, 559, 571, and 581.

The patterns are applied in order of priority first and then order of location in the external text file. Since all patterns are either priority 2 or 5, the order in which the patterns above are listed are the order in which they are applied. Therefore, pattern 512 is applied, and the query becomes:

```
SHOW CUSTOMERS.NAME OFENTITY!! CUSTOMERS
WHERE ORDERS OF ANY PRODUCTS SORTED BY CUS-
TOMERS.CITY
```

The OFENTITY!! keyword is later used in converting to the internal SQL format and indicates that CUSTOMER.S.NAME is a column of entity (table) CUSTOMERS. After the last pattern is applied, patterns 556 and 559 no longer match. Also, no new patterns match so patterns 571 and 581 remain. By Applying pattern 571, which has a higher priority, the query becomes:

```
SHOW CUSTOMERS.NAME OFENTITY!! CUSTOMERS
WHERE ORDERS OF ANY PRODUCTS ORDERBY CUS-
TOMERS.CITY
```

No new patterns are matched, and there is only one more pattern to match which, when applied, yields:

```
SHOW CUSTOMERS.NAME OFENTITY!! CUSTOMERS
WHERE ORDERS >=1 PRODUCTS ORDERBY CUSTOM-
ERS. CITY
```

Steps 408–412 are not applicable to this query, since there is no CREATE VIEW command needed for this type of query. If it were one of a specific set of queries which require CREATE VIEW SQL syntax, SQL Generator 20 would be called recursively to create the views. Since CREATE VIEW is not necessary, no new words or phrases for conversion were introduced.

In step 414, the query is broken into SQL components. The query then becomes:

```
SELECT CUSTOMERS.NAME WHERE ORDERS>=1 PROD-
UCTS ORDER BY CUSTOMERS.CITY
```

The LAST-ENTITY variable is set to CUSTOMERS, since the last table added to the select clause is from the table CUSTOMERS. The OFENTITY!! keyword introduced in the last pattern match is helpful in determining the LAST-ENTITY.

In steps 416–418, the Where clause “ORDERS>=1 PRODUCTS” is applied to the patterns shown above, resulting in one match, with pattern 605. By applying this pattern, the internal SQL structure for the query becomes:

```
SELECT CUSTOMERS.NAME
FROM CUSTOMERS T1
WHERE EXISTS
  SELECT *
  FROM PRODUCTS T2
  WHERE EXISTS
    SELECT *
    FROM ORDERS T3
    JOIN PRODUCTS T2
    JOIN CUSTOMERS T1
ORDER BY CUSTOMERS.CITY
```

The LAST-ENTITY variable is assigned the table PRODUCTS.

In step 420, if there were any table names in the SELECT portion it would expand to include all of the tables columns. Also any virtual table would be expanded. Neither are present in this example, but are performed by simple substitution.

In step 422, any columns in the SORT BY portion are added to SELECT if not present. This step converts the SELECT portion of the internal SQL to:

```
SELECT CUSTOMERS.CITY, CUSTOMERS.NAME
```

The date conversion function of step 424 is not applicable, since there are no dates in this example. Similarly, there are no virtual columns for expansion in step 426.

If any aliases need to be specified to the FROM clause, they are made in step 428. This query created the alias during the application of the Where rules, and no other tables were added to the from clause. The aliases are then substituted into the other sections as well. The internal SQL becomes:

```
SELECT T1.CITY T1.NAME
FROM CUSTOMERS T1
WHERE EXISTS
  SELECT *
  FROM PRODUCTS T2
  WHERE EXISTS
    SELECT *
    FROM ORDERS T3
    JOIN PRODUCTS T2
    JOIN CUSTOMERS T1
ORDER BY T1.CITY
```

In step 430, the ORDER BY clause is converted to:

```
ORDER BY 1
```

In step 432, required joins are computed from the internal SQL. They are represented here by the “JOIN Table Alias” statement, and indicates that those tables need to join the table listed in the FROM clause above it. From the prior discussion on join path calculations, the join paths created from the statements:

```
FROM ORDERS T3
JOIN PRODUCTS T2
JOIN CUSTOMERS T1
```

are [CUSTOMERS ORDERS] and [ORDERS LINE_ITEMS PRODUCTS].

Then, in steps 434 and 436, since the join path calculation introduced a new table, LINE_ITEMS, the table needs to be added to the FROM clause with an alias to make:

```
FROM ORDERS T3, LINE_ITEMS T4
```

In step 438, the joins are created and added to the where clause from the join paths and foreign key information in the conceptual layer to produce the following Where clause:

```
WHERE T3.ORDER#=T4.ORDER#
AND T2.PRODUCT#=T4.PRODUCT#
AND T1.CUSTOMER#=T3.CUSTOMER#
```

Steps 440–444 are not applicable to this example. Finally, in step 446, the internal SQL structure, which is now represented as:

```
SELECT T1.CITY T1.NAME
FROM CUSTOMERS T1
```

```
WHERE EXISTS
  SELECT *
  FROM PRODUCTS T2
  WHERE EXISTS
    SELECT *
    FROM ORDERS T3.LINE_ITEMS T4
    WHERE T3.ORDER# = T4.ORDER#
    AND T2.PRODUCT# = T4.PRODUCT#
    AND T1.CUSTOMER# = T3.CUSTOMER#
ORDER BY 1
```

is converted to textual SQL. The above representation of the internal SQL structure is in proper textual structure for a query. The process of conversion to the textual query from the internal structure is a trivial step of combining the clauses and running through a simple parser.

What is claimed is:

1. A database query system for interactively creating, with a user, a syntactically and semantically correct query for a relational database having a plurality of tables, each of said tables having a plurality of columns and having a predetermined relationship to another of said tables, said system comprising:
 - a conceptual layer manager for storing conceptual information about the relational database, said conceptual information including structural information concerning the identity of each of the tables and columns and the directionality and cardinality of the relationships between the tables;
 - a query assistant user interface ("QAUI") presenting to the user a selectable table set of selectable tables from among the tables in the database, a selectable column set of selectable columns from among the columns of each of said tables in the database, and a selectable column operations set of selectable column operations on the columns, from which the user may select tables, columns, and column operations to construct a database query for said database in an intermediate query language, said QAUI further accepting from the user selections of tables, columns, and column operations;
 - a query assistant expert ("QAES") coupled to said QAUI to receive from said QAUI the identity of each table, column, or column operation selected by the user, said QAES returning to the QAUI after each selection by the user an updated version of said selectable table set, said selectable column set, and said selectable column operations set, said QAES excluding from said selectable sets any table, column, or column operation which, if selected by the user, would, based on the then-current state of the database query and said conceptual information, produce a semantically incorrect query.
2. The database query system of claim 1 wherein said QAES includes:
 - a storage system for maintaining state information about the current state of a database query; and
 - a query expert logic system specifying to said QAUI said selectable sets by analyzing said state information maintained in said storage system and said conceptual information stored by said conceptual layer manager.
3. A database query system according to claim 2 wherein said storage system includes:
 - a set of state variables; and
 - a set of access routines for adding deleting and modifying said state variables.
4. A database query system according to claim 2 wherein said storage system includes:
 - a state database, said state database containing said state information; and
 - a set of database access routines for adding to, deleting from and modifying said state database.
5. A database query system according to claim 2 wherein said query expert logic system is composed of procedural logic.
6. A database query system according to claim 2 wherein said query expert logic system is a rule-based expert system.
7. A database query system according to claim 2 wherein said conceptual information further comprise one or more of the following: foreign keys, table join paths, table join expression for non-equijoins, virtual table definitions, virtual column definitions, table descriptions, column descriptions, hidden tables, and hidden columns.
8. A database query system according to claim 2 wherein said conceptual information further comprises table join expression for non-equijoins.

9. The database query system of claim 2 wherein if said current state of said database query includes an aggregate column operation on a column in a first table, said query expert logic system excludes from said selectable table set any other of said tables that is more detailed than said first table or is joinable with said first table only through another more detailed table.

10. The system of claim 2 wherein the database includes at least four tables and wherein if in the then-current state of said database query two of the tables are selected, said query expert logic system excludes from said selectable table set any other of said tables that does not form in combination with said two selected tables a navigable set.

11. The system of claim 2 wherein said query expert logic system excludes from said selectable column set for any selected one of said tables any numeric column for which, if in said current state of said database query an aggregate column operation is applied to another column based on said selected table or based on another table having a one-to-one relationship with said selected table.

12. A database query system according to claim 2 wherein said conceptual information further comprises virtual column definitions.

13. A database query system according to claim 12 wherein said virtual column definition contains primary key and foreign key references to define a join operation.

14. A database query system according to claim 1 wherein said each of said selectable table set, said selectable column set, and said selectable column operation set is mutually exclusive to a corresponding nonselectable table set, nonselectable column set, and nonselectable column operation set and is a subset of all tables, columns, and column operations, respectively, maintained by said conceptual layer manager which the user may next select in building a semantically correct database query.

15. A database query system according to claim 14 wherein said QAUI indicates to the user said selectable sets set of permissible selections and not displaying said nonselectable sets.

16. A database query system according to claim 14 wherein said QAUI displays said nonselectable sets and visually differentiates said selectable sets from said nonselectable sets.

17. A database query system according to claim 16 wherein said QAUI visually differentiates said sets by color.

18. A database query system according to claim 16 wherein said QAUI visually differentiates said selectable and nonselectable sets by type characteristic.

19. The database query system of claim 1 further comprising a query generator coupled to said QAUI to receive from said QAUI a completed database query in said intermediate query language, said query generator converting said query from said intermediate query language into a target query language different from said intermediate query language.

20. A database query system according to claim 19 wherein said target query language is Structured Query Language (SQL).

21. A database query system according to claim 19 wherein said query generator converts said intermediate language query into said target language by a set of successive transformations.

22. A database query system according to claim 21 wherein at least one of said set of successive transformations is transformation by pattern substitution.

23. A database query system according to claim 21 wherein said set of transformations comprises:

69

a set of structural transformations;
a set of transformation to include inferred information;
and
a set of transformations by pattern substitution.

24. A method for interactively building a syntactically and semantically correct query of a relational database from selections by a user, said database having a plurality of tables, each of said tables having a plurality of columns and having a predetermined relationship to another of said tables, said method comprising the steps of:

presenting to the user a selectable table set of selectable tables from among the tables in the database;

receiving from the user a selection of a first one of said selectable tables;

presenting to the user a selectable column set of selectable columns from among the columns based on said first selected table;

receiving from the user a selection of a first selected column from among said selectable columns;

presenting to the user a selectable column operation set of selectable column operations applicable to said first selected column;

receiving from the user a selection of one of said column operations;

presenting to the user a first updated version of said selectable table set from which the user may select a second selected table, said selectable table set excluding any table that is more detailed than said first selected table on which said selected column operation is applied or that is joinable with said first selected table

70

only through a more detailed table if said selected column operation is an aggregate operation.

25. The method of claim 24 wherein said first updated version of said selectable table set further excludes any table that is not joinable with said first selected table.

26. The method of claim 25 wherein the database includes at least four tables and further comprising the steps of:

receiving from the user a selection of a second selected table from said first updated version of said selectable table set; and

presenting to the user a second updated version of said selectable table set from which the user may select a third selected table, said selectable table set excluding any table the selection of which does not form in combination with said first and second selected tables a navigable set.

27. The method of claim 26 further comprising the steps of:

receiving from the user a selection of a second selected table from said first updated version of said selectable table set; and

presenting to the user an updated version of said selectable column set based on said second selected table, said selectable columns set excluding any column that contains non-numeric information if said second selected table is the same as said first selected table or has a one-to-one relationship with said first selected table.

* * * * *

MVIEWS: Multimodal Tools for the Video Analyst

Adam Cheyer
Artificial Intelligence Center
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
+1 415 859 4119
cheyer@ai.sri.com

Luc Julia
STAR Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
+1 415 859 4269
julia@speech.sri.com

ABSTRACT

Full-motion video has inherent advantages over still imagery for characterizing events and movement. Military and intelligence analysts currently view live video imagery from airborne and ground-based video platforms, but few tools exist for efficient exploitation of the video and its accompanying metadata. In pursuit of this goal, SRI has developed MVIEWS, a system for annotating, indexing, extracting, and disseminating information from video streams for surveillance and intelligence applications. MVIEWS is implemented within the Open Agent Architecture, a distributed multiagent framework that enables rapid integration of component technologies; for MVIEWS, these technologies include pen and voice recognition and interpretation, image processing and object tracking, geo-referenced interactive maps, multimedia databases, and human collaborative tools.

Keywords

Multimodal pen and voice user interfaces, image processing and object tracking, video analysis and annotation, agent architecture.

INTRODUCTION

Although sophisticated tools are now starting to appear that assist an image analyst in manipulating still photos, few systems exist to help an operator efficiently exploit full-motion video. Given video's inherent advantages for characterizing events and movement in a scene, the role of video analysis is taking on increased importance in military, intelligence, and surveillance domains. By considering how video can be best exploited in these contexts, we realize that video analysis poses new challenges and opportunities:

Permission to make digital/hard copies of all or part of this material for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copyright is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires specific permission and/or fee.

IUI 98 San Francisco CA USA
Copyright 1998 ACM 0-89791-955-6/98/ 01..\$3.50

- At the 1996 Atlanta Olympics, after a bomb went off inside of Olympic Park, investigators had to deal with the task of thoroughly searching for clues within some 600 amateur videos related to the incident. Had there been better tools available for indexing, time stamping, annotating, classifying and cross-referencing the videos, this process would have been much more manageable.
- The U.S. military, as part of peacekeeping and intelligence missions, routinely sends unmanned Predator airplanes over target sites of interest. While pilots remotely guide the airplane over the terrain, a second team of analysts is responsible for extracting information of relevance from the video and associated metadata. Although all results of these missions are recorded on videocassette, there is currently no automated method for querying this data repository at a later date. A searchable index would help ensure that this resource is not wasted, and providing the ability to replay audio and written annotations would help the analysts reviewing a video to quickly establish context for what they are seeing.
- In many surveillance or security-related tasks, a single operator is responsible for monitoring the output of many cameras distributed throughout the site. Object detection and tracking, in conjunction with automated alerts and sensor management, can augment the operator's ability to efficiently comprehend and fuse numerous information streams.

In this paper, we present a demonstration system called MVIEWS, for Multimodal Video Imagery Exploitation WorkStation, that attempts to address some of these challenges by bringing together multiple commercial and research technologies into a single toolset. Although each technology is interesting by itself, it is the *integrated use* of these capabilities that can greatly enhance the effectiveness of a video analyst.

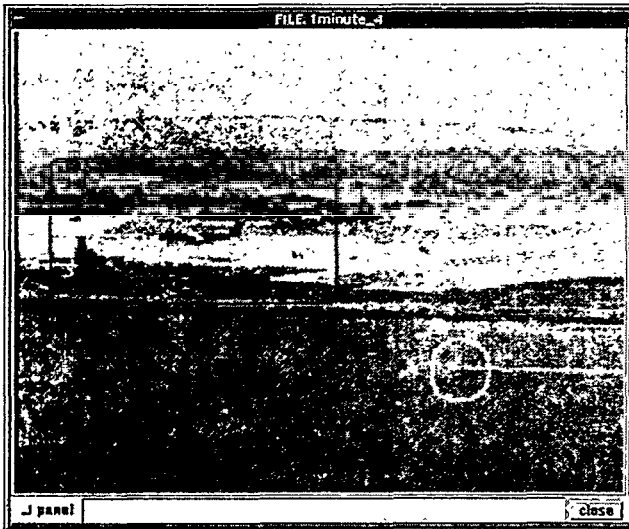


Figure 1. Video Player. Moving objects in surveillance regions are tracked (plane). Multimodal commands can target specific objects (boat).

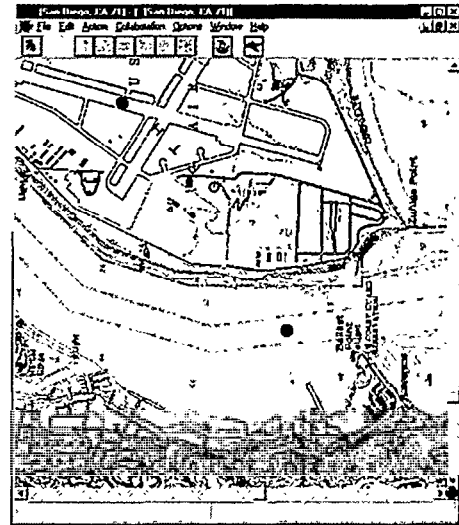


Figure 2. Multimodal Map. Objects tracked in a video are simultaneously displayed on a geo-referenced map.

SYSTEM DESCRIPTION AND DESIGN

Design Approach

The original design for MVIEWES was based on a vision and on a set of guiding principles. The vision can be described as follows.

Imagine a single operator at a terminal, exploiting ten ground and aerial video sensors. Software agents are providing real-time object detection, alerts, queuing, tracking, and information fusion. The operator uses only voice commands and an electronic pen to control the workstation and sensors, to add multimodal annotations to the video streams, and to collaborate with operations and intelligence specialists at remote locations. As a situation develops, agents and humans work together on assessment and characterization, while documenting the process through semiautomatically generated reports.

When setting out in pursuit of this vision, we tried to also keep in mind a few frequently overlooked design criteria:

- People are indispensable. They are good at processing complex visual problems, but they are subject to fatigue and boredom. Automation is an aid, not a replacement.
- User interfaces for complex tasks can quickly become complex. We needed to create as natural, invisible and efficient an interface as possible, combining graphical user interface (GUI) techniques when effective with other, more fluid modalities, such as speech and pen.
- The utility of information greatly increases in conjunction with other supporting information. Thus,

we required an open and extensible system that places the needed information and tools at the operator's fingertips.

System Description

The first public demonstration of MVIEWES was given at the Exploitation Technology Symposium (ETS-97) held at Naval Research and Development (NRAD) headquarters in San Diego. Describing this event will provide a good overview of how the current MVIEWES implementation can be used.

Given the visuals provided by NRAD's location, we chose to exhibit MVIEWES using a border patrol scenario. After securing the necessary permits, we installed a camera on the roof of the demonstration building, such that it overlooked activity in the harbor below and at a nearby military airport. Our video analyst could investigate the movements of small recreational boats, an occasional commercial or military ship, plenty of windsurfers, and a few airplanes and helicopters.

Inside the exposition's demonstration facilities, an operator was seated in front of a Sun¹ Ultrasparc 1 computer, monitoring the live video feed. The operator interacts with MVIEWES by using pen and voice (Figure 1), to perform one of the following functions:

- *Add annotations*, simply by speaking and drawing directly on top of the video. As an example, a surveillance operator might speak "The movements of

¹ All product or company names mentioned in this document are the trademarks of their respective holders.

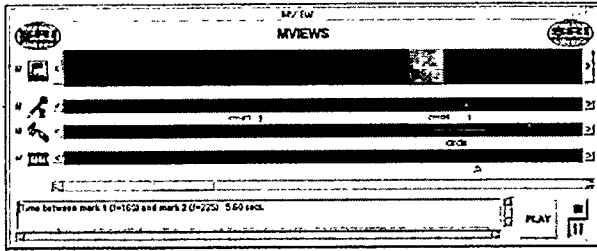


Figure 3. Media Track Editor. A video is replayed with multimedia overlays, and indexed fields are located in the video or the video database.

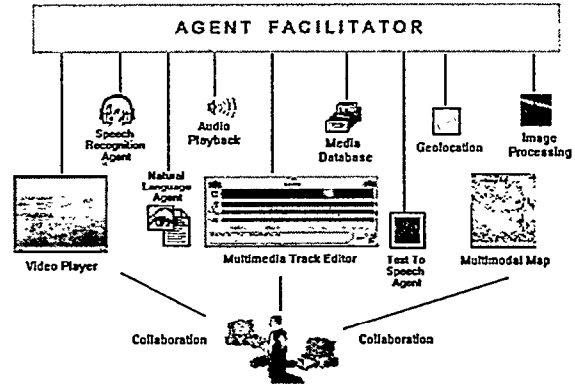


Figure 4. MVIEWs architecture.

this motorboat appear unusual” while drawing a circle around the vessel and tracing its path using the pen.

- *Generate reports*, constructed from multimodal input and multimedia data. A typical interaction might contain: “Report, convoy of three vehicles, heading rapidly west along access road.” This information would be saved with the video frame and associated metadata, including time, date, camera type, and spatial coordinates.
- *Specify commands* to the system, involving object tracking (e.g., “Track this boat”), image processing (e.g., “Stabilize image”, “Grab this region”), and setting alerts (e.g., “Notify me if this object moves” or “If more than three objects enter this region, alert me”).
- *Collaborate* with remote participants, for example “Bob, can you identify this vehicle type?”

In addition to the video display, the operator could call up an interactive map (Figure 2), simultaneously displaying any objects tracked in the video as geo-referenced points in 2D space. The map display, also controlled through pen and voice, provided additional information about the region. For example, in the Predator UAV domain, the map allows the operator to examine the terrain ahead of the plane’s path, and call up supplementary data (e.g., “Show me all military bases near here”).

As a means of attracting further attention to our ETS demonstration, we sent out a person carrying a wireless handheld pen computer to mingle with a reception gathering nearby. After targeting a small group, the demonstrator would show them the map display, look over the balcony and say, “See that boat down there? It’s being automatically tracked by software agents from a live video image, and this computer is receiving the reports. Why don’t you go in there and check it out?”

Near the first workstation, we positioned another computer where a second analyst could work collaboratively with the

first. While one operator monitored and annotated the continuously advancing live video feed, the second was at liberty to provide more detailed analyses of recorded segments of the video. Using the Media Track Editor (Figure 3) as his primary interface, the analyst navigated within the video segment, requested image enhancements, performed timing and distance measurements, and queried the multimedia database for other video segments of interest. The Media Track Editor is structured as a timeline, with annotations, extracted frames and clips, and recognized speech and pen clearly highlighted. By providing an interface for quickly skimming through a video clip and replaying selected sections along with their multimodal annotations, MVIEWs establishes context for what the analyst is examining, accenting what was important to the operator at the time the video was recorded. Instead of fast forwarding through the entire video clip to understand its content, an analyst can save time by perusing annotations from the timeline at different granularities of detail.

IMPLEMENTATION

To implement MVIEWs, we needed to quickly integrate numerous technologies, written in a variety of programming languages, some requiring specialized computer hardware. To facilitate a loosely coupled, dynamic, heterogeneous and distributed integration, we took advantage of the services provided by the Open Agent Architecture™ (OAA™).²

The Open Agent Architecture

Similar in objective to distributed object frameworks such as OMG’s CORBA or Microsoft’s DCOM, a distributed agent architecture such as the OAA can provide integration of components written in different

² For more information about the OAA, see <http://www.ai.sri.com/~oaa/>

programming languages³ and running on different platforms.⁴ However, OAA agents possess qualities beyond ordinary distributed objects. Agent interactions are more flexible and adaptable than tightly bound IDL⁵ method calls in CORBA or DCOM, and are able to take advantage of parallelism and dynamic execution of complex goals. Instead of preprogrammed unitary method calls to known object services, an agent can express its requests in terms of a high-level logical description of what it wants done, along with optional constraints specifying how the task should be performed. This information is processed by one or more Facilitator agents, which plan, execute and monitor the coordination of the subtasks required to accomplish the end goal.

The OAA has been used to implement more than twenty different applications, including

- Multi-robot control and coordination [4]
- Office automation and unified messaging [2]
- Collaborative multimodal user interfaces [1, 12]
- Frontends [8] and backends [11] for the Web
- Development tools [10] for creating and assembling new agents with the OAA

Each OAA project can take advantage of the core services provided by the architecture as well as of the growing number of technologies now accessible through an agent interface. These services and technologies include speech recognition, natural language understanding, text extraction, multimodal fusion and reference resolution, reactive planning, virtual reality, image processing, web-related technologies, user modeling, and collaboration tools.

The core services of the OAA are implemented by an agent library, which has been ported to several different programming languages, working closely with a Facilitator agent, responsible for domain-independent coordination and routing of information and goals. These basic services can be classified into three areas: agent communication and cooperation, distributed data services, and trigger management.

Interagent Communication Language

The Interagent Communication Language (ICL) provides the means for interaction among agents. When an agent

³ Programming languages: C, C++, Prolog, Lisp, Java, Borland Delphi, and Microsoft Visual Basic.

⁴ Platforms: UNIX (SunOs, Solaris, Lynx), Windows (3.1, 95, NT), all Java platforms.

⁵ Interface Definition Language: specifies an object's methods using a C++-like header file.

wants to make a request of the agent community, it describes the goal it wants achieved as well as parameters specifying constraints on how the goal is to be accomplished. The request is sent to a Facilitator agent, which uses the declarative specifications it stores about each agent's capabilities, and the parameters defined for the incoming goal, to produce a fully specified execution plan detailing tasks for distributed agents to perform. The Facilitator agent is then responsible for monitoring and coordinating the execution of the plan, by routing requests (potentially to agents in parallel), collecting results, backtracking when subgoals fail, and finally providing the results to the requesting agent.

ICL requests are expressed using the syntax and semantics of Prolog, a decision influenced by our desire to involve the user as closely as possible in agent interactions. ICL expressions can be generated from the Prolog-based logical forms produced by many natural language parsers, allowing the user to make requests of the agent community in plain English. As a simple example, the English request "What is the telephone number of John Bear's manager?" would be converted to the ICL expression:

```
oaa_Solve( (manager( 'John Bear', M),  
           phone_number(M, P)),  
          [query(var(P))] ).
```

Parameters can specify both low-level constraints or high-level advice. Examples of low-level constraints might include the maximum amount of time for the solution, the maximum number of solutions returned for a query, how the information should be returned (e.g., as a blocking call or asynchronous streamed response), and rarely, which agents are allowed to participate in the computation.

As an example of high-level advice parameters, notice that the way parallelism should occur depends on the type of task being solved. If three email agents are available on the network, the request "Send this by email to Luc" should probably not be sent to all three agents at once, but rather if the first doesn't succeed, the others should be tried in succession. Compare this with a database query, where it might be very desirable to send the request in parallel to as many available agents as possible. When solving a math problem, different answers returned by different mathematicians could signal a problem, whereas if the participants are students composing poems, varied answers are a requirement.

Data Management

OAA's distributed data facilities share much in common with the distributed goal resolution process described in the previous section. In the same way that agents register the tasks they are capable of performing, agents also declare descriptions of the data they manage. An agent can then add, delete, change, or query a data value, and

this request will be automatically routed by the Facilitator agent to the appropriate agent or agents.

Data declarations and functions also make use of the notion of parameter lists. In this case, parameters specify information about permissions, scoping, persistence, whether duplicate values are allowed, and so forth. Data parameters are also used provide synchronous collaboration features to OAA applications; the 'shareable' attribute determines whether a data value is synchronized among all participants of a distributed collaborative session.

Triggers

Triggers allow an agent, or set of agents, to monitor some potentially complex state in the world, performing an action if the trigger's test conditions become true.

Triggers or rules exist in many commercial systems today; for instance, mail programs often allow the user to define actions (e.g., delete, archive, forward) to perform if an email of a certain type arrives. However, in these systems, the action must be predefined and fixed. With OAA triggers, the action part of a trigger may be any compound task executable by the dynamic community of agents. As new, perhaps unanticipated, agents connect to the system at runtime, what the user can say and do literally changes. For instance, if a new fax agent suddenly becomes available, the user can now say or write "If email arrives..., fax it to Bill", even if this action had never been conceived of by the original developers of the application.

Four types of triggers are currently defined by the OAA:

1. Data triggers: *"If the airline flight time changes..."*
2. Time triggers: *"In ten minutes..."*, *"every Thursday from now until Christmas..."*
3. Communication triggers: *"If any agent sends Msg..."*
4. Task triggers (specific to the domain of a particular agent): *"If mail arrives about..."*, *"If this Web page changes ..."*

Triggers are stored using the data management facilities, so they can be added, deleted, inspected, protected, and automatically distributed like any other database predicate.

MVIEWS Component Technologies

The MVIEWS application is implemented as a collection of OAA agents, as depicted in Figure 4; we'll now take a closer look at each of the component technologies.

Video Player

Two separate video players have been adapted for use with the MVIEWS system, each with slightly different properties. The first is a public domain UNIX-based program called XAnim, a software-based player capable of

displaying numerous video file formats, including MPEG and AVI. The second, MP, was written to take advantage of special libraries provided by Sun to access their hardware video boards. MP can play either from MPEG files or a video source (e.g., live camera or VCR). We are considering integrating yet a third player to provide a Windows PC solution.

The video players were adapted to work within the OAA framework by including the OAA agent library, and by publishing the video players' capabilities using the ICL formalism. In addition, the programs were extended to permit drawing on top of the video by using a mouse or electronic pen.

Speech Recognition

Speech recognition (SR) is used both for entering commands to the system and for extracting content from a user's verbal annotations for video indexing and report generation. The SR technology used by MVIEWS is a large vocabulary, continuous speech, speaker-independent system developed at SRI's STAR Laboratory and then commercialized by a spin-off company, Nuance Communications.⁶ The recognizer is based on a hidden Markov model approach, and takes as input a set of models either compiled from a regular expression grammar notation, or constructed through a learning process over a large corpus of data. In the current demonstration system, a grammar defines the set of possible spoken commands, as well as the set of keywords and phrases that can be recognized from annotations or verbal reports.

Natural Language

Two aspects of natural language (NL) processing are used within the MVIEWS system to handle the results from the speech recognition process: NL understanding, to interpret commands to the system, and information extraction from text, to produce indices, summarizations, and reports from spoken annotations.

The OAA is often used to integrate different levels of NL understanding, depending upon the requirements of the system. In most OAA-based systems, prototypes are initially constructed using relatively simple NL components, and as the vocabulary and grammar complexities grow, more powerful technologies can be incrementally added. The current MVIEWS demonstration system has a relatively limited set of commands that are processed by two of our low-end NL systems: Nuance's template-slot tools and DCG-NL, a Prolog-based top-down parser. As the MVIEWS prototype matures, more efficient NL systems can be added, such as

⁶ <http://www.nuancecom.com/>

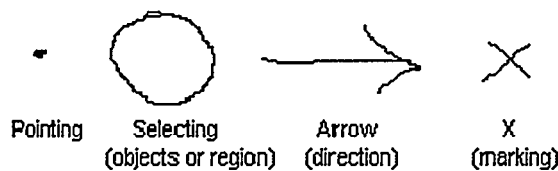


Figure 5. Gesture set.

Gemini, a robust bottom-up parser based on unification grammars, which interleaves syntax and semantics.

A current DARPA-funded project, which will be folded into the MVIEWWS system, focuses on the information extraction task in the Predator domain. Applying SRI's FASTUS [6] and adding better domain coverage for speech recognition will be an improvement over the current implementation, which is based on simple keyword spotting and a hand-coded grammar defining possible reported utterances.

Pen Recognition and Annotation

The pen modality is used in conjunction with speech to add multimodal annotations to a video document, and to issue commands to the system. For commands, a set of pen gestures (Figure 5) can be recognized using algorithms developed in [9].

In our experience, most pen annotations made by users also fall into the class of gestures, usually supplemented by a descriptive audio annotation. Since handwriting has rarely been used, incorporating handwriting recognition into the system has been a low-priority task. However, UNIX-based handwriting recognition libraries have been obtained from Communications Intelligence Corporation (CIC⁷), another SRI spin-off company, and may play an important role for labeling objects with out-of-vocabulary names, a task difficult for speech recognition systems.

Image Processing and Object Tracking

The problem with most image processing and object tracking algorithms is that they are often highly specialized, working well for certain situations and not at all in others. An image and video analyst needs to have an entire library of routines at his or her call.

In the current MVIEWWS prototype, we have integrated several image functions, such as stabilization and extraction of selected regions, as well as two object tracking algorithms.

The first of the two tracking algorithms is adapted for detecting fast-moving, relatively small objects within specified surveillance regions in the image. This process

requires specialized hardware, running on a Datacube Max Video 200 pipeline image processing system (MV200) with a Motorola 68040 host processor. The Lynx operating system on the MV200 is capable of reading and writing directly in the image memories, using a VME bus. The signal from the incoming video stream is digitized into 256 gray levels and then processed at close to 15 frames per second. Each processing step involves detection of motion between adjacent image frames, followed by temporal correspondence to correlate the moving segments in the video sequence. The strength of the approach is in the temporal association process, which is capable of handling occlusions of moving targets and false alarms from the motion algorithm. As moving targets are detected, their position and ID are passed through the OAA to all interested agents.

The second tracking algorithm, running locally on the Sun Ultrasparc workstation, is good at tracking slow-moving objects, given their initial position (e.g., "Track this car", or "If this boat moves, notify me"). This routine works by comparing via convolution a small subimage of the current video frame with a same-sized subimage from the previous frame. Tracking is initiated by selection of a seed subimage covering the object to be tracked. In our experiments, these subimages ranged in size from 11x11 to 27x27 pixels, depending on the size of the object to be tracked. The object model is formed by storing the location of the subimage within the frame along with the pixel values of the subimage and the square root of the sum of all pixel values within the subimage.

To find the location of the tracked object in a new frame, we find the local maximum of convolution scores by shifting the model subimage around the neighborhood of its previous location. When a local maximum is found, the pixel values of the subimage centered at that location in the new frame are taken as the new model. The model is updated every frame. The benefit is that the model can adapt to changing views of the tracked object. The drawback is that the algorithm can sometimes be fooled when a tracked object moves into a region where it cannot be distinguished from the background.

Multimodal Map

The multimodal map (MMAP) component of the MVIEWWS system allows a user to interact with a dynamic map display through a natural combination of speaking, writing, and drawing directly on the map surface. Multiple modalities may be entered simultaneously or in any sequential order, and merged to produce a command or request. This fusion makes use of the inherent parallelism of the OAA, with multiple agents competing and cooperating to resolve ambiguities arising during the interpretation process.

⁷ <http://www.cic.com/>

The multimodal map has been used in various OAA applications, such as providing a natural user interface to travel-related sources on the Web [1], and for guiding and monitoring multiple mobile robots [4]. Adding MMap's functionality as part of the MVIEWES application demonstrates OAA's extensibility and flexibility, in that no code had to be written to incorporate the technology into the MVIEWES domain.

Human Collaborative Tools

Within the OAA, a human user will typically interact with distributed software agents, and the agents themselves will communicate and cooperate with each other. A natural extension to this paradigm is to allow multiple human users to work with each other in a collaborative fashion.

The OAA has been extended to include services that facilitate adding synchronous collaborative functionality to any OAA-based user interface. The session management, state replication and an activity-based floor mechanism are provided in a peer-to-peer topology by the Synchronous Collaborative Object Oriented Toolkit (SCOOT) [3], developed by SRI's Augmented Collaboration Group, or alternatively by a purely OAA-based collaboration mechanism (centralized).

Lessons Learned

Although we have not yet run formal experiments to evaluate the utility of the MVIEWES system, our experiences do suggest several lessons.

The first is simply that there is a strong need in the Intelligence community for better tools to help an analyst interact more efficiently with video. Video's role in the analysis process is growing: in the case of the Predator UAV, its camera was originally intended strictly as sensor for the pilot to guide the plane, but ended up playing a role in battlefield assessment. In general, the MVIEWES concept has been well received at ETS and elsewhere, and many viewers have suggested domains to which it could be applied.

On the implementation side, two important questions were: how much and what kind of information should flow among distributed agents; and how should we deal with inaccuracies in technologies such as speech recognition or image processing.

Regarding information routing, we chose to limit interagent exchange to messages containing semantic representations of the data, not the data itself (instead of moving video across a network, we split the physical cables so each machine could have a local input source). Agents registered triggers, filters and constraints on broadcast data (as described in the section on OAA) and simple heuristics were inserted for regulating the rate with which information needed to be updated: a fast moving

object requires more frequent updates, than a slow one. Clearly, our simple prototype has not solved all hard problems in this area; however the facilitated approach seems promising for managing an efficient data flow.

Regarding recognition technologies, we found that speech and pen are currently reliable enough to be of use for controlling tasks in the user interface, but that for other tasks (speech: transcription of annotations; image processing: object tracking), recognition technologies produce imperfect results given a broad class of input. We chose to design MVIEWES such that as these technologies mature, they can take on an increased role. Multimedia annotation and playback are reliable and useful features by themselves, and provide data on which transcription and classification technologies can act in the future. An analyst can apply image functions such as stabilization and enhancement with confidence, and then get a sense of when this can be augmented by object tracking or detection. With image processing, a single algorithm will not be sufficient for all input, so we incorporated multiple algorithms, with their use under the direction of the human user. Eventually, we may be able to automate the decision about under which conditions each should be used.

RELATED WORK

Commercially available tools provide much useful functionality for video manipulation and annotation. One good example is Z/Videoware from Z Microsystems.⁸ Z/Videoware allows a user to play digital video clips and add audio annotations. It possesses an innovative feature that tries to classify the video by examining the closed-caption text embedded within, and then sorting the video appropriately into different folders.

Another example of a system for video annotation and analysis is VANNA [5]. This application provides a highly tailorable user interface, and an efficient system for annotating the video, using a variety of input devices, such as mouse, trackball, keyboard, touch screen and electronic pen.

Although commercial systems provide some of the functionality that one would want for video annotation, they are missing features that we feel are required for effective video exploitation, such as collaboration, natural user interfaces, and the ability to call up a variety of related data and tools from the same user interface. Although several research systems attempt to apply more advanced technologies to image processing [13], we are not aware of such systems focusing on video.

One effort that has a great deal in common with MVIEWES

⁸ Z Microsystems' homepage: <http://www.zmicro.com/>

is a DARPA-funded project called QuickTurn [7], by MITRE and Carnegie Mellon University. MVIEWWS and QuickTurn share many of same goals (an integrated environment combining advanced user interfaces with tools and databases for the intelligence analyst) and technologies (collaboration, mediated databases, maps, image tools). However, MVIEWWS attempts to focus its solutions within the context of the video analyst (e.g., object tracking, indexed search on pen and voice annotations).

CONCLUSIONS AND FUTURE DIRECTIONS

The current version of MVIEWWS can only be considered a working prototype system, but this IR&D project has already shown potential for enhancing the tools and techniques available in the domain of video exploitation and analysis. By using an open, distributed approach as the underlying architecture, we were able to rapidly bring together pertinent technologies, and facilitate the future development of the system as components are added, improved or replaced. Although MVIEWWS consists of a number of capabilities that are interesting by themselves, it is the *integrated use* of these capabilities that can greatly enhance the effectiveness of the operator.

Improvements and extensions will be made to the system. One comparatively large project is already under way to improve the speech recognition and data extraction from annotations in the Predator domain, by combining robust speech recognition and the FASTUS text extraction software. Other improvements to MVIEWWS will involve a wider use of the collaboration technology, adding more object tracking and image processing techniques, and identifying and integrating additional video-related technologies. We will also pursue the opportunity to perform user experiments to better quantify benefits of the system.

ACKNOWLEDGMENTS

This IR&D-funded project is the result of contributions from many talented people spanning six centers within SRI. The participants include: Jeff DeCurtins, Gopalan Ravichandran, Bikash Sabata (video and image processing), Greg Myers, Bob Bolles, Eric Rickard, Joel Cain (vision, design and direction), Luc Julia, Adam Cheyer (agent architecture, speech, gesture, handwriting).

REFERENCES

1. Cheyer, A. and Julia, L. Multimodal maps: An agent-based approach. In *Proc. of the International Conference on Cooperative Multimodal Communication (CMC/95)*, Eindhoven, May 1995.
2. Cohen, P., Cheyer, A., Wang, M., and Baeg, S. An open agent architecture. In *AAAI Spring Symposium*,

pages 1—8. Stanford University, March 1994.

3. Craighill, E., Fong, M., Skinner, K., Lang, R., and Gruenefeldt, K. SCOOT: An Object-Oriented Toolkit for Multimedia Collaboration. In *Proc. ACM Multimedia '94*, pages 41—49, San Francisco, October 1994.
4. Guzzoni, D., Cheyer, A., Julia, L., and Konolige, K. Many Robots Make Short Work. *AI Magazine*, Vol. 18, Number 1, pages 55—64. Spring 1997.
5. Harrison, B. L. and Baecker, R. M. Designing Video Annotation and Analysis Systems. In *Proc. of the Graphics Interface 92 Conference*, pages 157—166. Vancouver, B.C., May 11-15, 1992.
6. Hobbs, J., Appelt, D., Bear, J., Israel, D., Kameyama, M., Stickel, M., and Tyson, M. "FASTUS: a cascaded finite-state transducer for extracting information from natural-language text," in *Finite State Devices for Natural Language Processing* (E. Roche and Y. Schabes, eds.), Cambridge MA: MIT Press, 1996.
7. Holland, Roderick. QuickTurn: Advanced Interfaces for the Imagery Analyst. *DARPA/ITO Intelligent Collaboration & Visualization (IC&V) Program PI Meeting*. Dallas, Texas, October 10, 1996. <http://www.ito.darpa.mil/Proceedings/icv/agenda.html>
8. Julia, L., Cheyer, A., Neumeyer, L., Dowding, J., and Charafeddine, M. [HTTP://WWW.SPEECH.SRI.COM/DEMOS/ATIS](http://WWW.SPEECH.SRI.COM/DEMOS/ATIS). In *AAAI Spring Symposium*, pages 72—76. Stanford University, March 1997.
9. Julia, L., and Faure, C. Pattern recognition and beautification for a pen-based interface. In *ICDAR'95*, pages 58—63, Montreal, Canada, 1995.
10. Martin, D., Cheyer, A., and Lee, GL. Agent development tools for the open agent architecture. In *Proc. of the International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM)*. London, April 1996.
11. Martin, D., Oohama, H., Moran, D., and Cheyer, A. Information brokering in an agent architecture. In *PAAM'97*. London, April 1997.
12. Moran, D., Cheyer, A., Julia, L., and Park, S. Multimodal user interfaces in the Open Agent Architecture. In *Proc. of IUI-97*, pages 61—68. Orlando, Jan. 1997.
13. Srihara, R., Zhang, Z., and Chopra, R. Show & Tell: Using Speech Input for Image Interpretation and Annotation. In *AAAI-97 Spring Symposium, Workshop on Intelligent Integration and Use of Text, Image, Video and Audio Corpora*, pages 17—24. Stanford University, March 1997.

On Representing Salience and Reference in Multimodal Human-Computer Interaction

From: AAAI Technical Report WS-98-09. Compilation copyright © 1998, AAAI (www.aaai.org). All rights reserved.

Andrew Kehler¹, Jean-Claude Martin², Adam Cheyer¹, Luc Julia¹, Jerry R. Hobbs¹
and John Bear¹

¹ SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025 USA

² LIMSI-CNRS, BP 133, 91403 Orsay Cedex, France

Abstract

We discuss ongoing work investigating how humans interact with multimodal systems, focusing on how successful reference to objects and events is accomplished. We describe an implemented multimodal travel guide application being employed in a set of Wizard of Oz experiments from which data about user interactions is gathered. We offer a preliminary analysis of the data which suggests that, as is evident in Huls et al.'s (1995) more extensive study, the interpretation of referring expressions can be accounted for by a rather simple set of rules which do not make reference to the type of referring expression used. As this result is perhaps unexpected in light of past linguistic research on reference, we suspect that this is not a general result, but instead a product of the simplicity of the tasks around which these multimodal systems have been developed. Thus, more complex systems capable of evoking richer sets of human language and gestural communication need to be developed before conclusions can be drawn about unified representations for salience and reference in multimodal settings.

Introduction

Multimodal systems are particularly appropriate for applications in which users interact with a terrain model that is rich in topographical and other types of information, containing many levels of detail. Applications in this class span the spectrum from travel guide systems containing static, two-dimensional models of the terrain (e.g., a map-based system), to crisis management applications containing highly complex, dynamic, three-dimensional models (e.g., a forest fire fighting system). We are currently investigating how humans interact with multimodal systems in such settings, focusing on how reference to objects and events is accomplished as a user communicates by gesturing with a pen (by drawing arrows, lines, circles, and so forth), speaking natural language, and handwriting with a pen.

In this report, we begin to address the question of how knowledge and heuristics guiding reference resolution are to be represented. Is it possible to have a unified representation for salience that is applicable across multimodal systems, or do new tasks require

new representations? Can constraints imposed by the task be modularized in the theory, or are they inherently strewn within the basic mechanisms? Can linguistic theories of reference, which typically treat gestural and spoken deixis as a peripheral phenomenon, be naturally extended to the multimodal case, in which such deixis is the norm?

A Fully Automated Multimodal Map Application

The basis for our initial study is an implemented prototype multimodal travel guide application (Cheyer & Julia 1995) that was inspired by a multimodal Wizard of Oz simulation (Oviatt 1996). The system provides an interactive interface on which the user may draw, write, or speak. The system makes available information about hotels, restaurants, and tourist sites that have been retrieved by distributed software agents from commercial Internet World Wide Web sites.

The types of user interactions and multimodal issues handled can be illustrated by a brief scenario featuring working examples. Suppose Mary is planning a business trip to Toronto, but would like to schedule some activities for the weekend. She turns on her laptop PC, executes a map application, and selects Toronto.

To determine the most appropriate interpretation for the incoming streams of multimodal input, our approach employs an agent-based framework to coordinate competition and cooperation among distributed information sources, working in parallel to resolve the ambiguities arising at every level of the interpretation process. With respect to interpreting anaphora, such as in the command "Show photo of hotel", separate information sources may contribute to the resolution:

- Context by object type: The natural language component can return a list of hotels talked about.
- Deictic: Pointing, circling, or arrow gestures might indicate the referent, which may occur before, during, or after an accompanying verbal command.
- Visual context: The user interface agent might determine that only one hotel is currently visible.

M: [Speaking] Where is downtown?
Map scrolls to appropriate area.

M: [Speaking and drawing region]
 Show me all hotels near here.
Icons representing hotels appear.

M: [Writes on a hotel] Info?
A textual description appears.

M: [Speaking] I only want hotels with a pool.
Some hotels disappear.

M: [Draws a crossout on a hotel near a highway]
Hotel disappears.

M: [Speaking and circling]
 Show me a photo of this hotel.
Photo appears.

M: [Points to another hotel]
Photo appears.

M: [Speaking] Price of the other hotel?
Price appears for previous hotel.

M: [Speaking and drawing an arrow] Scroll down.
Display adjusted.

M: [Speaking and drawing an arrow toward a hotel]
 What is the distance from here to China Town?
A line and number representing distance displayed.

- Database queries: Information from a database agent can be combined with results from other resolution strategies, such as location information for the hotel asked about.
- Discourse analysis: The discourse history provides information for interpreting phrases such as “No, the other one.”

The map application is implemented within a multi-agent framework called the Open Agent Architecture (OAA).³ The OAA provides a general-purpose infrastructure for constructing systems composed of multiple software agents written in different programming languages and running on different platforms. Similar in spirit to distributed object frameworks such as OMG’s CORBA or Microsoft’s DCOM, agent interactions are more flexible and adaptable than the tightly bound object method calls provided by these architectures, and are able to exploit parallelism and dynamic execution of complex goals. Instead of preprogrammed single method calls to known object services, an agent can express its requests in terms of a high-level logical description of what it wants done, along with optional constraints specifying how the task should be performed. This specification request is processed by one or more Facilitator agents, which plan, execute and monitor the coordination of the subtasks required to accomplish the end goal (Cohen *et al.* 1994).

³Open Agent Architecture and OAA are trademarks of SRI International. Other brand names and product names herein are trademarks and registered trademarks of their respective holders.

Application functionality in the map application is thus separated from modality of user interaction. The system is composed of 10 or more distributed agents that handle database access, speech recognition (Nuance Communications Toolkit or IBM’s Voice-Type), handwriting (by CIC) and gesture (in-house algorithms) recognition, and natural language interpretation. These agents compete and cooperate to interpret the streams of input media being generated by the user. More detailed information regarding agent interactions for the multimodal map application and the strategies used for modality merging can be found in Cheyer and Julia (1995) and Julia and Cheyer (1997).

Data Collection

Despite the coverage of the system’s current anaphora resolution capabilities, we are interested in collecting naturally-occurring data which may include phenomena not handled by our system. We therefore designed a Wizard of Oz (WOZ) experiment around the travel guide application. In WOZ experiments, users believe they are interacting directly with an implemented system, but in actuality a human “wizard” intercepts the user’s commands and causes the system to produce the appropriate output. The subject interface and wizard interface are depicted in Figure 1.

Experiment Description Subjects were asked to plan activities during and after a hypothetical business trip to Toronto. They planned places to stay, sights to see, and places to dine using speech, writing, and pen-based gestures. The task consisted of four subtasks. To provide experience using each modality in isolation, during the first two tasks subjects planned half days using speech only and pen only respectively. In the third task, subject planned two half-days using any combination of these modalities they wished. Finally, the subjects completed a direction giving task, begun by picking up a phone placed nearby. On the other end was an experimenter who told the subject that he wants to meet for dinner, providing the name of the hotel at which he is staying and the restaurant at which they are to meet. The subject then interacted with the system to determine directions to give to the experimenter. For all tasks, the subjects were given only superficial instruction on the capabilities of the system. The tasks together took an average of 40 minutes. At the end of a session, the subjects were given surveys to determine whether they understood the task and the modalities available to them, and to probe their thoughts on the quality of the system.

The interactions were recorded using video, audio, and computer storage. The video displays a side-by-

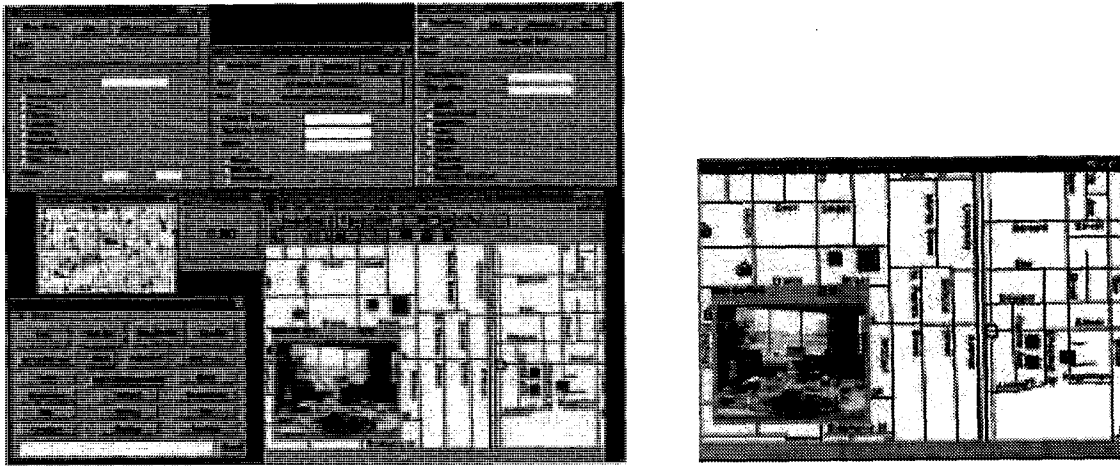


Figure 1: The Wizard Interface (left) and the Subject Interface (right)

side view with the subject on one side and the map interface on the other. The video and audio records are used for transcription, and the computer storage for reenacting scenarios for evaluation.

Coevolution of Multimodal and Wizard-of-Oz Systems In our quest for unconstrained, naturally-occurring data, we sought to place as few assumptions on the user interactions as possible. Unfortunately, WOZ experiments using simulated systems often necessitate such assumptions, so that facilities allowing the wizard to respond quickly and accurately can be encoded. We have improved upon this paradigm by having the wizard use our implemented and highly capable multimodal system to produce the answers to the user.

As described by Cheyer et al. (1998), our multimodal map application already possessed two qualities that allowed it to be used as part of a WOZ experiment. First, the system allows multiple users to share a common workspace in which the input and results of one user may be seen by all members of the session. This enables the Wizard to see the subject's requests and remotely control the display. Second, the user interface can be configured on a per-user basis to include more or fewer graphical user interface (GUI) controls. Thus, the Wizard can use all GUI command options, and also work on the map by using pen and voice. Conversely, the subject is presented with a map-only display. To extend the fully automated map application to be suitable for conducting WOZ simulations, we added only three features: a mode to disable the automatic interpretation of input from the subject, domain-independent logging and playback functions, and an agent-based mechanism for sending WOZ-specific in-

structions (e.g., Please be more specific.) to the user with text-to-speech and graphics.

The result is a hybrid WOZ experiment: While a naive user is free to write, draw, or speak to a map application without constraints imposed by specific recognition technologies, the hidden Wizard must respond as quickly and accurately as possible by using any available means. In certain situations, a scrollbar or dialog box might provide the fastest response, whereas in others, some combination of pen and voice may be the most efficient way of accomplishing the task. In a single experiment, we simultaneously collect data input from both an unconstrained new user (unknowingly) operating a simulated system – providing answers about how pen and voice are combined in the most natural way possible – and from an expert user (under duress) making full use of our best automated system, which clarifies how well the real system performs and lets us make comparisons between the roles of a standard GUI and a multimodal interface. We expect that this data will prove invaluable from an experimental standpoint, and since all interactions are logged electronically, both sets of data can be applied to evaluating and improving the automated processing.

Performing such experiments and evaluations in a framework in which a WOZ simulation and its corresponding fully functional end-user system are tightly intertwined produces a bootstrap effect: as the automated system is improved to better handle the corpus of subject interactions, the Wizard's task is made easier and more efficient for future WOZ experiments. The methodology promotes an incremental way of designing an application, testing the design through semi-automated user studies, gradually developing the automated processing to implement appropriate behavior

for input collected from subjects, and then testing the finished product while simultaneously designing and collecting data on future functionality – all within one unified implementation. The system can also be used without a Wizard, to log data about how real users make use of the finished product.

Data Analysis

At the time of this writing, 17 subjects out of a planned 25 have completed the tasks. We are currently in the process of transcribing and analyzing this data, and so we limit our discussion to a subset of 10 of the sessions. Our conclusions must therefore remain preliminary.

Our analysis of the data covers a broad range of factors concerning modality use. In addition to classical metrics used for analyzing multimodal corpora (monomodal features, temporal relationship between speech and gesture), we are analyzing the commands using a typology based on types of cooperation: specialization, equivalence, redundancy, complementarity, concurrency, and transfer (Martin 1997; Martin, Julia, & Cheyer 1998). Our focus here, however, concerns the use of referring expressions, and we therefore restrict our analysis to this issue.

Models of linguistic reference generally consist of two components. The first is the evolving representation of the discourse state, or “discourse model”, which usually includes a representation of the salience of previously introduced entities and events. For instance, entities introduced from an expression occupying subject position are generally considered as being more salient for future reference than those introduced from the direct object or other positions. The second component is a representation of the properties of referring expressions which dictates how they should be interpreted with respect to the discourse model (Prince 1981; Gundel, Hedberg, & Zacharski 1993). For instance, pronouns have been claimed to refer to entities that are highly salient or ‘in focus’, whereas full definite noun phrases need not refer to salient entities, or even ones that have been mentioned at all. Similarly, the choice among different deictic expressions (i.e., ‘this’ vs. ‘that’) is presumably guided by factors relating to the relative places at which their antecedents reside within the discourse model. Within this picture, the representation of discourse state and the interpretation of referring expressions against it are kept distinct; furthermore, they are considered independent of the task underlying the interaction.

An alternative embodied in some multimodal systems, including ours, could be termed the ‘decision list’ approach. Here, heuristics are encoded as a decision list (i.e., a list of if-then rules applied sequen-

tially) which do not necessarily enforce a strict separation between the representation of multimodally-integrated salience factors and the identities and properties of particular referring expressions. Furthermore, these rules might even query the nature of the task being performed or the type of command being issued, if task analyses would suggest that such differences be accounted for (Oviatt, DeAngeli, & Kuhn 1997).

A unified, modularized theory of reference which is applicable across multimodal applications is presumably preferable to a decision list approach. Huls et al. (1995) in fact take this position and propose such a mechanism. They describe data arising from sessions in which subjects interacted with a system using a keyboard to type natural language expressions and a mouse to simulate pointing gestures. To model discourse state, they utilize Alshawi’s (1987) framework, in which *context factors* (CFs) are assigned significance weights and a decay function according to which the weights decrease over time. Significance weights and decay functions are represented together via a list of the form $[w_1, \dots, w_n, 0]$, in which w_1 is an initial significance weight which is then decayed in accordance with the remainder of the list. The *salience value* (SV) of an entity *inst* is calculated as a simple sum of the significance weights $W(CF_i)$:

$$SV(inst) = \sum_{i=1}^n W(CF_i^{inst})$$

Four “linguistic CFs” and three “perceptual CFs” were encoded. Linguistic CFs include weights for being in a major constituent position ([3,2,1,0]), the subject position ([2,1,0], in addition to the major constituent weight), a nested position ([1,0]), and expressing a relation ([3,2,1,0]). Perceptual CFs include whether the object is visible ([1,...,1,0]), selected ([2,...,2,0]), and indicated by a simultaneous pointing gesture ([30,1,0]). The weights and decay functions were determined by trial and error.

To interpret a referring expression, the system chooses the most salient entity that meets all type constraints imposed by the command and by the expression itself (e.g., the referent of “the file” in “close the file” must be something that is a file and can be closed). This strategy was used regardless of the type of referring expression. Huls et al. tested their framework on 125 commands containing referring expressions, and compared it against two baselines: (i) taking the most recent compatible reference, and a pencil-and-paper simulation of a focus-based algorithm derived from Grosz and Sidner (1986). They found that all 125 referring expressions were correctly resolved with their approach, 124 were resolved correctly with the Grosz

and Sidner simulation, and 119 were resolved correctly with the simple recency-based strategy.

The fact that all of the methods do very well, including a rather naive recency-based strategy, indicates a lack of difficulty in the problem. Particularly noteworthy in light of linguistic theories of reference is that this success was achieved with resolution strategies that were not tied to choice of referring expression. That is, well-known differences between the conditions in which forms such as “it”, “this”, “that”, “here”, and “there” are used apparently played no role in interpretation.

We were thus inclined to take a look at the reference behavior shown in our corpus. Table 1 summarizes the distribution of referring expressions within information-seeking commands for our 10 subjects. (Commands to manipulate the environment, such as to scroll the screen or close a window, were not included.) On the vertical axis are the types of referential form used. The symbol ϕ denotes “empty” referring expressions corresponding to phonetically unrealized arguments to commands (e.g., the command “Information”, when information is requested for a selected hotel). Full NPs are noun phrases for which interpretation does not require reference to context (e.g., “The Royal Ontario Museum”), whereas definite NPs are reduced noun phrases that do (e.g., “the museum”).

On the horizontal axis are categories indicating the information status of referents. We first distinguish between cases in which an object was gestured to (e.g., by pointing or circling) at the time the command was issued, and cases in which there was no such gesture. “Unselected” refers to a (visible) object that is not selected. “Selected Immediate” includes objects that were selected and mentioned in the previous command, whereas “Selected Not Immediate” refers to objects that have remained selected despite intervening commands that have not made reference to it (e.g., due to intervening commands to show the calendar or scroll the screen). There was also one outlying case, in which the user said “Are there any Spanish restaurants here”, in which “here” referred to the area represented by the entire map.

These data show a divergence between the distribution of referring expressions and the heuristics one might use to resolve them. On one hand, there are distributional differences in even our admittedly limited amount of data that accord roughly with expectations. For instance, unselected entities, which are presumably not highly salient, were never referred to with pronominal forms without an accompanying gesture. Instead, nonpronominal noun phrases were used (20 full NPs and 2 definite NPs), and in all cases the content of the noun phrase constrained reference to one possible

antecedent (e.g., “the museum” when only one museum was visible). Also, the antecedents of empty referring expressions were almost always highly-focused (selected, immediate) objects when no accompanying gesture was used, and “it” always referred to a selected, immediate antecedent. Finally, in accordance with their generally deictic use, “this NPs” (e.g., “this museum”) and “this” were usually accompanied by a simultaneous gesture. “Here” was only used when accompanied by such a gesture, whereas “there” was used for all types of selected referents.

Certain other facets of the distribution are more contrary to expectation. For instance, in 36 cases a full NP was used to refer to a selected, immediate object which, as such, was a candidate for a reduced referential expression. In four of these cases, the user also gestured to the antecedent, resulting in an unusually high degree of redundancy. We suspect that such usage may result from a bias some users have regarding the ability of computer systems to interpret natural language.

Despite the distributional differences among the referential forms, a simple algorithm can be articulated which handles all of the data without making reference to the type of referential expression used nor its distributional properties. First, the algorithm narrows the search given any type constraints imposed by the *content* (vs. the *type*) of the referring expression, as when full and definite NPs are used. As indicated earlier, in these cases the constraints narrowed the search to the correct referent. The remaining cases are captured with two simple rules: if there was a simultaneous gesture to an object, then that object is the referent; otherwise the referent is the currently selected object.

While our preliminary findings accord with Huls et al., we have articulated our rules in decision list form rather than a salience ordering scheme. In fact, at least part of the Huls et al. analysis appears to be of the decision list variety, albeit cast in a salience ordering format. For instance, they found, as did we, that all referring expressions articulated with simultaneous gesturing to an object refer to that object. While they encode this preference with a very large weight (30), this value is chosen only to make certain that no other antecedent can surpass it.

To conclude, the question of whether a unified view of salience and reference for multimodal systems can be provided remains open. It appears that the nature of the tasks used in our experiments and by Huls et al. makes for a relatively easy resolution task. This could be due to two reasons: either reference is generally so constrained in multimodal interactions that the distinctions made by different referring expressions

Form	No Gesture			Simultaneous Gesture			Total
	Unselected	Selected Immediate	Selected Not Immediate	Unselected	Selected Immediate	Selected Not Immediate	
Full NP	20	32	5	10	4	0	71
Definite NP	2	1	1	0	0	0	4
“here”	0	0	0	5	3	0	8
“there”	0	7	3	0	3	1	14
“this” NP	0	0	0	2	10	0	12
“that” NP	0	1	0	0	0	0	1
“this”	0	4	0	8	5	0	17
“they”	0	1	0	0	0	0	1
“it”	0	6	0	0	2	0	8
ϕ	0	22	2	13	1	0	38
TOTAL	22	74	11	38	28	1	174

Table 1: Distribution of Referring Expressions

become unimportant for understanding, or the systems that have been developed have not been complex enough to evoke the full power of human language and gestural communication. We expect that in fact the latter is the case, and are currently designing systems in more complicated domains to test this hypothesis.

Conclusions and Future Work

We have described an implemented multimodal travel guide application being used in a WOZ setting to gather data on how successful reference is accomplished. We presented a preliminary analysis of data which suggests that, as is evident in Huls et al.’s (1995) more extensive study, the interpretation of referring expressions can be accounted for by a set of rules which do not make reference to the type of expression used. This is contrary to previous research on linguistic reference, in which the differences between such forms have been demonstrated to be crucial for understanding.

We suspect that this not a general result, but instead a product of the simplicity of the tasks around which these multimodal systems have been developed. We are currently planning the development of a crisis management scenario which would involve expert or trainee fire-fighters directing resources to objectives while using a multimodal computerized terrain model. This model will be three-dimensional and dynamic, in contrast to the two-dimensional, static map application. We expect that the complexity of the task will evoke much richer interactions, and thus may serve to clarify the use of reference in these settings.

Acknowledgements

This work was supported by National Science Foundation Grant IIS-9619126, “Multimodal Access to Spatial Data”, funded within the Speech, Text, Image, and MULTimedia Advanced Technology Effort (STIMULATE).

References

- Alshawi, H. 1987. *Memory and Context for Language Interpretation*. Cambridge University Press.
- Cheyen, A., and Julia, L. 1995. Multimodal maps: An agent-based approach. In *Proceedings of CMC95*. 103–113.
- Cheyen, A.; Julia, L.; and Martin, J.-C. 1998. A unified framework for constructing multimodal experiments and applications. In *Proceedings of CMC98*, 63–69.
- Cohen, P.; Cheyer, A.; Wang, M.; and Baeg, S. 1994. An open agent architecture. In *AAAI Spring Symposium*. 1–8.
- Grosz, B., and Sidner, C. 1986. Attention, intentions, and the structure of discourse. *Computational Linguistics* 12(3):175–204.
- Gundel, J. K.; Hedberg, N.; and Zacharski, R. 1993. Cognitive status and the form of referring expressions in discourse. *Language* 69(2):274–307.
- Huls, C.; Bos, E.; and Classen, W. 1995. Automatic referent resolution of deictic and anaphoric expressions. *Computational Linguistics* 21(1):59–79.
- Julia, L., and Cheyer, A. 1997. Speech: a privileged modality. In *Proceedings of EUROSPEECH’97*. 103–113.

Martin, J.-C. 1997. Towards intelligent cooperation between modalities. The example of a system enabling multimodal interaction with a map. In *Proceedings of the IJCAI-97 Workshop on Intelligent Multimodal Systems*. 63-69.

Martin, J.-C.; Julia, L.; and Cheyer, A. 1998. A theoretical framework for multimodal user studies. In *Proceedings of CMC98*, 104-110.

Oviatt, S. 1996. Multimodal interfaces for dynamic interactive maps. In *Proceedings of CHI96*. 95-105.

Oviatt, S.; DeAngeli, A.; and Kuhn, K. 1997. Integration and synchronization of input modes during multimodal human-computer interaction. In *Proceedings of CHI97*. 415-422.

Prince, E. 1981. Toward a taxonomy of given-new information. In Cole, P., ed., *Radical Pragmatics*. New York, New York: Academic Press. 223-255.

An Open Agent Architecture*

Philip R. Cohen
Adam Cheyer
SRI International
(pcohen@ai.sri.com)
Michelle Wang
Stanford University
Soon Cheol Baeg
ETRI

ABSTRACT

The goal of this ongoing project is to develop an open agent architecture and accompanying user interface for networked desktop and handheld machines. The system we are building should support distributed execution of a user's requests, interoperability of multiple application subsystems, addition of new agents, and incorporation of existing applications. It should also be transparent; users should not need to know where their requests are being executed, nor how. Finally, in order to facilitate the user's delegating tasks to agents, the architecture will be served by a multimodal interface, including pen, voice, and direct manipulation. Design considerations taken to support this functionality will be discussed below.

INTRODUCTION

Agents are all the rage. "Visioneering" videos, such as Apple Computer's Knowledge Navigator, have helped to popularize the notion that programs endowed with agency, if not intelligence, are just around the corner. Soon, users need not themselves wade into the vast swamp of data in search of information, but rather the desired, or better yet, needed information will be presented to the user by an intelligent agent in the most comprehensible form, at just the right time.

Although rosy scenarios are easy to come by, intelligent agents are considerably more difficult to obtain. Still, substantial progress is being made on a variety of aspects of the agent story. At least three general conceptions of agent-based software systems can be found in current thinking:

1. Agents are programs sent out over the network to be executed on a remote machine.
2. Agents are programs on a given machine that offer services to others.

*This paper was supported by a contract from the Electronics and Telecommunications Research Institute (Korea). Our thanks are also extended to AT&T for use of their text-to-speech system.

3. Agents are programs that assist the user in performing a task.

Each of these models can be found to some extent in present-day software products, for example, in (1) General Magic's emerging TELESRIPT interpreter, (2) Microsoft's OLE 2.0 and (3) Apple Computer's Newton and Hewlett Packard's New Wave desktop, respectively. Given this space of conceptualizations, we need to be specific about ours.

Definitions and Objectives

Listed below are characteristics of what we are terming agents followed by an example of those characteristics as found in our system:

- *Delegation* — e.g., the ability to receive a task to be performed without the user's having to state all the details
- *Data-directed Execution* — e.g., the ability to monitor local or remote events, such as database updates, OS, or network activities, determining for itself the appropriate time to execute.
- *Communication* — e.g., the ability to enlist other agents (including people) in order to accomplish a task.
- *Reasoning* — e.g., the ability to prove whether its invocation condition is true, and to determine what are its arguments.
- *Planning* — e.g., the ability to determine which agent capabilities can be combined in order to achieve a goal.

Our initial prototype includes agents that exhibit aspects of all the above capabilities, except planning (but see [7]). Our goal is to develop an open agent architecture for networked desktop and handheld machines. The system we are building should support distributed execution of a user's requests, interoperability of multiple application subsystems, addition of new agents, and incorporation of existing applications. Finally, it should be transparent; users should not need to know where their requests are being executed, nor how.

AGENT ARCHITECTURE

Based loosely on Schwartz's FLiPSiDE system [17], the Open Agent Architecture is a blackboard-based framework allowing individual software "client" agents to communicate by means of goals posted on a blackboard controlled by a "Server" process.

The Server is responsible both for storing data that is global to the agents, for identifying agents that can achieve various goals, and for scheduling and maintaining the flow of communication during distributed computation. All communication between client agents must pass through the blackboard. An extension of Prolog has been chosen as the interagent communication language (ICL) to take advantage of unification and backtracking when posting queries. The primary job of the Server is to decompose ICL expressions and route them to agents who have indicated a capability in resolving them. Thus, agents can communicate in an undirected fashion, with the blackboard acting as a broker. Communication can also take place also in a directed mode if the originating agent specifies the identity of a target agent.

An agent consists of a Prolog meta-layer above a knowledge layer written in Prolog, C or Lisp. The knowledge layer, in turn, may lie on top of existing standalone applications (e.g. mailers, calendar programs, databases). The knowledge layer can access the functionality of the underlying application through the manipulation of files (e.g., mail spool, calendar datafiles), through calls to an application's API interface (e.g. MAPI in Microsoft Windows), through a scripting language, or through interpretation of an operating system's message events (Apple Events or Microsoft Windows Messages).

Individual agents can respond to requests for information, perform actions for the user or for another agent, and can install triggers to monitor whether a condition is satisfied. Triggers may make reference to blackboard messages (e.g. when a remote computation is completed), blackboard data, or agent-specific test conditions (e.g. "when mail arrives...").

The creation of new agents is facilitated by a client library furnishing common functionality to all agents. This library provides methods for defining an agent's capabilities (used by the blackboard to determine when this agent should participate in the solving of a subgoal), natural language vocabulary (used by the interface agent), and polling status. It also provides functionality allowing an agent to read and write information to the blackboard, to receive requests for information or action, and to post such requests to the blackboard, a specific agent, or an entire population of appropriate agents.

When attempting to solve a goal, an agent may find itself lacking certain necessary information. The agent can either post a request of a specific agent for the information, or it may post a general request on the black-

board. In the latter case, all agents who can contribute to the search will send solutions to the blackboard for routing to the originator of the request. The agent initiating the search may choose either to wait until all answers return before continuing processing, or may set a trigger indicating that when the remote computation is finished, a notification should interrupt local work in progress. An agent also has access to primitives permitting distributed AND and OR-parallel solving of a list of goals.

Distributed Blackboard Architecture

As discussed above, the Open Agent Architecture contains one blackboard "server" process, and many client agents; client agents are permitted to execute on different host machines. We are investigating an architecture in which a server may itself be a client in a hierarchy of servers; if none of its client agents can solve a particular goal, this goal may be passed further along in the hierarchy. Following Gelerntner's LINDA model [8], blackboard systems themselves can be structured in a hierarchy, which could be distributed over a network (see Figure 1).¹

When a goal (G) is requested to be posted on a local blackboard (BB1), and the blackboard server agent at BB1 determines that none of its child agents has the requisite capabilities to achieve the goal, it propagates the goal to a more senior blackboard server agent (BB4) in the hierarchy. BB4 maintains a knowledge base of the predicates that its lower level blackboards can evaluate. When a senior server receives such a request, it in turn will propagate the request down to its subsidiary servers. These subsidiary servers either have immediate client agents who can evaluate the goal, or can themselves pass on the goal to another subsidiary server. In the case illustrated in Figure 1, BB4 determines that none of its subsidiary blackboards can handle the goal, and thus sends the goal to its superior agent (BB5). BB5 passes the goal to BB6, who in turn passes it to BB9. When such a referred goal is passed through the hierarchy of blackboards, it is accompanied by information about the originating blackboard (indicated by the BB1 subscript on G), including information identifying its input port, host machine, etc. This continuation information will enable a return communication (with answers or failure) to be routed to the originating blackboard. Also, the identity of the responding knowledge source BB9 can be sent back to the originator, so that future queries of the same type from BB1 may be addressed directly to BB9 without passing through the hierarchy of blackboards.

Operational Agents

A variety of agents have been integrated into the Open Agent Architecture:

¹This is referred to as a "federation architecture" in [9].

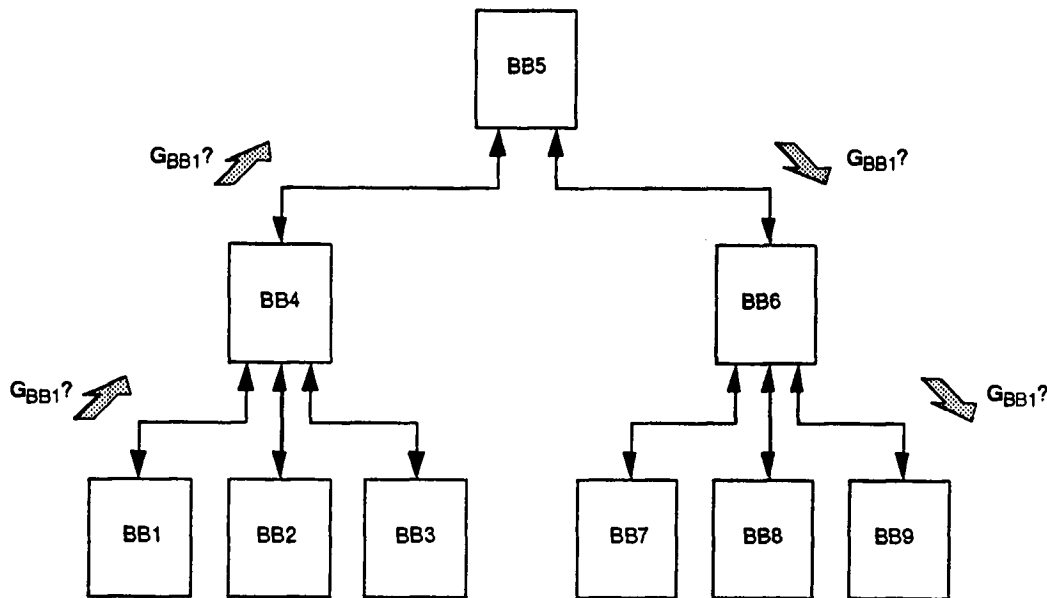


Figure 1: Hierarchy of Blackboard Servers

- a *User-interface* agent that accepts spoken or typed (and soon, handwritten) natural language queries from the user and presents responses to the queries.
- a *Database* agent, written in C, that interacts with a remote X.500 Directory System Agent database containing directory information.
- a *Calendar* agent, which can report upon where a person might be, or when they might be performing a particular action. This information is retrieved from data created by Sun Microsystem's CalenTool application.
- a *Mail* agent that can monitor incoming electronic messages, and forward or file them appropriately. The mail agent works with any Unix-compatible mail application (e.g. Sun's MailTool).
- a *News* agent that scans Internet newsgroups searching for specified topics or articles.
- a *Telephone* agent, that can dial a telephone using a ComputerPhone controller, and can communicate with users in English, using NewTTS, AT&T's text-to-speech system.

Communication Language

The key to a functioning agent architecture is the interagent communication language. We explain ours in terms of its form and content. Regarding the former,

three speech act types are currently supported: *Solve* (i.e., a question), *Do* (a request) and *Post* (an assertion to the blackboard). For the time being, we have adopted little of the sophisticated semantics known to underlie such speech acts [5, 18, 19]. However, in attempting to protect an agent's internal state from being overwritten by uninvited information, we do not allow one agent to change another's internal state directly — only an agent that chooses to accept a speech act can do so. For example, a fact posted to the blackboard does not necessarily get placed in the database agent's files unless it so chooses, by placing a trigger on the blackboard asking to be notified of certain changes in certain predicates (analogous to Apple Computer's Publish and Subscribe protocol).

Although our interagent communication language is still evolving, we have adopted Horn clauses as the basic predicates that serve as arguments to the speech act types. However, for reasons discussed below, we have augmented the language beyond ordinary Prolog to include temporal information.

Because delegated tasks and rules will be executed at distant times and places, users may not be able simply to use direct manipulation techniques to select the items of interest, as those items may not yet exist, or their identities may be unknown. Rather, users will need to be able to *describe* arguments and invocation

conditions, preferably in a natural language. Because these expressions will characterize events and their relationships, we expect natural language tense and aspect to be heavily employed [6]. Consequently, the meaning representation (or "logical form") produced by the multimodal interface will need to incorporate temporal information, which we do by extending a Horn clause representation with time-indexed predicates and temporal constraints. The blackboard server will need to decompose these expressions, distribute pieces to the various relevant agents, and engage in temporal reasoning to determine if the appropriate constraints are satisfied.

With regard to the content of the language, we need to specify the language of predicates that will be shared among the agents. For example, if one agent needs to know the location of the user, it will post an expression, such as `solve(location(user, U))`, that another agent knows how to evaluate. Here, agreement among agents would be needed that the predicate name is `location`, and its arguments are a person and a location. The language of nonlogical predicates need not be fixed in advance, it need only be common. Achieving such commonality across developers and applications is among the goals of the ARPA "Knowledge Sharing Initiative," [13] and a similar effort is underway by the "Object Management Group" (OMG) CORBA initiative to determine a common set of objects.

A difficult question is how the user interface can know about the English vocabulary of the various agents. When agents enter the system, they not only register their functional capabilities with the blackboard, they also post their natural language vocabulary to the blackboard, where it can be read by the user interface. Although conceptually reasonable for local servers (and somewhat problematic for remote servers) the merging of vocabulary and knowledge is a difficult problem. In the last section, we comment on how we anticipate building agents to enforce communication and knowledge representation standards.

Example Scenario

The following is an example of an operational demonstration scenario that illustrates inter-agent communication (see Figure 2).

The user tells the interface agent (in spoken language) that "When mail arrives for me about a security break, get it to me". The interface agent translates this statement into a logical expression, and posts the expression to the blackboard. The blackboard server determines that a trigger should be installed on the mail agent, causing it to poll the user's mail database. Once the mail agent has determined that a message matching the requested topic has arrived for the user, it posts a query to find out the user's current location. The calendar agent responds, noting that the user is supposed to be in a meeting which is being held in a particular room;

the database agent is then queried for the phone number of the room. Finally, the telephone agent is instructed to call the number, ask for the user (using voice synthesis), perform an identification verification by requesting a touchtone password, and then read the message to the user. We intend to add agents that would increase the number of ways in which a user might be contacted: agents to control fax machines, automatic pagers, and a notify agent that uses planning to determine which communication method is most appropriate in a given situation.

Comparison with Other Agent Architectures

The most similar agent architectures are FLiPSiDE [17] and that of Genesereth and Singh [9]. Like FLiPSiDE (Framework for Logic Programming Systems with Distributed Execution), our Open Agent Architecture uses Prolog as the interagent communication language, and introduces a uniform meta-layer between the blackboard server and the individual agents. Some aspects of FLiPSiDE's blackboard architecture are more complex than in our system. It uses a multi-level locking scheme to try to reduce deadlock and minimize conflicts in blackboard access during moments of high concurrency. The system also uses separate knowledge sources for controlling triggers, ranking priorities and scheduling the executing of knowledge sources, whereas we incorporate these sorts of actions directly into the blackboard server. Some features important to our system that are not addressed by FLiPSiDE are the ability to handle temporal constraints over variables, and the possibility for an agent to explicitly request AND and OR-parallel solving of a list of distributed goals.

Genesereth and Singh's architecture is more ambitious than ours in its employing a full first-order logic as the interagent communication language. As yet, we have not needed to expand our language beyond Horn clauses with temporal constraints, but this step may well be necessary. Genesereth and Singh use KIF (Knowledge Interchange Format) [13] as their basic language of predicates and as a knowledge integration strategy. Because of our user interface considerations, which in turn are heavily influenced by the form-factor constraints of future handheld devices, we will need to be able to merge contributions by different agents of their natural language vocabulary, related pronunciations, and semantic mappings of those vocabulary items to underlying predicates.

MAIL MANAGEMENT

In our earlier scenario, the mail agent was rather limited. To test our user interface and agent architecture more fully, we are creating a more substantive mail management agent, MAILTALK.

It has become common to develop mail managers that manipulate messages as they arrive according to a set

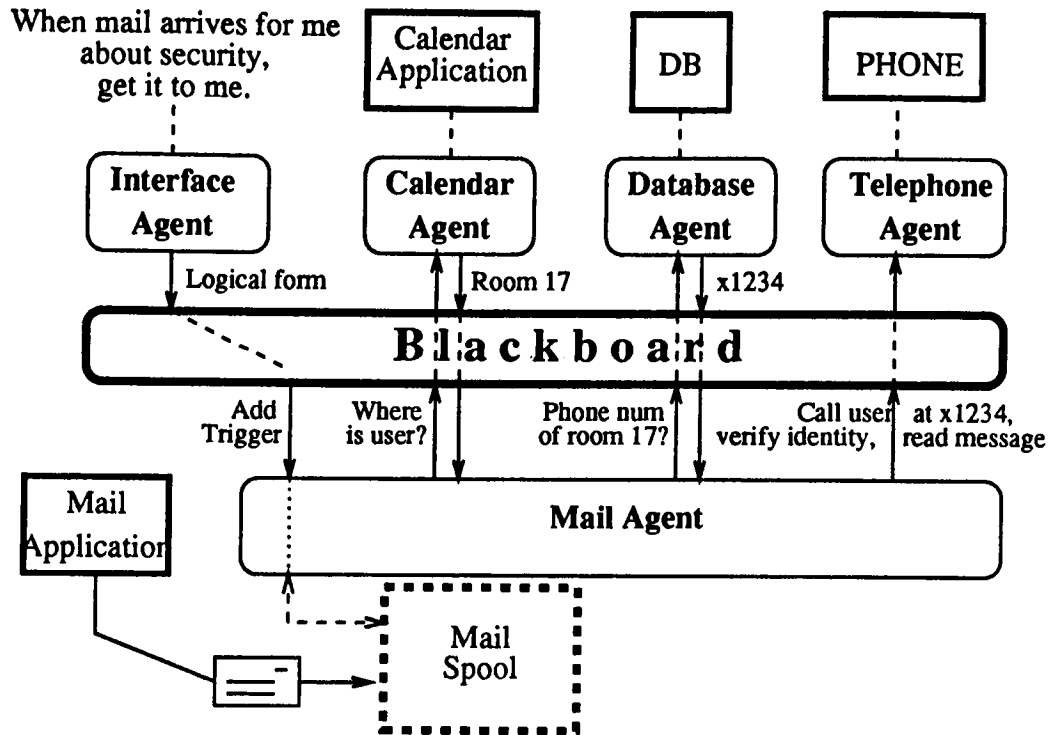


Figure 2: Example of agent interaction

of user-specified rules. The virtue of such systems is that users can make mail management decisions once, rather than consider each message in turn. However, a number of problems exist for such systems, as well as for all agent systems that we know of, especially when considered as tools for the general population.

- End users cannot easily specify the rules. In a number of current systems, a scripting language needs to be employed [1, 20], and in one system, users were required to write rules in a temporal query language [10]. We believe such methods for rule creation effectively eliminates the class of nontechnical users. Other systems employ templates that the user fills out [12]. Although this technique may work in many cases, it limits the power of the rules that users can create because they must search for an icon at which to point in order to specify the contents of a slot. Otherwise, they need to know or select the special syntax or concept name required. However, the selection of items from long menus is infeasible for handheld devices with little screen territory.
- End users cannot determine in advance how the *collection* of rules will behave once a new rule is added. This lack of predictability and the lack of debugging tools will undermine the utility of agent-based systems, especially in a networked environment.

- End users cannot easily determine what happened. Generally, little or no history of the database of events and rule firings is kept, and few tools are provided for reviewing that history.²
- The mail manager is a special purpose system, interacting loosely, if at all, with other components. Without tighter integration, the architecture and user interface for dealing with mail rules may diverge from what is offered for other agents.

Our prototype MAILTALK was built to address these concerns.

Rule specification. Based on technology developed for the SHOPTALK factory simulation system [2, 3, 4], MAILTALK permits users to specify rules by describing complex invocation conditions, and arguments with a multimodal interface featuring typed and spoken natural language, combined with direct manipulation. For example, the user can delegate to the mail agent as follows: "When Jones replies to my message about 'acl tutorials', send his reply to the members of my group." Here, Jones's reply cannot be selected or pointed at since it does not yet exist. The English parser produces expressions in the temporal logic, which are evaluated against various

²An exception to this is the use of "Mission Status Reports" in the Envoy agent framework [15].

When:	A message from someone in the AIC has been read
ARCHIVE	
Which message(s):	it
In which file:	<AIC-Mail>

Figure 3: Creating a mail rule

databases (e.g., the mail database, or a simulation database).

Predicting behavior. By giving end users the power to write their own rules means we have given them the freedom to make their own mistakes. Before letting a potentially erroneous collection of agents loose on one's mail (or, more generally, the network), we encourage users to *simulate* the behavior of those agents. Included with MAILTALK is a knowledge-based simulation environment that allows users to create hypothetical worlds, and permits them to send test messages or re-examine old mail files. In response, the system fires the relevant rules, and updates a simulation database with the events that have happened. This database can extend the actual mail file, permitting expressions that depend on the entire database to be evaluated (e.g., "when more than 5 messages from cohen are in < point to icon for mail file>, move them to <icon for 'unimportant mail'>).

Reviewing History. In order to determine if the resulting behavior was in fact desired, users can ask questions about the results of the simulation, can view the simulation graphically, and can rewind the history to interesting times (e.g., when a message was read, or when a message was forwarded to a member of a given mail group). When satisfied with the resulting behavior of the collection of rules, users can install them in the real world to monitor the real mail file. Moreover, users can ask questions about the real mail database, such as "Who has replied to my message of November 26 about budgets?"

Example

The following is an example of the kind of processing found in MAILTALK. First, the user determines that she wants to test out a mail management rule before installing it. She creates a new "hypothetical world," and proceeds to create a rule by selecting the **Archive** action from a menu. This results in a template's being presented, which she fills out as shown in Figure 3.

The user enters an English expression as the invocation condition, points at the icon for a file (**AIC-Mail**), and deposits it into the destination field.³ This rule

³For a discussion of the usability advantages of such templates over simply entering the above in one sentence, please see [3, 14].

definition is parsed into a Prolog representation, augmented with temporal information and constraints.

The user then proceeds to *digest* an old mail file, which simulates the sending of the old messages, updating the simulated mail database. The animated simulation indicates that the rule has been fired, but just to be certain that the appropriate messages were put into the desired file, the user asks "When did I read a message from someone in the AIC?", followed by "Where are those messages now?" When satisfied, she transfers this rule to the real world, and requests that incoming mail be monitored.

It should be noted that the reading of a message creates an event that triggers a rule. In general, that verb (i.e., 'read') could be one that results from an agent's action (e.g., forwarding), and thus a cascade of rule activations would ensue. It is to ensure that users understand such complexities that we offer the simulation facility.

Comparison with Other Mail Managers

Numerous mail managers exist, and space precludes a comprehensive survey. Only the more comparable ones will be discussed below.

The mail management system most similar to our is ISCREEN [16]. It allows a keyword and forms-based creation of rules, and offers a simple simulation capability in which a user can pose test messages. In response, the system applies its rules and explains in English what it would have done. Because mail is filtered using a boolean combination of keywords in various fields, ISCREEN can detect that various rules will conflict, and can ask the user for a prioritization. The user can employ organizational expressions (e.g., "manager"), which the system resolves based on a Prolog-based Corporate Directory database. Our use of the X.500 Directory System Agent offers the same capability based on an emerging international standard.

The TAPESTRY mail system [10] incorporates a mail database (as opposed to just a mail file), that is queried by a temporal query language. MAILTALK share this basic underlying model, but rather than have users write temporal queries, the user interface creates the temporal logic expressions through English language descriptions, which are then evaluated over the mail database.

The INFORMATION LENS system [12] provides various message types, which can enter into filtering rules (e.g., when a message of type **Weekly Sales Report** arrives, forward it to ...), or can become arguments for other actions (e.g., opening a spreadsheet). This approach takes the first step to integrating mail with other agent-like behavior, but a more fuller integration is possible once it is realized that rule-based mail management is analogous to database monitoring (as shown in TAPESTRY), and that a more general agent architecture can subsume mail management as a special case. It is this latter approach that we are following by embed-

ding the mail manager as an agent in the architecture.

IMPLEMENTATION

An initial implementation of each of the pieces described above has been developed (in Prolog and C) on a Unix platform, with the exception of the pen/voice interface, which is being implemented now. Communication is based on TCP/IP. The blackboard architecture has been ported to Windows/NT, and agents that encapsulate Microsoft API's will be developed. Also planned is a port of the blackboard interpreter to the Macintosh. When completed, the architecture will support multiple hardware and software platforms in a distributed environment.

FUTURE PLANS

In addition to the integration activities discussed above, a number of future research activities are needed. In order that an agent be invocable, its capabilities need to be mapped into terms understood by the ensemble of agents, and also by users. Moreover, as discussed earlier, the natural language vocabulary needed to invoke an agent's services, including lexical, syntactic, and semantic properties, will also be posted on the blackboard for use by the user interface. In general, however, this advertising of vocabulary can lead to conflicts among definitions. We intend to develop an API Description Tool, with which the agent designer describes the services provided by that agent. The tool will produce mappings of expressions in ICL into those services, including vocabulary and knowledge representations that can be merged into a common whole. Techniques used in developing natural language database porting tools (e.g., TEAM [11]) will be investigated.

In order to generalize the simulation approach in MAILTALK to encompass the entire collection of agents, the API Description Tool also needs to supply information sufficient to allow the agent architecture to simulate an agent's behavior. It will need to characterize the preconditions and effects of agent actions, thereby also providing a basis for a server's planning to incorporate the agent into a complex action that satisfies a user's stated goal [7].

Finally, an interesting question is where to situate the temporal reasoning subsystem. Currently, it is located with the blackboard server, but it could also be distributed as part of the agent layer, enabling other agents to accept complex expressions for evaluation and/or routing. We intend to experiment with various architectures.

References

- [1] S.-K. Chang and L. Leung. A knowledge-based message management system. *ACM Transactions on Office Information Systems*, 5(3):213-236, 1987.
- [2] P. R. Cohen. Integrated interfaces for decision support with simulation. In B. Nelson, W. D. Kelton, and G. M. Clark, editors, *Proceedings of the Winter Simulation Conference*, pages 1066-1072. Association for Computing Machinery, December 1991. invited paper.
- [3] P. R. Cohen. The role of natural language in a multimodal interface. In *The 2nd FRIEND21 International Symposium on Next Generation Human Interface Technologies*, Tokyo, Japan, November 1991. Institute for Personalized Information Environment. Also appears in *Proceedings of UIST'92*, ACM Press, New York, 1992, 143-149.
- [4] P. R. Cohen, M. Dalrymple, D. B. Moran, F. C. N. Pereira, J. W. Sullivan, R. A. Gargan, J. L. Schlossberg, and S. W. Tyler. Synergistic use of direct manipulation and natural language. In *Human Factors in Computing Systems: CHI'89 Conference Proceedings*, pages 227-234, New York, New York, April 1989. ACM, Addison Wesley Publishing Co.
- [5] P. R. Cohen and H. J. Levesque. Rational interaction as the basis for communication. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*. MIT Press, Cambridge, Massachusetts, 1990.
- [6] M. Dalrymple. The interpretation of tense and aspect in English. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, Buffalo, New York, June 1988.
- [7] O. Etzioni, N. Lesh, and R. Segal. Building softbots for UNIX. Department of Computer Science and Engineering, University of Washington, unpublished ms., November 1992.
- [8] D. Gelernter. *Mirror Worlds*. Oxford University Press, New York, 1993.
- [9] M. Genesereth and N. P. Singh. A knowledge sharing approach to software interoperation. Computer Science Department, Stanford University, unpublished ms., January 1994.
- [10] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61-70, December 1992.
- [11] B. J. Grosz, D. Appelt, P. Martin, and F. Pereira. Team: An experiment in the design of transportable natural language interfaces. *Artificial Intelligence*, 32(2):173-244, 1987.
- [12] T. W. Malone, K. R. Grant, F. A. Turbak, S. A. Brobst, and M. D. Cohen. Intelligent information-sharing. *Communications of the ACM*, 30(5):390-402, May 1987.

- [13] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3), 1991.
- [14] S. L. Oviatt, P. R. Cohen, and M. Wang. Reducing linguistic variability in speech and handwriting through selection of presentation format. In K. Shirai, editor, *Proceedings of the International Conference on Spoken Dialogue: New Directions in Human-Machine Communication*, Tokyo, Japan, November 1993.
- [15] M. Palaniappan, N. Yankelovitch, G. Fitzmaurice, A. Loomis, B. Haan, J. Coombs, and N. Meyrowitz. The Envoy framework: An open architecture for agents. *ACM Transactions on Information Systems*, 10(3):233-264, July 1992.
- [16] S. Pollock. A rule-based message filtering system. *ACM Transactions on Office Information Systems*, 6(3):232-254, July 1988.
- [17] D. G. Schwartz. Cooperating heterogeneous systems: A blackboard-based meta approach. Technical Report 93-112, Center for Automation and Intelligent Systems Research, Case Western Reserve University, Cleveland, Ohio, April 1993. Unpublished Ph.D. thesis.
- [18] J. R. Searle. *Speech acts: An essay in the philosophy of language*. Cambridge University Press, Cambridge, 1969.
- [19] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51-92, 1993.
- [20] R. Turlock. SIFT: A Simple Information Filtering Tool. Bellcore, Mountain, New Jersey, 1993.

PAAM 97

422

Q 335
I 565X
1997

Proceedings of the Second International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology

Conference Organisation

The Practical Application Company

Conference and Programme Chair

Barry Crabtree

Sponsorship Co-ordinator

Clive Spenser

21st-23rd April 1997

Westminster Central Hall, London, UK

No part of this book may be reprinted or reproduced without the written consent of the organiser and publisher.

PAAM, P.O. Box 137, Blackpool, Lancashire, FY2 9UN, UK

Tel: +44 (0)1253 358081, Fax: +44 (0)1253 353811

Email: proceedings@pap.com

ISBN 0 9525554 6 8

Published by The Practical Application Company Ltd

Information Brokering in an Agent Architecture

David Martin

Artificial Intelligence Center
SRI International
martin@ai.sri.com

Hiroki Oohama

Laboratory of Information Technology
NTT Data
hama@lit.rd.nttdata.co.jp

Douglas Moran

Artificial Intelligence Center
SRI International
moran@ai.sri.com

Adam Cheyer

Artificial Intelligence Center
SRI International
cheyer@ai.sri.com

Abstract

To date, document identification based on keyword matching strategies has been the basis of most efforts to provide assistance in accessing information resources on the Internet. However, in view of the limitations on human browsing time and the evolution of more capable software agents, we can expect rapid expansion in the use of fully queryable information sources (such as databases and knowledge bases) and *semiqueryable* sources (such as form-based query pages on the World Wide Web, and collections of Web pages that are structured by textual markups).

The ability to obtain information from a wide variety of queryable and semi-queryable sources is a prerequisite for the success of many types of software agents. Providing access to these sources — whether for end users or for software agents acting on behalf of users — poses a number of interesting challenges, many of which can themselves be addressed using agent-based approaches.

This paper describes a working prototype Information Broker system, developed within the Open Agent Architecture framework, that provides transparent access to a variety of information sources, each encapsulated as an independent agent. In this system, a broker *meta-agent* provides flexible mediation services, accepting queries expressed in broker or source schemas, gathering and integrating all available responses from the relevant sources, and allowing for the addition or deletion, at runtime, of participating information sources. Other broker features support the use of conversions, normalizations, and other basic domain knowledge in queries, and *persistent queries* (for which the Broker notifies requestors of changes in information sources that could affect their results). Source agents implement caching and retrieval strategies that alleviate problems related to the long access times and unreliability of Internet sources.

1 Introduction

The rapid growth of the World Wide Web and other forms of Internet and intranet access, and the need for automated assistance in making use of these resources, are now widely appreciated. The most immediate need, and the one receiving the greatest attention to date, is in the area of document retrieval; that is, the identification of textual documents that are relevant to some topic of interest. The topic of interest is generally indicated by an expression containing keywords, and the goal of the retrieval is to locate documents from which a human reader can extract useful information. A number of systems — such as NTT Data's *InterInfo*¹ in the commercial realm and *Amalthaea* [11] in the realm of agent research — are providing increasingly sophisticated approaches to document retrieval.

1.1 Structured and Semistructured Information Sources

An equally important set of challenges, but one that has not yet become as widely recognized, exists in the area of structured and semistructured information sources. By *structured* information sources, we mean sources that can be queried by using well-defined, general query languages, such as relational databases, object-oriented databases, and knowledge bases.

By *semistructured* information sources, we include a variety of sources that contain sufficient structure to be treated as databases, even though they do not provide the full generality and power of a query language. In the context of the World Wide Web, this structure is usually provided in one (or both) of two ways: by an informal form-based query interface or by the presence of HTML (HyperText Markup Language) markups and other textual markers used according to site-specific conventions.

We refer to a semistructured source that provides an informal form-based query interface as a *semiqueryable* source. It should be apparent that, even when such an interface is provided, its “query” capability usually falls far short of the power and generality of a structured information source. When a source provides structure by textual markers, we call that a *semistructured textual* source.

An example of a semistructured source — in which both types of structure are present — begins with a Web page that allows one to ask for hotels by filling in one or more of the following items: a location, a hotel chain, or a class of accommodation (economy, standard, luxury). This page is *semiqueryable*, because even though it allows for the construction of certain simple queries, there are many others that cannot be constructed.² The result of this query is a list of hypertext links to hotel pages, each of which contains the same basic data in more or less the same format. Each of these pages is a *semistructured textual* source of information, and each can be transformed into a set of data records, using parsing techniques, given that the format is known.

¹All product names mentioned in this document are the trademarks of their respective holders.

²For instance, one cannot ask, in a single query, for luxury hotels in Amsterdam and economy hotels in Paris.

1.2 The Need for Information Brokering

An *information brokering* system is one that provides coordinated access to a heterogeneous collection of structured and semistructured information sources. There are three key reasons why structured and semistructured information sources — and thus, information brokering systems — will be of rapidly increasing importance in the Internet and intranet worlds. First, human browsing time is both limited and, at least in the workplace, extremely expensive. Whereas *document retrieval* returns a body of text from which a human browser can extract some required data, *data retrieval* from structured and semistructured sources returns the required data itself. Thus, in performing tasks where the data requirements are well defined, there is significant economic pressure to make use of structured and semistructured sources.

Second, enterprises have large investments in legacy databases, and naturally want to leverage these in the context of the World Wide Web. This is especially true in light of the decentralization of corporate resources, and other trends in business process engineering. One straightforward way to leverage existing databases is simply to make them accessible to employees via Web interfaces. But the potential for leverage goes much further, when one considers the variety of ways in which legacy databases can be used in combination with other enterprise information resources, and with information from sources on the Internet. One role of information brokering systems is to facilitate the creation and maintenance of applications that rely on such combinations of information sources.

Third, queryable information sources are an essential requirement for the evolution of software agents that provide services within the context of the Internet or an intranet. By *services*, we refer not just to document retrieval, but to the full gamut of services that can be built around networked information. To provide a travel planning service or a stock tracking service, an agent must be able to retrieve data about a specific domain, in a form it can make use of — which requires access to structured and semistructured information. (Although natural language processing technologies have made impressive advances in recent years, there is still a long way to go before a program could reliably obtain this required data from *unstructured* text.) Moreover, many agents will need to access a wide and changing variety of information sources (consider, for example, an agent that finds the best available price for some product). As new sources become available, they will need to quickly acquire access to those sources. Information brokering systems can provide this access in a way that can be reused by numerous application agents.

1.3 Information Brokering and Agent Technology

Software agents will be very active *consumers* of the services provided by information brokering systems. We also argue that agent technology yields a promising approach to constructing the *providers* of these services. That is, the individual information sources as well as the brokering component(s) can very naturally be instantiated as agents, and their efforts coordinated by using the mechanisms of an agent architecture.

One other general observation may be made here regarding the relationship between infor-

mation brokering and agent technology: challenges encountered in coordinating access to information agents may be viewed as special cases of those encountered in coordinating the activities of all types of agents. Thus, many of the techniques developed to coordinate the satisfaction of a query by multiple agents may well be applicable to the more general problem of facilitating the activities of agents in completing various other types of tasks.

To summarize, the role of queryable information sources will expand rapidly in the context of Internet and intranet resources. Their use in these contexts poses several interesting challenges. Addressing these challenges is the focus of the Information Broker project, developed within the framework of the Open Agent Architecture (OAA). Following a brief discussion of these challenges in the next section, the remainder of this paper describes the system that has been developed in the initial work on this project. Section 3 provides an overview of the system, with background information about the OAA. Section 4 describes the system's architecture, and subsequent sections describe its individual components.

2 Challenge Areas

Our focus in this work has been on providing capabilities in two areas: *mediation* allows for the transparent interoperation of heterogeneous information sources; flexible *retrieval strategies* address issues such as long access times and unreliability of Internet resources.

2.1 Mediation

Mediation is a process that permits a requestor to get information from a wide variety of sources, without having to be aware of the identities, locations, schemas, access mechanisms, or contents of those sources. A component that performs mediation presents a single schema to its requestors, accepts queries expressed in that schema, and handles all the details of getting the appropriate data from the relevant information sources — each of which is likely to operate with a different schema.

The capabilities provided by a mediator may be broken down into three subareas: delegation, translation, and optimization. *Delegation* is the process of selecting the appropriate sources from which to satisfy each subquery of a given query. *Translation* of each subquery into the schemas of the selected sources must take place before the subqueries are sent to the sources — and results returned from the sources must be translated back into the schema of the original query. *Optimization* results in a query execution plan that obtains results from the selected sources as efficiently as possible, and exploits parallelism wherever possible.

It should be noted that, in their essence, the problems of mediation are not unique to networked environments. Some of these problems have already been studied for some time, under the heading of *heterogeneous databases*. However, the emergence of the Internet has triggered a renewed interest in these problems.³ This is not only due to the massive volume

³Other projects currently exploring these issues are described in [1], [5], [6], [7], [8], [12], and [13].

of data that is now becoming accessible over the Internet, but also because of new aspects of these problems that occur in the context of the Internet.

For example, the rapid creation (or discovery) of new information sources on the Internet, and frequent restructuring of existing sources, make it very valuable for a mediator to be able to accommodate new sources without going offline for reconfiguration. The absence of any centralized control over information sources implies that a mediator must be able to accommodate considerable variation in their schemas. The overhead involved in making HTTP connections to a number of different sites requires that a mediator be able to plan a retrieval so that it requires a minimal number of different sources.

2.2 Retrieval Strategies

In addition to the problems of mediation, Internet and intranet environments pose a number of other challenges for integrated information systems. For instance, the *unreliability* and *uneven quality* of Web sources may call for the exploitation of redundant or overlapping sites. Wide variation in the access mechanisms of *semistructured* sources suggests that strategies are needed to insulate the mediation component from this variation.

The use of semistructured textual sources on the Web also introduces a new set of time factors, because first, a very large number of URL (Universal Resource Locator) accesses may be required to process a single query, and second, parsing the text from a single site can itself be time consuming.

Our approach to addressing these issues includes the use of caching for Web-based sources, and several related retrieval strategies, which may be specified by information requestors. The use of caching raises additional issues, such as when cache updates should take place, and at what level of granularity.

3 The Information Broker System

To explore relevant issues, we created an information brokering architecture that integrates elements from software agent technology, database technology, and Internet retrieval technology. To demonstrate this approach, we constructed a prototype system, in which a Broker agent coordinates access to a variety of information resources in the domain of travel.

The prototype system illustrates two different ways in which the Broker agent can be used:

- A direct query interface allows the user to enter specific queries about hotels, relays the queries to the Broker, and formats the results for the user. In the prototype system, this interface is provided by a Web page.
- A service-providing agent insulates the user from query details, and can employ a user model in obtaining and evaluating the information needed for a specific task. In the

prototype system, this is demonstrated by the Travel Planner Agent (TPA), which accepts some basic trip constraints from the user, obtains hotel and flight data from the Broker, and uses that data to formulate desirable itineraries.

In both cases, the Broker obtains and integrates the requested information from several heterogeneous sources, each of which participates in the system as an independent agent. These include several Web-based sources providing hotel information, a relational database of flight information, a Web-based source that provides the latest available rates of currency exchange, and a corporate database listing rates for hotel chains which grant a discount.

Broker features allow for expression of queries in broker or in source schemas, the use of conversions, normalizations, and other basic domain knowledge in queries, flexibility in the means of retrieval, and *persistent query* capabilities. These features are explained in greater detail below.

The following subsection gives a brief overview of the OAA, the framework in which the agents of the brokering system operate.

3.1 The Open Agent Architecture

The Open Agent Architecture provides a framework for integrating a society of software agents, each possessing a high degree of independence and autonomy, within a distributed environment. A collection of agents satisfies requests from users, or other agents, by acting cooperatively, under the direction of one or more *facilitators* (which are themselves agents of a special type).

The system's architecture uses a hierarchical configuration in which each application agent connects as a client of a facilitator. Facilitators provide content-based message routing, global data management, and process coordination for their set of connected agents. Facilitators can, in turn, be connected as clients of other facilitators.

Agents share a common communication language and a number of basic structural characteristics and capabilities. An agent library provides this common functionality. For example, every agent can install local or remote triggers on data, events or messages; manipulate global data stored by facilitators; and request solutions for a set of goals, to be satisfied under a variety of different control strategies. In addition, the agent library provides functionality for parsing and translating expressions in the Interagent Communication Language, and for managing network communication using TCP/IP.

The OAA's Interagent Communication Language (ICL) is the interface language shared by all agents, no matter what machine they are running on or what computer language they are programmed in. The ICL has been designed as an extension of the Prolog programming language, in order to take advantage of unification and backtracking during interactions among agents.

The OAA is described in greater detail in [3], [9], and [10].

4 System Architecture

As shown in Figure 1, the central functionality of the system is provided by the *Information Broker* agent, working in close cooperation with the *OAA Facilitator*. The Broker accepts requests (queries) from either a direct query interface or a service-providing (helper) agent, as shown at the left of the figure. While the demonstration system includes only one of each type, there is no limit in principle to the number of requestors the Broker can serve, except for the constraints imposed by processing time and communications bandwidth.

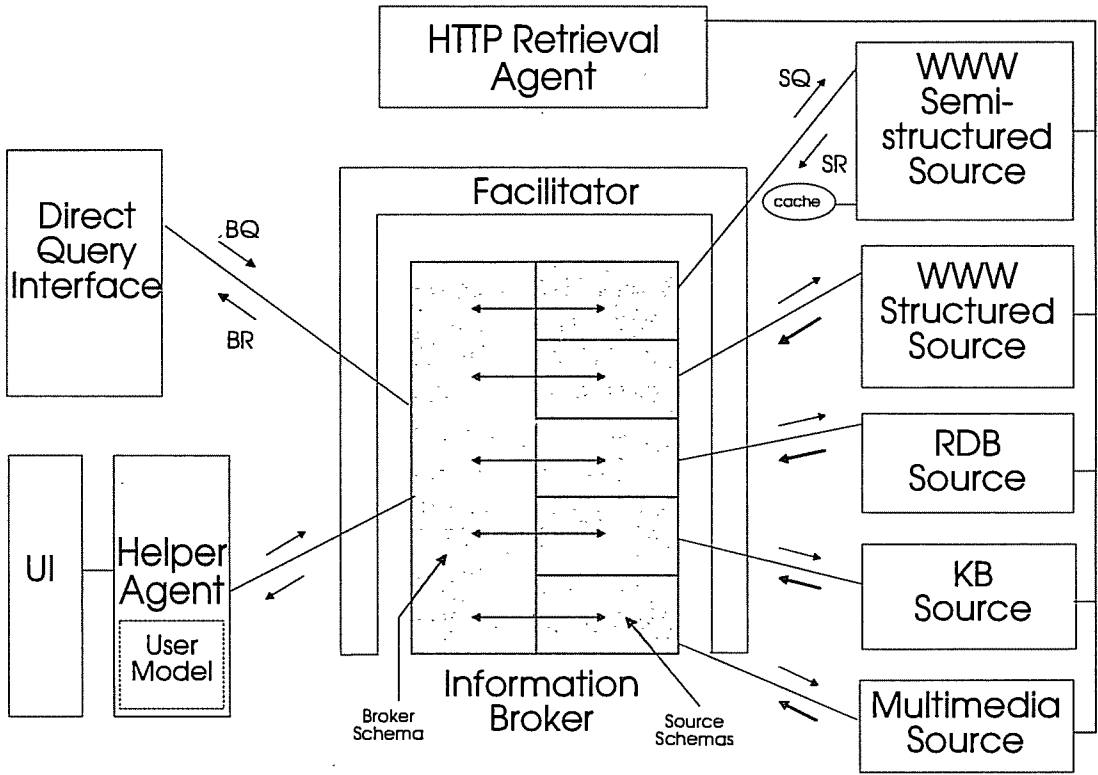


Figure 1: Information Broker system architecture

The Broker delegates, translates, and relays the appropriate subqueries to the available *source agents* (shown at the right of Figure 1), and then accepts the results and reintegrates them for return to the requestor. Each source agent is an encapsulation, as an OAA agent, of some information source. Web-based source agents make use of an HTTP retrieval agent, which is shown at the top of the figure.

In the figure, *BQ* (Broker Query) and *BR* (Broker Response) refer to items expressed in the broker schema, whereas *SQ* (Source Query) and *SR* (Source Response) refer to items expressed in a source schema — as explained in the next section. *RDB* abbreviates Relational

DataBase, and *KB* abbreviates Knowledge Base.⁴

5 The Broker Agent

The Broker agent provides transparent access to a collection of heterogeneous information sources in a given domain. *Transparent* means that an information requestor (a query interface agent or some other agent) need not be concerned with any details regarding the Broker's information sources. The Broker publishes a fixed schema, the *broker schema*, that is used in constructing queries, and this schema is all that is needed by a requestor to make use of the Broker's services.

Schema, here, is essentially the same as a relational database schema. From the developer's point of view, equivalently, it may be thought of as a collection of Prolog predicates.⁵ With respect to the broker schema, some of these predicates are implemented locally, at the Broker, whereas others are implemented remotely, at the sources.

Queries submitted to the Broker are expressions in the Interagent Communication Language (ICL), the language of the OAA. Each query is syntactically the same as a Prolog goal, usually a compound goal. For an OAA developer using the Broker's services, the ICL has two advantages: first, it is easily understood, and familiar to anyone who knows Prolog or the logical style of expressing database queries; second, it allows the developer to take advantage of Prolog-style unification and backtracking in expressing and processing queries.

When the Broker receives a query, it uses *schema mapping rules* to determine which parts of the query should go to which sources, and then to translate each subquery into the schemas of those sources to which it will be sent. (This process is described in greater detail below.) After the responses to the subqueries are received, the Broker translates them into the broker schema and integrates them into a single response, which is then returned to the requestor.

Thus, the Broker insulates an information requestor from the heterogeneity of schemas that is likely to be present in any collection of information sources. The Broker has this core functionality in common with heterogeneous database systems. However, the Broker goes further than many existing systems, in that it allows for the addition or removal of information sources at runtime. This can include sources of which the Broker has no prior knowledge.

To permit this dynamism in the participation of available information sources, the Broker makes use of capabilities provided by the OAA. When an information source agent comes online, it *registers* its presence with the Broker by calling one of the Broker's agent interface procedures, *broker_register_source*. As one of the arguments to this procedure, the source agent passes to the Broker a set of schema mapping rules, which contain the knowledge that

⁴The prototype system does not currently include knowledge base or multimedia sources.

⁵For this reason, we use the term *predicate*, where some readers with a database background might prefer *relation*. In most places in this paper, these two terms can be used interchangeably.

is needed by the Broker to translate between the broker schema and the source schema.⁶

This approach to schema mapping means that the Broker need have no prior knowledge of the schema of any of the participating information sources. What *is* required is that the developer of the source agent be aware of the Broker's schema, which, as mentioned earlier, has been fixed and "published". Thus, to allow for the dynamic participation of sources, an important part of the Broker's knowledge originates with the sources, each of which provides the mapping rules for its schema.

At the same time, because the schema mapping rules are *used* at the Broker, rather than at the source agent, it is possible to bring a source online with minimal changes to the original data or to the original capabilities of the source. This may be done by creating an agent *wrapper* around the original source component — an approach that is supported by the underlying OAA libraries and development tools.

Source agents that need to go offline may do so in an orderly fashion by calling another of the Broker's agent interface procedures, *broker_unregister_source*, to inform the Broker that the source is no longer available. However, it is also important to allow for sources that may die or become unavailable unexpectedly. Here again, the Broker uses mechanisms provided by the OAA Facilitator. To do this, the Broker installs a *trigger* on the Facilitator, for each source agent, which is set to fire whenever that source agent dies or becomes unavailable for any reason. The effect of the trigger is to send a notification message to the Broker, so that it can unregister the source and retract its schema mapping rules.

5.1 Domain Specialization

A Broker agent may be specialized for a given domain, by incorporating domain knowledge. To illustrate, our prototype Broker agent is specialized with basic knowledge relevant to travel, such as distances between major cities — data that is not likely to be included in any of the participating sources, but that is very likely to be useful in conjunction with the sources' data. Questions about this domain knowledge can then be included in queries submitted to the Broker. For example, using the direct-query hotel interface, it is possible to ask about hotels that are in a given city C, or in nearby cities (cities within a certain distance from C). By filling in knowledge gaps in this way, it is possible for the Broker to provide a retrieval capability that is greater than the sum of the individual information sources.

Procedures for converting and normalizing units represent another type of domain knowledge that is useful in a Broker. For example, in the travel domain, our prototype Broker provides procedures for converting and normalizing units of currency and the formatting of addresses, which vary widely from country to country. These procedures can be used in schema mapping rules, to ensure that results are returned in units and formats that are appropriate for the current user. This alleviates the need for each information source to provide these conversions

⁶These and other communications between the Broker and the source agents are handled through the services of the OAA Facilitator. We illustrate this, in Figure 1, by showing the Facilitator partially surrounding the Broker.

and normalizations, and thus makes it easier to prepare information sources to work with the Broker.

5.2 Schema Mapping Rules

Delegation and translation of queries by the Broker are determined by schema mapping rules, which are provided at runtime by each information source that connects to the Broker.

There are several types of schema mapping rules, the most important of which is the *broker predicate* rule. A broker predicate rule, which provides a partial definition of some predicate in the broker schema, has the syntax of a Prolog rule, and essentially the same semantics. Each mapping rule takes the form

$$B(\bar{X}) \leftarrow S_1(\bar{Z}_1), \dots, S_n(\bar{Z}_n)$$

where

1. $B(\bar{X})$ is a predicate in the broker schema.
2. Each of S_1, \dots, S_n may be taken from any of the following sets:
 - predicates in the source schema (i.e., the source providing the rule)
 - broker domain predicates
 - ICL (Interagent Communication Language) built-in predicates, including
 - standard comparison operators such as $</2$, $\leq/2$, $>/2$, and $\geq/2$
 - a small set of utility predicates such as *member/2* and *append/2*
3. As in an ordinary Prolog clause, each element of the argument lists $\bar{X}, \bar{Z}_1, \dots, \bar{Z}_n$, may be a variable or an instantiated value.

Broker domain predicates provide domain-specific knowledge, and most are purely extensional. For example, in the travel domain of the prototype system, *city_distance(City1, City2, Miles)* provides a table of distances between given cities.

An information source may be associated with multiple rules defining the same broker predicate, and multiple sources can define the same broker predicate.

Although the body of the broker predicate rule is characterized as a conjunction of predicates, it should be noted that in practice, these rules, like ordinary Prolog rules, are not strictly limited to conjunction. Disjunction, negation (that is, Prolog-style negation as failure), and a few other control operators are also allowed, but for any rule body containing these, standard equivalences of logic may be used to find an equivalent set of conjunctive rules.

5.3 Query Processing

The Broker's strategy in handling a query is to provide all possible answers, given its current schema mapping rules. It does this in such a way as to maintain Prolog semantics in performing query evaluation, and incorporates several straightforward optimizations that are important, considering the distributed nature of the information sources.

Of these optimizations, the most important is called *chunking*. The goal of chunking is simply to identify the largest subqueries that can be sent to an information source (after translation, as explained below) without changing the meaning of the original query. A *chunk* may contain broker predicates that are known to be satisfiable by the source, and ICL built-in predicates, but not broker domain predicates.

Chunking has three advantages. First, it requires that fewer communications occur between the Broker and an information source, reducing the overhead required to establish and complete communications. Second, it allows the information source to make better use of whatever optimization capabilities it may provide. Finally, it can significantly reduce the amount of data returned from a source, by allowing joins and other operations to be performed at the source, rather than at the Broker.

When the Broker receives a query, it takes the following steps to produce a *query execution plan*:

- Determine, for each predicate in the query that is not an ICL built-in, what set of sources provides solutions for that predicate.
 - If the predicate is a broker predicate, this is done by testing for unification of the predicate against the heads of the broker predicate rules. If it unifies with the heads of one or more of the rules associated with a given source, that source is placed in the set of sources for the predicate.
 - If the predicate is a broker domain predicate, then the Broker itself is regarded as the sole source of solutions.
- Determine which are the largest subqueries that can be treated as chunks, and which sources can handle each chunk. In this process, as an optimization, ICL built-in predicates (including arithmetic comparisons) are included with chunks to be solved by sources, wherever possible, rather than being solved by the Broker itself.
- For each chunk, rewrite it as a disjunction of translated subqueries, where each disjunct is the translation of the subquery for one of the sources that can handle that chunk. The translation of a chunk for a given source is obtained by substituting, for each broker predicate in the chunk, the disjunction of the bodies of the broker predicate rules, for that source, whose heads unify with the predicate.

In the resulting query execution plan, each translated subquery is labeled with the name of the source by which it is to be solved. The plan is then interpreted according to Pro-

log semantics, except that each translated subquery is solved remotely, by the appropriate information source.

5.4 Persistent Queries

An extremely useful service provided by the Broker agent is the *persistent query*. A persistent query is one that the Broker initially answers in the normal way, but then remembers and monitors against changes in the available information sources. When an information source is added, removed, or updated, the Broker checks the persistent queries to see if their results may have been affected by the change.

As mentioned in section 5, the Broker detects additions of sources by calls made to *broker_register_source*, and removals by calls to *broker_unregister_source*, or by the firing of a trigger installed on the Facilitator. Updates of sources are brought to the Broker's attention, with an indication of the source predicates involved, by means of triggers installed on the sources. In each of these cases, the Broker performs a straightforward analysis of each persistent query, using the schema mapping rules for the source in question, to determine if it may have been affected.

If so, the Broker sends a change notification to the user or agent that submitted the persistent query. For example, in the case of our direct-query interface, the result of a persistent query change notification is an email message to the user who entered the query. The email message informs the user that the relevant data has changed, reconstructs the user's query input, and allows him to resubmit his query with a single button push, so as to obtain the latest available results for that query. (The interactive nature of the email message is achieved by using a multipart MIME format for the message, and reading it using the mail handling component of a Web browser.)

The TPA deals with persistent query change notifications in a more comprehensive fashion, as explained in section 7.

5.5 Queries Expressed in a Source Schema

The Broker also provides the capability of answering queries that are expressed in the schema of an information source, rather than in the broker schema. This capability can have great value in systems where one or more information sources are provided by legacy databases.

The need for this capability arises when a legacy database provides a well-developed user interface to which users are accustomed, and it is desirable to retain that interface, while at the same time integrating the legacy database with the Broker's services. In this case, the goal is for the user to formulate a query using the legacy interface (which generates queries in the source schema of the legacy database), but to have that query answered by all the Broker's information sources, with the results returned to the legacy interface. This is made possible, with a minimum of new engineering effort, by arranging for the Broker to handle

queries in that source schema.

In this situation, the legacy database becomes one of the Broker's sources, and participates along with other sources in answering queries submitted to the Broker in the broker schema. But in addition, this legacy source may also act as an information requestor, submitting queries that are formulated in terms of its source schema. (A syntactic translation may be needed to place the query in ICL syntax, but in most cases, this is relatively straightforward to implement.) To make this possible, the legacy source must first submit, to the Broker, a collection of *source predicate rules*, which contain the knowledge needed by the Broker to translate queries from the source schema into the broker schema. These rules are similar to broker predicate rules, except that their heads are predicates from the source schema, and their bodies are compositions of predicates from the broker schema.

When a query expressed in a source schema is submitted, the Broker uses the source predicate rules to translate it into a query in the broker schema, and then solves that query in its usual way, with respect to all sources other than the legacy source. These results are then translated into the schema of the legacy source. The legacy source is not included in the Broker's normal mediation algorithms. Rather, the Broker submits the original query back to the legacy source just as it was. Finally, the Broker integrates all the responses and returns them to the source.

6 Source Agents

An information source agent that works with a Broker agent may be of several different types. The primary requirements are first, that it be able to handle the Prolog-style query syntax that the Broker generates, and second, that it be able to characterize its contents in terms of schema mapping rules. The first requirement is not a very restrictive one, as it is considered straightforward to translate from a Prolog-style syntax to SQL or other relational query languages. The second requirement means, roughly, that a source can be presented in terms of a relational model. (Although it should be possible to accommodate some sources that are inherently object-oriented or knowledge-based, this support has been left for future work.)

In meeting the second requirement, one must consider the origin and access mechanisms of the source's data, which can vary widely between information sources. For instance, some may be traditional relational databases, whereas others may contain data extracted from the Web. In the case of Web data, several types of data sources may be identified for our purposes. First, there are fully structured sources — relational databases that are accessible via a completely general query language, such as SQL. These are relatively uncommon at present. Second, there are semistructured sources. As mentioned in the Introduction, these may be either *semiqueryable* (accessible via a form-based query page), *semistructured textual*, or some combination of the two. semistructured textual sources are Web sites that include large collections of pages, each having enough structure, in HTML markups, headings, and/or other textual clues, to allow for extraction of a set of data elements.

In addition to traditional relational databases, implemented as Prolog databases, the prototype system makes use of several semistructured textual Web sources. To extract the data from the relevant pages, parsing techniques, based on context-free grammars, are used. A library of reusable code has been developed in support of these techniques.

6.1 Caching and Retrieval Strategies

As mentioned earlier, Web-based sources introduce challenges for data retrieval. Foremost among these are long access times (especially relevant to semistructured sources) and low reliability. In the Information Broker system, these are addressed through a combination of caching and flexible retrieval strategies.

6.2 Caching

Caching is accomplished both by batch update procedures and by update procedures that result from individual queries. A batch update procedure is one that runs offline (that is, independently of the source's activities in response to queries), and regenerates a complete cache. For a semistructured data source, a batch update could involve accessing hundreds or thousands of pages and parsing each one for relevant data, so this could easily take hours. Each of the Web-based information sources has been initialized in this way, and is set up so that query processing can continue normally by one agent, while a batch update is performed by a separate agent.

Cache updates can also be triggered by normal query processing. Whenever a query is satisfied (or partially satisfied) with data retrieved directly from the Web, this data is used to update the cache.

6.3 Retrieval Strategies

Flexible retrieval strategies are used to determine whether a query is satisfied entirely from a cache, directly from the Web at query time, or from a combination of the two. A request that comes in to the Broker can include, in addition to the query itself, a specification of a retrieval strategy, which will be propagated to each source agent that is employed in answering the query.

Data records in a cache are time-stamped, and some strategies are implemented in relation to these time stamps. In addition, a source agent can specify a default longevity for each predicate that it caches. For example, since room rates are relatively stable for hotels in our Web sources, the longevity for the relevant predicates is specified as two weeks. For the agent that provides current monetary exchange rates, however, longevities are set at 24 hours.

Currently, four retrieval strategies are supported. *All-from-web* calls for retrieval of all data from the Web, at query time, regardless of how long it might take. *All-from-cache* calls for retrieval exclusively from the cache. *From-web-if-expired* means that for each data element that is accessed in answering the query, its time-stamp is checked, and if it is older than the default longevity for that predicate, then the element is updated from the Web. *From-web-if-older-than(N)* is similar, except that the longevity is specified (as N seconds) for that particular query.

Consistent implementation of caching and retrieval strategies by the various source agents is supported by a library of reusable code.

7 The Travel Planner Agent

Even though a Broker provides a general, transparent means of accessing multiple data sources with a single query, there is still much that can be done to assist the user in making good use of the information provided by the Broker. A variety of service-providing components can be envisioned, which access the Broker on behalf of the user, thus insulating her from the details of query construction altogether. The instantiation of the Broker as a software agent helps to make this possible, because the Broker's services are thus made readily available to a wide variety of agents that come online.

The Travel Planner Agent provides an example of a high-level service-providing agent that insulates the user from formulating Broker queries. TPA's function is to formulate a list of possible itineraries for a trip, given a few basic trip constraints that are supplied by a user. These trip constraints include such things as dates of departure and return, destination city, and overall budget. Each itinerary prepared by TPA includes complete details about a possible choice of flights and hotel stays for the trip, and itineraries are scored and ranked according to user preferences.

To evaluate alternative itineraries, TPA maintains a user model indicating the user's preferences about a variety of travel parameters. For example, with respect to air travel selections, these parameters include such things as airline, class of travel, alternative airports, and time of day for departure. The presence of the user model also allows TPA to minimize the amount of new information required from the user when formulating an information request.

To minimize the user's effort in maintaining the user model, TPA has the capability of questioning the user about her experiences after a trip has been completed. Once a trip has been scheduled, TPA remembers the return date. Shortly after that date, using services provided by other OAA agents, TPA will send a questionnaire to the user by email, asking that she provide ratings for some of the parameters of the trip, such as a specific hotel or hotel chain. If a trip parameter has previously been rated by the user, she will be given an opportunity to update the rating; otherwise, she will be prompted to select a new rating.⁷ The questionnaire is presented as an HTML form, and a "Submit" button causes the user's

⁷For most parameters, ratings are integers between -5 and +5.

responses to be returned to TPA, for incorporation into the user model.⁸

TPA also provides a persistent query service, based upon that provided by the Broker. When it receives a persistent query change notification from the Broker, the TPA determines what set of itineraries was based on that query, automatically resubmits the queries relevant to those itineraries, obtains the new results, and recomputes the requested itineraries based on these results. It compares these itineraries to the results of the original request, which it has saved, to determine whether the resulting itineraries have been affected in any way. If so, it sends the new itineraries to the end user by email.

Note that TPA's persistent query service is more comprehensive than that of the Broker, in that it compares its newly generated itineraries against the previous ones, before generating a user notification. By contrast, when the Broker provides a persistent query change notification, it is based on advice from one of the Broker's sources that its data has changed in some way, and indicates that the query results *may have been affected* by the change. That is, the Broker does not determine with certainty whether the query results have been affected, but it gives the requesting user or agent the opportunity to make that determination. The reason for this difference is that the Broker is intended to handle large numbers of queries from (potentially) a large number of agents. Thus, it is not reasonable to expect the Broker to save the results of all queries handled.

8 Future Directions

Although it has not yet been exploited, our architecture allows for cooperation between multiple broker agents. This can be done by having one broker register as an information source with respect to another, and present mapping rules that allow its broker schema to be regarded as a source schema. These relationships between brokers could be organized around the structure of the relevant domains. For example, whereas our Broker directly uses sources in the subdomains of air travel and accommodations, among others, it would be natural to have these two subdomains handled by independent brokers.

The storage and use of broker predicate rules at the Broker, rather than at the individual sources, raises an opportunity for further simplifying the introduction of new information sources. Currently, each source has to be given the ability to process the ICL query syntax. Although this is easy when working with an implementation language for which the agent library is available, it could be a significant obstacle in other contexts. As a substitute, if a source agent exists without this ability, it would be possible for the Broker to perform the interpretation of the query syntax on behalf of that agent, in such a way that the agent need only respond to atomic requests (that is, where each request consists only of a single predicate). On the other hand, for those source agents which *do* handle the ICL syntax, but do not possess the ability to perform any optimizations, the Broker could provide more sophisticated optimizations of the compound requests that go to those agents.

More can be done, in the Broker, to ensure effective utilization of information sources.

⁸As of this writing, the implementation of the questionnaire capability is incomplete.

Currently, the use of a source in answering a subquery is determined by unification of the subquery predicates against the heads of broker predicate rules. Although this provides a useful degree of discrimination of sources, greater discrimination could be achieved by attaching additional characterizations of source contents. For example, it should be possible to specify that a source selection depends on an argument of a subquery predicate being a member of some fixed set of values, or within a certain range, if numeric. In addition, rules could be annotated with an approximate indication of what portion of the available data the source can provide, if known. This would allow the Broker, in some cases, to avoid sending requests to two or more sources that are likely to return the same data.

Finally, we would like the Broker to provide estimates of retrieval times to interested requestors. That is, it should be possible for a requestor to find out how long the wait might be for the results to a given query, with a given retrieval strategy. This would assist the requestor in selecting a retrieval strategy, and, assuming the requestor is a software agent, would allow it to manage its interactions with users more effectively.

9 Related Work

The growth of the Web has contributed to a renewal of interest in the problem of providing a uniform interface to multiple information sources, which has received considerable attention recently in both the AI and Database literature. Within this body of work, the Information Broker project is characterized by its emphasis on the following: dynamic availability of sources, supported by modular sets of schema mapping rules; ease of bringing new and legacy information sources online; transparent access to semistructured sources, supported by caching and retrieval strategies; and thorough integration with an existing agent framework.

In the database community, several systems are being built around the notion of a *mediator*, including CARNOT [4], TSIMMIS [2], and HERMES [14]. Broadly speaking, for each of a chosen set of queries, these systems provide a procedure to answer the query using the available sources. Given a new query, these systems attempt to answer it by relating it to the set of known queries. In the Information Broker, sources are described independently of the queries for which they will be used, which is less restrictive as to which queries are answerable, and also makes it easier to add or remove sources. TSIMMIS has also given attention to the use of semistructured sources, with an approach based on *yacc*-style recognition grammars.

The Information Manifold [8] shares our emphasis on dynamic addition and removal of sources. It too employs modular sets of source description rules, but relies on a different form of rule, where source schema predicates are *characterized* in terms of broker predicates (whereas, in the Information Broker, broker predicates are *defined* in terms of source predicates). There are interesting tradeoffs between the two approaches. In particular, although Information Manifold characterizations provide the broker with more information regarding the contents of each source, and thus allow for greater optimization in the selection of the sources that are relevant to a query, their use is computationally expensive, and the form of the rules must be more tightly restricted.

In the AI community, systems such as SIMS [1] and InfoMaster [6] are also based on explicit representations of the contents of information sources. In SIMS, these are based on mappings of source schemas to classes and attributes specified in the description logic language LOOM, and planning technology is used to generate query plans appropriate for the sources. SIMS' transformation rules, which guide the planning process, do not guarantee that a query-answering plan will be found if one exists. In addition, defining mappings using arbitrary logic rules, as in the Information Broker, provides greater flexibility than in SIMS. In the RETSINA (Reusable Task Structure-based Intelligent Network Agents) architecture [15], *information agents* play a role corresponding to that of our Broker agents, and use KQML in much the same way that our agents use ICL. In InfoSleuth [7], which employs KQML and KIF for communications, this role is implemented jointly by an Execution Agent and a Broker Agent.

10 Summary

Structured and semistructured information sources will be increasingly important as we learn to use the Internet and intranets in more sophisticated ways. Transparently coordinated access to heterogeneous, rapidly changing collections of these sources is a prerequisite for the evolution of many types of software agents.

We have described an approach and a prototype system that provide this access, within the context of an agent framework, the Open Agent Architecture. Information consumers and providers are instantiated as agents, and interact through the services of a Broker agent, which include mediation, dynamic addition and deletion of sources, and persistent queries. Provider agents present a variety of information sources as queryable databases, hiding details of access and storage, and provide caching and flexible retrieval strategies.

Our approach to information brokering fits naturally into the agent paradigm, and benefits from the autonomy it allows for individual providers and consumers, and the flexibility of interactions between them. In addition, the agents are able to take advantage of specific capabilities provided by the agent framework, including communication and coordination mechanisms, the Interagent Communication Language, triggers, and support provided for creating wrappers around legacy applications implemented in a variety of languages.

References

- [1] Yigal Arens, Chin Y. Chee, Chun-Nan Hsu, and Craig A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal on Intelligent and Cooperative Information Systems*, 2(2):127-158, 1993.
- [2] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yanniss Papakonstantinou, Jeffrey Ullman, , and Jennifer Widom. The TSIMMIS project: In-

- tegration of heterogeneous information sources. In *Proceedings of the IPSJ Conference*, Tokyo, Japan, October 1994.
- [3] P. R. Cohen, A. Cheyer, M. Wang, and S. C. Baeg. An open agent architecture. In O. Etzioni, editor, *Proceedings of the AAAI Spring Symposium Series on Software Agents*, pages 1–8, Menlo Park, California, March 1994. American Association for Artificial Intelligence.
 - [4] C. Collet, M. N. Huhns, and W Shen. Resource integration using a large knowledge base in CARNOT. *IEEE Computer*, 55(62), 1991.
 - [5] Hector Garcia-Molina, Dallan Quass, Yannis Papakonstantinou, Anand Rajaraman, Yehoshua Sagiv, Jeffrey D. Ullman, and Jennifer Widom. The TSIMMIS approach to mediation: Data models and languages. In *Proceedings of the Second International Workshop on Next Generation Information Technologies and Systems (NGITS '95)*, Naharia, Israel, June 1995.
 - [6] Infomaster home page. Available via World Wide Web URL <http://infomaster.stanford.edu/>.
 - [7] Infosleuth project index. Available via World Wide Web URL <http://www.mcc.com/projects/infosleuth/>.
 - [8] Thomas Kirk, Alon Y. Levy, Yehoshua Sagiv, and Divesh Srivastava. The information manifold. In *Proceedings of the AAAI Spring Symposium on Information Gathering in Distributed Heterogeneous Environments*, Stanford, California, March 1995.
 - [9] David L. Martin, Adam Cheyer, and Gowang-Lo Lee. Agent development tools for the open agent architecture. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, pages 387–404, Blackpool, Lancashire, UK, April 1996. The Practical Application Company Ltd.
 - [10] Douglas B. Moran, Adam J. Cheyer, Luc E. Julia, and David L. Martin. The open agent architecture and its multimodal user interface. In *Proceedings of the 1997 International Conference on Intelligent User Interfaces (IUI97)*, Orlando, Florida, 6-9 January 1997.
 - [11] Alexandros Moukas. *Amalthea*: Information discovery and filtering using a multiagent evolving ecosystem. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, pages 421–436, Blackpool, Lancashire, UK, April 1996. The Practical Application Company Ltd.
 - [12] Xiaolei Qian. Semantic interoperation via intelligent mediation. Final Report CDRL A009, Computer Science Laboratory, SRI International, Menlo Park, California, 1996. Unpublished project report.
 - [13] Michael Stonebraker, Paul M. Aoki, Avi Pfeffer, Adam Sah, Jeff Sidell, Carl Staelin, and Andrew Yu. Mariposa: A wide-area distributed database system. Sequoia 2000 Technical Report 95/63, University of California, Berkeley, CA, June 1995. Appeared in *VLDB Journal* 5, 1 (January 1996), pp. 48 – 63.

- [14] V.S. Subrahmanian, S. Adali, A. Brink, R. Emery, J. Lu, A. Rajput, T. Rogers, R. Ross, and C. Ward. HERMES: A heterogeneous reasoning and mediator system. Technical report, University of Maryland, 1995.
- [15] K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng. Distributed intelligent agents. In *IEEE Expert*, December 1996.

BT Technology Journal

Vol.14 No.1 JANUARY 1996

THEME Speech technology for telecommunications

Foreword by C Wheddon

Editorial by F A Westall, R D Johnston and A V Lewis

*F A Westall, R D Johnston and
A V Lewis*

Speech technology for telecommunications

W T K Wong et al

Low-rate speech coding for telecommunications

*P A Barrett, R M Voelcker and
A V Lewis*

Speech transmission over digital mobile radio channels

J H Page and A P Breen

The Laureate text-to-speech system — architecture and applications

M Edgington et al

Overview of current text-to-speech techniques: Part I — text and linguistic analysis

M Edgington et al

Overview of current text-to-speech techniques: Part II — prosody and speech generation

R D Johnston

Beyond intelligibility — the performance of text-to-speech synthesisers

K J Power

The listening telephone — automating speech recognition over the PSTN

M Pawlewski et al

Advances in telephony-based speech recognition

F Scahill et al

Speech recognition — making it work for real

R D Johnston

Are speech recognisers still 98% accurate or has the time come to repeal 'Hyde's law'?

D J Attwater and S J Whittaker

Issues in large vocabulary interactive speech systems

P J Wyard

Spoken language systems — beyond prompt and response

M P Hollier and G Cosier

Assessing human perception

A V Lewis and F A Westall

Whither speech? — The future of telephony

BT Laboratories
Martlesham Heath Ipswich Suffolk England IP5 7RE

Distributed by Chapman and Hall



The BT Technology Journal is a quarterly periodical of technical papers published by British Telecommunications plc to promote an awareness, among workers in similar fields world-wide, of the Research and Development undertaken by BT in telecommunications and related sciences.

EDITORIAL BOARD

G White *BSc PhD CEng FIEE SMIEEE, Chairman*
J R W Ames *MSc CEng MIEE*
E L Cusack *BSc PhD*
I G Dufour *Eurlng CEng FIEE*
P G Flavin *CEng MIEE*
P W France *MSc PhD*
J R Grierson *MA PhD CEng FIEE*
R C Nicol *PhD CEng FIEE*
S G Stockman *BA PhD*
A M Jell *BA, Editor*
D N Clough *MA, Assistant Editor*

Enquiries to the Editor: (01473) 623232, facsimile (01473) 620915, e-mail: btj@ipswich.sac.co.uk
Internet access to the BT Laboratories information pages is available on: <http://www.labs.bt.com>

Unless otherwise stated, copyright of the papers appearing in the Journal is reserved by British Telecommunications plc. The views of contributors are not necessarily those of the Editorial Board, do not necessarily represent BT policy, nor reflect an endorsement for any commercial products.

The BT Technology Journal is distributed by Chapman and Hall, 2-6 Boundary Row, London SE1 8HN, UK.

The Journal is published four times per year in January, April, July and October. Subscription prices for 1996 are: print + Internet access: \$214 (USA/Canada) £126 (EU) £140 (all other countries); print only: \$185 (USA/Canada) £108 (EU) £122 (all other countries). Subscription prices for individuals are (print only): \$95 (USA/Canada) £54 (EU) £54 (all other countries). Individual subscriptions must be paid for by personal cheque or credit card.

Any payment in US\$ should be made to Routledge, Chapman and Hall Dollar Account: 051-70700-4, Barclays Bank New York Ltd., 300 Park Avenue, New York, NY 10022, USA.

Subscription rates to USA include airfreight to New York and second class postage thereafter. All other territories outside UK and Europe will be served by accelerated surface post.

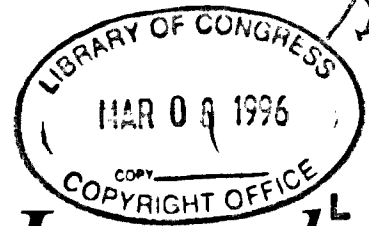
Second class postage paid at Rahway, NJ. Postmaster: send address corrections to The BT Technology Journal, c/o Mercury Airfreight International Ltd Inc., 2323 Randolph Avenue, Avenel, NJ 07001, USA (US mailing agent).

All subscription enquiries should be made to Chapman and Hall Subscriptions Department.

All enquiries concerning editorial matters should be made to the Editor, SAC Technographic Ltd, 38 Anson Road, Martlesham Heath, Ipswich, Suffolk IP5 7RG (for voice, fax, e-mail details, see above).

BT Laboratories

Martlesham Heath Ipswich Suffolk England IP5 7RE



BT Technology Journal

Vol.14 No.1 JANUARY 1996

THEME Speech technology for telecommunications

Foreword by C Wheddon

Editorial by F A Westall, R D Johnston and A V Lewis

*F A Westall, R D Johnston and
A V Lewis*

Speech technology for telecommunications

9

Speech is the easiest, most expressive and most natural means of human communication. Most of us have received intensive training in using it from the day we were born! But speech is more than just a way of transmitting words or ideas — it conveys the essence of human emotion, moods, and personality. It is BT's core business, accounting for over 90% of revenues. It is also our primary means to access the 26 million customers of the UK telephone networks, and to around a half a billion telephone users world-wide. This paper introduces the key speech technologies, described in detail in the associated papers in this issue, and makes some personal predictions about future trends and challenges in this important, exciting and far-reaching field.

W T K Wong

Low rate speech coding for telecommunications

28

Over the last decade major advances have been made in speech coding technology which is now widely used in international, digital mobile and satellite networks. The most recent techniques permit telephone network quality speech transmission at 8 kbit/s, but there are still demands for even lower rates and more flexible, good quality coding techniques for various network applications. This paper reviews the developments so far, and describes a new class of speech coding methods known as speech interpolation coding which has the potential to provide toll-quality speech coding at or below 4 kbit/s.

*P A Barrett, R M Voelcker and
A V Lewis*

Speech transmission over digital mobile radio channels

45

The design of a speech channel for digital mobile radio applications is a trade-off between the key performance dimensions of speech quality, robustness to errors, delay, complexity and bit rate. An appropriate balance is often difficult to achieve, but is vital to customer satisfaction. This paper identifies the considerations in selecting a speech codec for mobile telephony applications, outlines techniques for robust and efficient speech transmission over a digital mobile radio channel and discusses how the resulting performance can be assessed. Throughout the paper, the half-rate GSM digital mobile radio system is used as an example.

Spoken language systems — beyond prompt and response

P J Wyard, A D Simons, S Appleby, E Kaneen, S H Williams and K R Preston

Spoken language systems allow users to interact with computers by speaking to them. This paper focuses on the most advanced systems, which seek to allow as natural a style of interaction as possible. Specifically this means the use of continuous speech recognition — natural language understanding to interpret the utterance, and an intelligent dialogue manager which allows a flexible style of 'conversation' between computer and user. This paper discusses the architecture of spoken language systems and the components of which they are made, and describes both a variety of possible approaches and the particular design decisions made in some systems developed at BT Laboratories. Three spoken language systems in the course of development are described — a multimodal interface to the BT Business Catalogue, an e-mail secretary which can be consulted over the telephone network, and a multimodal system to allow selection of films in the interactive TV environment.

1. Introduction

No science fiction image of the future is complete without the ever-present personable computer which can understand every word said to them. In spite of these popular media images, the goal of completely natural interaction between humans and machines is still some way off.

Interactive voice response (IVR) systems, which provide services over the telephone network, have been available since the mid-1980s. Initially they were restricted to interactive TouchTone® input with voice providing the response to the user. The use of such services was therefore limited to the population with TouchTone keypads. More recently applications using automatic speech recognition (ASR) have been developed. These often simply allow the option of spoken digit recognition as an alternative to keypad entry, thus allowing the service to be launched even in areas where TouchTone penetration is poor. Moving on from such systems the words which are spoken can be matched to the service. This allows these ASR-based services to be more user-friendly than their TouchTone counterparts because the user can directly answer the question: 'Which service do you require?' with 'weather' or 'sport' rather than 'for weather press 1 for sport press 2', etc. However, they still rely on selection from a predetermined menu of items at any point in the dialogue.

More sophisticated services are now becoming possible using emerging larger vocabulary speech recognition technology. However, it is not sensible to simply extend the menu-based approach to accommodate larger vocabularies.

Although well-engineered simple applications may be easy to use, more advanced services are likely to have complicated menu structures. If information can only be provided one item at a time, using a 'prompt and response' dialogue, rigid interaction styles may steer the user through a complex dialogue. This can result in the user becoming lost, or ending up with the wrong information. These problems are particularly significant for inexperienced users. On the other hand, experienced users may become bored by the large number of responses needed when they know exactly what they want. The menu-based structure required by systems which rely on isolated word input is often the limiting factor for new services. This limitation of the user interface is one of the greatest barriers to the usability of many IVR services.

Moving beyond the menu-style interaction towards conversational spoken language will allow users to express their requirements more directly and avoid tedious navigation through menus. This approach will also allow the user to take control of the interaction rather than using the more common 'prompt and response' dialogue.

BT is interested in the development of spoken language systems (SLS) to provide a key competitive advantage. SLSs allow users to interact with computers using conversational language rather than simply responding to system prompts with short or one word utterances. With the rapid increase in competition, service differentiation becomes a key factor in gaining market share. Systems

187

which allow users 24-hour remote access to information provide a very useful service for people who are in different time zones, or away from their office, or who need information immediately during unsocial hours. SLSs can be used to automate such services and also those which currently require human operators, thus freeing their time to deal with difficult situations where more complex, or more personalised advice is needed.

Current trends in information networking and the phenomenal growth of the Internet bring their attendant problems for our customers in keeping up with technology, finding what they need, and using information to their best advantage. Spoken language system technology can greatly enhance our customers' ease of access to information, thus increasing network revenue through new and increased usage. Systems which combine several modes of input and output, such as speech, graphics, text, video, mouse-control, touch and virtual reality, are known as multimodal spoken language systems. These allow far greater freedom of expression for users who, as a result, should feel more comfortable and less as though they are 'talking to a computer'. They are able to point, use gestures, speak, type; whatever comes most naturally to them. Spoken language systems will become increasingly important in the near future as progress in technology becomes more widely available.

The goal is to be able to build systems which are not restricted only to those motivated users who are prepared to spend time learning the language the machine understands. These new systems can be used by anyone who wants occasional access to a particular service. They will also help the user successfully gain the information or service they require by simply calling a number and asking for what they want. In fact, the aim is to put back some of the intelligence which existed in the network 50 years ago when a user simply lifted the handset and asked to be connected to the service or number required.

This paper discusses the design and implementation of spoken language systems and is organised as follows. Section 2 gives an outline of the architecture of an SLS. Section 4 describes the components of an SLS in some detail, giving concrete examples from current systems. Section 3 discusses some of the systems currently under development at BTL. These include a multi-modal system for access to the BT Business Catalogue, a speech-in/speech-out system for remote e-mail access and a system for accessing information about films. Section 5 discusses future work which needs to be carried out to improve the quality and usability of SLSs, and section 6 draws some conclusions.

2. System overview

This section outlines a typical spoken language system architecture, from the information processing point of view (platform and inter-process communication issues are not dealt with to any great extent in this paper). The architecture and the key processing components are outlined.

The most basic form of SLS, a speech-in/speech-out (rather than multimodal) system, requires at least the following major components (described briefly below and in more detail in section 4).

- Speech recognition — to convert an input speech utterance to a string of words.
- Meaning extraction — to extract as much of the meaning as is necessary for the application from the recogniser output and encode it into a suitable meaning representation.
- Database query — to retrieve the information specified by the output of the meaning extraction component. Some applications (e.g. home banking) may require a specific transaction to occur. Many applications may be a mixture of database query and transaction processing.
- Dialogue manager — this controls the interaction or 'dialogue' between the system and the user, and coordinates the operation of all the other system components. It uses a dialogue model (generic information about how conversations progress) to aid the final interpretation of an utterance. This may not have been achieved by the 'meaning extraction' component, because the interpretation relies on an understanding of the conversation as a whole.
- Response generation — to generate the text to be output in spoken form. Information retrieved by the database query component will be passed to the response generation component, together with instructions from the dialogue manager about how to generate the text (e.g. terse/verbose, polite/curt, etc).
- Speech output module (text-to-speech synthesis or recorded speech).

At its simplest, processing consists of a linear sequence of calls to each component, as shown in Fig 1. A typical output of each stage from an application which accesses the BT Business Catalogue is shown. It is not necessary to understand the output of the 'meaning extraction' component in detail to realise that meaning extraction can be a non-trivial exercise. The simple linear sequence shown in Fig 1 is, in general, too inflexible. It is better if the dialogue manager is given greater control, to call the other components in a flexible order, according to the results at

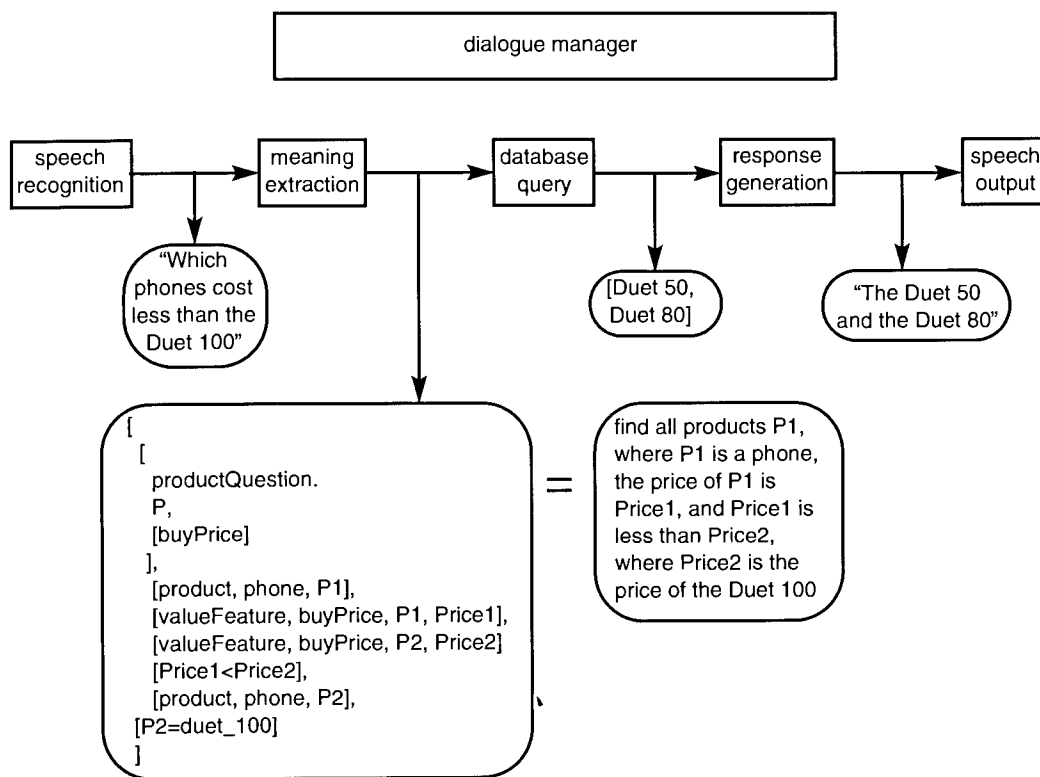


Fig 1 Example of a linear process flow in a spoken language system.

each stage. This leads to an architecture of the type shown in Fig 2.

The need for this more flexible architecture is illustrated by the processing sequence in Fig 3 which shows the dialogue manager as control centre, calling each component in an order determined by the results of processing at each stage. Although every processing stage is passed through the dialogue manager, this is not included in the sequence unless some non-trivial decision or action is taken. The example given in Fig 3 is largely driven by limitations of the recogniser, but the need for this sort of flexible architecture goes far beyond this. It will eventually enable the dialogue manager to act in an intelligent manner, co-ordinating the components and combining their outputs in a nonlinear manner.

So far in this section, the discussion has covered speech in/speech out systems. However, systems such as the BT

Business Catalogue access system (see section 3.1) are multimodal and require a screen and a means of inputting text and mouse clicks and outputting text and graphics. These components must be added to the architecture shown in Fig 2 and the dialogue manager and response generator must be upgraded to deal with the extra modalities. However, most of the discussion of this section applies equally to multimodal systems.

3. Example systems

In this section three spoken language systems under development at BT Laboratories are described:

- access to the BT Business Catalogue, known as BusCat — this was the first multimodal continuous speech input spoken language system,

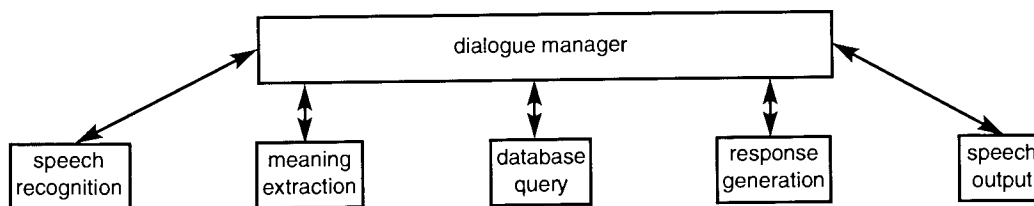


Fig 2 Role of a dialogue manager in a spoken language system.

- an e-mail access system, which is speech in/speech out only, but has the conversational features described in this paper — it is also a dial-up service over the telephone network,
- a film access system, in which users will be able to select films and videos using continuous speech and button pushes on a remote control handset — this system is targeted at the interactive TV environment.

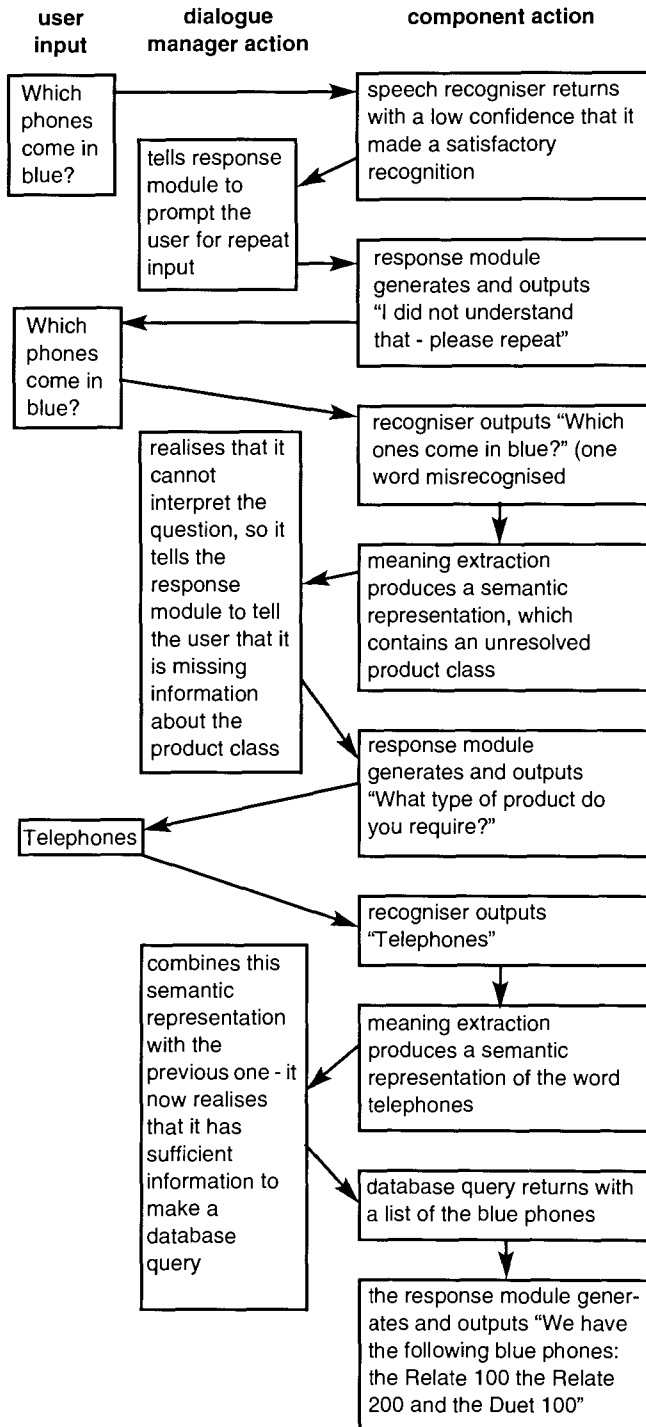


Fig 3 Nonlinear process flow in spoken language systems.

3.1 BusCat

The SLS BusCat provides direct access to a subset of the BT Business Catalogue, which covers a range of products such as telephones, answering machines and phone systems. The user has a screen displaying a Netscape WWW browser and speech input/output facilities. All the normal WWW browser features are present, such as the ability to click on links to other pages, and a display consisting of mixed text and graphics (see Fig 4).

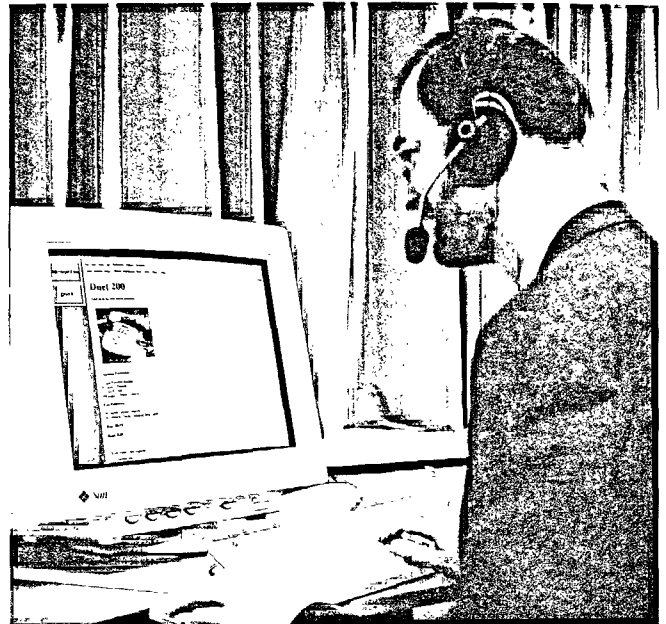


Fig 4 The SLS BusCat system in use.

Additionally, in this system users may use continuous speech input, type questions into a free-text window, and listen to speech output generated by a text-to-speech (TTS) system. This multimodal interface enables users to request specific information about the products in the catalogue, or to browse through the catalogue.

The overall structure of the system is shown in Fig 5. The system can cope with multiple simultaneous users.

In addition to its internal knowledge bases, the system has the capability to access external databases across a network. One application for this might be to provide a multimodal interface for such databases. Another is to allow the internal knowledge bases to be periodically updated from an external database.

The speech recogniser used is BT's Stap recogniser [1], and the text-to-speech system is BT's Laureate [2] system.

The example in Table 1 gives a flavour of what it feels like to interact with the system. Here the user is already logged on to the system. From each WWW page there is a choice of:

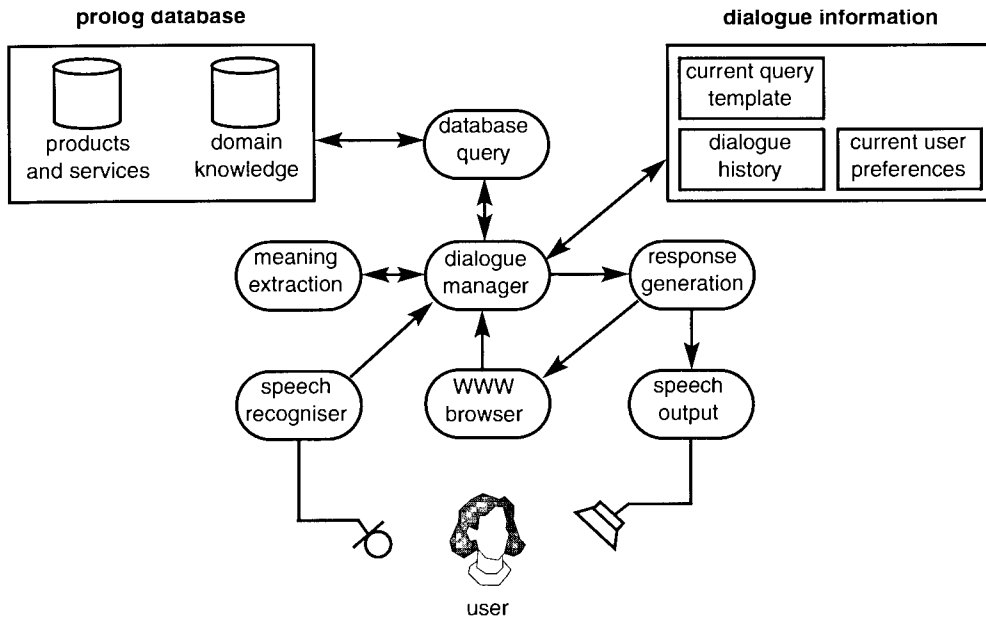


Fig 5 Architecture of BusCat.

Table 1 An example session with BusCat.

User input	System response
'What is on-hook dialling?'	Textual (and optionally spoken) explanation of on-hook dialling: 'Time spent waiting for someone to answer the phone can often be lost time. But with this feature, you can dial without picking up the phone handset, leaving you free to carry on with something else until the second your call connects,' and a list of five phones which have this feature: Vanguard 10e, Relate 200, Relate 300, Relate 400, Converse 300.
'Which phones have on-hook dialling and cost less than 60 pounds?'	Text: 'The following products meet your requirements,' and a list of four phones, each with a small picture, a short description and a price (Vanguard 10e, Relate 200, Relate 300, Converse 300).
'Which ones come in grey?'	Text: 'The following products meet your requirements,' and a list of three phones, each with a small picture, a short description and a price (Vanguard 10e, Relate 200, Relate 300).
The user clicks on the link next to the picture of the Relate 200.	The system responds with a large picture of the Relate 200, a full description including all its features and a price.

- speaking to the system,
- clicking on a link,
- typing into the free-text field.

In the interaction the user wants to know what on-hook dialling is. Having received an explanation of this feature, he decides he wants a phone with on-hook dialling which costs less than £60. Then he remembers he also wants it in grey to match his living room. He finally selects the Relate 200 telephone.

3.2 E-mail access

BT is very interested in the mobile telephony market. Speech-only natural language systems are very attractive to this market because people want to be able to keep in touch while on the move. They are likely to only have a mobile phone with which to do so. Speech access to information can have an added safety advantage over TouchTone interfaces as well as helping to improve the usability of the service.

E-mail has been chosen as the vehicle for producing a speech in/speech out natural language system because the information source (the users own e-mail) is readily available. An existing system allows selected users to access their e-mail using TouchTone commands, thus providing information on whether natural language access confers real benefits to the users. It will certainly enable queries which are not possible currently, such as: 'Do I have any messages from John?'

The main differences between the e-mail service and the BusCat service are as follows. The e-mail service uses telephony speech only, rather than a multimodal system. The dialogue is more robust to errors and will allow the user or the system to take the initiative whereas the Business Catalogue system is based on the user taking the initiative. The e-mail system architecture is being designed to decouple domain-independent and domain-dependent parts of the system, and the same approach is being used for the

film access system (see section 3.3). This means that in the future there will be a good level of confidence in being able to produce spoken language systems for new domains more quickly than at present.

An example of a dialogue the system might have with the user is shown in Table 2.

Table 2 An example session with an e-mail service.

User	System	Note
(dials up system)	Good morning Alison. You have ten new messages and 20 old messages. Your first new message is from Mike Harper, regarding, Thanks for demo.	The default startup is to tell the user how many old and new messages there are and to start listing the new ones.
Read it.	Message is: 'Thank you for your demo yesterday. The customers were most impressed.'	
Forward it to Peter Smith and Sandra	Sandra Jones is already on the distribution list	Resolution of partially specified name. Check on preconditions for forwarding
	Forwarding message to Peter Smith	
	Next message is from Colin North, regarding, Team Meeting.	System assumes you want to carry on listing new messages unless you tell it otherwise.
Forward to Language Group	Forwarding message to all the people on your distribution list called 'Language Group'.	Does not confirm on forwarding a message.
	Next message is from Laurence Cooper: Plans for Workshop	
Let's hear it.	Message is: 'Hi Alison, Have you completed the plans yet?'	
Reply	I have four possible responses, yes, OK, no, or I'll call you.	Choice of several stock replies
Tell him yes	Reply to Laurence Cooper: Text is 'Yes'. Please confirm.	Confirms before sending a reply
OK	Reply sent. Next message is from...	
Stop.	... OK. What would you like?	Detection of user speaking terminates prompt. Listing new messages ended by user.
Have I any recent messages from Steve Brown or Steve Clark	In the last week you have received 5 messages from Steve Brown and 4 from Steve Clark	Slightly more complex user query. System makes explicit what it thinks of as 'recent'.
List the ones from Clark.	Message 1 is about Meeting on the 31st Message 2 is about Agenda for team meeting	Resolution of partial specified name.

3.3 Film access for interactive multimedia services

Interactive multimedia services (IMS) are currently being implemented by BT in Colchester and Ipswich [3]. The usability of the interface is still constrained to navigation using a TV remote control. This means that multiple layers of menus need to be traversed to get to the information required. Spoken language access would allow users to go straight to the information they are searching for, without requiring them to learn complex navigation procedures.

The video-on-demand subset of the IMS, which consists of over 4000 hours of material, including films, educational programmes, children's programmes, etc, was chosen. The SLS will allow users to give instructions such as: 'I want a comedy film starring Harrison Ford'. Part of the benefit of developing such a system, is to ensure that the generic SLS framework is truly domain independent.

There is currently a text-based interface to the Internet movie database [4]. This allows users to enter queries such as: 'Tell me the ratings of comedy movies starring Harrison Ford'. The system performs the meaning extraction using a caseframe parser (section 4.2). This allows it to pick out the salient information from among extraneous words.

It seems likely, from human analysis of typical queries about films, that this method is suitable.

An issue yet to be addressed is how to best reconcile the advantages of using speech, with the limitations of current recognition technology. This is clearly illustrated in the present example, since the text-based interface can query the database of over 50 000 films and 100 000 cast names. No speech recogniser yet built can cope with this range of vocabulary. The obvious solution is to restrict the size of the database. A possible step in the right direction would be to couple the 'meaning extraction' component and recogniser much more closely, so that meaning extraction and recognition happen simultaneously. This might enable the recogniser to cut down the vocabulary size 'on the fly'. For example, given the input sentence: 'Which comedy movies star Burt Lancaster,' it could be established straightaway that the user was talking about comedies, then only about cinema films, and finally that the user was only interested in an actor. Therefore, by the time the recogniser gets to the name 'Burt Lancaster,' the number of possible words has reduced considerably.

This is the subject of further research and is discussed in more detail in the next section.

4. Components of a spoken language system

4.1 Speech recognition

The job of a speech recogniser is typically thought of as converting a speech utterance into a string of text. The internal workings of speech recognisers are explained in some depth elsewhere [5]. This section looks at the recogniser's place within an SLS, and, in particular, at the language model (LM) the recogniser uses and the form of output that it provides.

A language model embodies information about which words or phrases are more likely than others at a given point in a dialogue.

One might imagine that in a system that accepts fluent language, for example an automated travel agent, the speech recogniser might need only one language model, that of the entire English language. It could then recognise anything that anyone said to it (assuming they are speaking English) and could inform the dialogue manager accordingly. Speech recognition is not yet accurate enough and a model of the entire English language does not exist. Instead, to get a working system, the recogniser must be given as much help as possible. It must be given hints about what the user is likely to say next to improve the chances of correctly recognising what has been said. If the dialogue manager knows that the customer wants to go on a cruise and has just asked them where they would like to go, it should prime the recogniser to be expecting a response that may well concern one of a number of specified cruise ports and, by the same token, is unlikely to have anything to do with backpacking in Nepal.

Recognisers use a language model to hold this information. There are a number of ways that the dialogue manager can update the information that the recogniser is using. The simplest option is just to tell the recogniser which of a predefined set of language models to use. Then there is a range of possibilities for updating or modifying predefined language models. Certain portions of a grammar can be made more likely than others, e.g. phrases to do with the time of day might be expected at one stage in the dialogue, while requests concerning holiday destinations might be more likely at another, and the language model can be adjusted accordingly. Alternatively, the recogniser might have a grammar, a portion of which allows the sequence 'from <airport> to <city>' where the range of possible airports and cities is specified by the dialogue manager only immediately prior to recognition. This could be dependent, say, on which country is under discussion.

The following subsections discuss in more detail:

- language models,
- perplexity of a language model,
- advantages and disadvantages of language models,
- loading language models into the recogniser,
- output from the recogniser.

4.1.1 Language models for the recogniser

The primary knowledge source for the speech recognition component is a set of statistical models, known as hidden Markov models or HMMs, which encode how likely a given acoustic utterance is, given a string of spoken words. A recogniser can decode a speech utterance purely on the basis of this acoustic-phonetic knowledge, and this is basically what happens in the case of single isolated-word recognition. However, in the case of recognising a string of words (which form part of a spoken language), the recogniser can use a second knowledge source, namely the intrinsic probability of the given string. This second knowledge source is known as the language model.

To take a classic example, a given utterance may have almost equal acoustic-phonetic probabilities of being 'recognise speech' or 'wreck a nice beach'. However, the intrinsic probability of the first string is likely to be higher than that of the second, particularly if this utterance came from the domain of a technical journal on speech technology.

This can be expressed mathematically as follows. Let X be the acoustic utterance and let S be the sentence to be recognised. The task is to find the sentence S for which the posterior probability $p(S|SX)$ is a maximum. Using Bayes' rule this can be rewritten as a requirement to find:

$$\operatorname{argmax}_S \{p(S) * p(X|S)\}$$

In this formula, $p(S)$ is the prior probability of the sentence according to the language model, and $p(X|S)$ is the conditional probability of observing the acoustic utterance given the sentence (encoded in the HMMs of the recogniser).

In general, the language model in a recogniser consists of all the language information to which it has access in order to help constrain the recognition, by making certain word strings less likely than others, or indeed impossible. This language information may be the same information as used in the 'meaning extraction' component of the system (see section 4.2). For example, when the grammar in the meaning extraction component is compiled down to a finite state network (FSN) for use in the recogniser. Figure 6 gives an example of a recogniser FSN. More commonly, the

language information in the recogniser is represented by a statistical model, possibly derived from the same corpus data as the language information in the ‘meaning extraction’ component, but generally only employing recent context (one or two words). This type of model is known as an n-gram model. Such models are easy to integrate with the standard acoustic decoding architecture in speech recognisers, but ignore some of the available language information which might improve accuracy.

N-gram models can be word n-grams, class n-grams or a hybrid of the two. n is typically 2 or 3, and these models are referred to as bigrams and trigrams respectively. A word bigram model gives the probability of all possible next words on the basis of the current word only, i.e. it is of the form $p(w_2|w_1)$. A trigram model is based on two words of context, $p(w_3|w_1w_2)$. The probability of a sentence such as ‘delete that one’ is obtained by multiplying the probabilities for each word, given its predecessor(s), e.g. for a bigram model:

$$p(\text{Sent}) = p(\text{delete}|\text{start}) * p(\text{that}|\text{delete}) * p(\text{one}|\text{that}) * p(\text{end}|\text{one})$$

A class n-gram model is one in which the words are grouped into classes before computing the n-gram statistics. These classes may be syntactic categories such as nouns or verbs, hand-crafted semantic categories (e.g. all days of the week might be grouped together, all names of people, etc), or automatically learnt classes. The probability of a word w_3 , given two previous classes, c_1, c_2 , is given by:

$$p(w_3|c_1 c_2) = p(c_3|c_1 c_2) * p(w_3|c_3)$$

The advantage of a class n-gram model is that there are fewer parameters to estimate. The disadvantage is that a word is predicted only on the basis of the class history, so some distinctions which may be present in the training data are lost; for example, ‘Monday morning’ may be more likely than ‘Monday afternoon’, whereas for all the other days of the week, ‘DAY morning’ and ‘DAY afternoon’ may be equally likely.

An FSN gives all possible word sequences in the language model, and it may have probabilities attached to the transitions if reliable statistics for these can be obtained. For example, a fragment of a FSN LM is shown in Fig 6.

Both n-gram and FSNs give a probability for the next word w_n given a word history w_1, \dots, w_{n-1} . In the case of n-grams, the history is only one or two words, as stated above

(e.g. $i = n - 2$ for a trigram model). In the case of FSNs, the history extends back to the start of the sentence (i.e. $i = 1$).

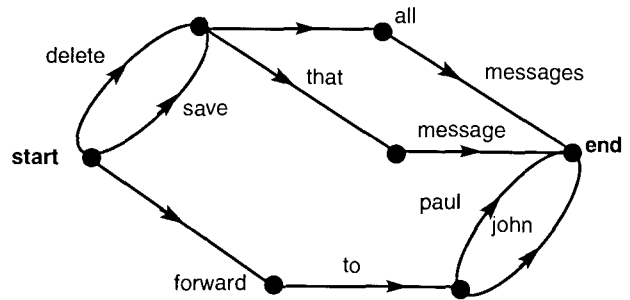


Fig 6 A recogniser finite state network.

4.1.2 Perplexity

In some cases, the constraints imposed by the language model are so great that the job of acoustic decoding becomes much easier than one would have imagined from the size of the vocabulary. This is because the LM is effectively ruling out many possible strings from the search space in which the acoustic decoder is operating. It is possible to calculate a figure called the perplexity associated with a given LM, which is a measure of how difficult the acoustic matching task for the recogniser is. Usually, the lower the perplexity, the higher the recognition accuracy. Perplexity can be roughly defined as the average branching factor, i.e. the average number of words which are allowed to follow a given word. Perplexity may either be calculated intrinsic to the LM (for an FSN, perplexity is literally the average branching factor of the FSN), or with reference to a test corpus, which is the way one usually calculates the perplexity of an n-gram LM. In the latter case, perplexity is defined as the reciprocal of the geometric mean of the probability of the next word, for all words in the test corpus. Intuitively, perplexity is low if the language model tends to know what is coming next, which means that the next word probabilities are relatively high. The formula for test set perplexity is:

$$PP = \left[\prod_i P(w_i) \right]^{-1/N}$$

where $P(w_i)$ is the probability of the i th word, N is the number of words in the test set, and the index i runs over each word in the test set.

Typical values of perplexity may range from about 40 to several hundred in current research systems.

4.1.3 Advantages and disadvantages of FSNs and n-gram LMs

An advantage of FSNs is that they have a fairly low perplexity, i.e. the number of legal following words is usually a small subset of the entire vocabulary. They can

also be constructed manually, before a proper training corpus for the domain exists.

One disadvantage of FSN language models is that they can easily be over-restrictive, ruling out many perfectly valid input strings, simply because of the impossibility of anticipating all the different things users will want to say. This disadvantage can be mitigated by making the FSN more flexible, with liberal use of optional phrases (as in BusCat), or going some way to turning it into an unconstrained phrase network, where any phrase (a few words) can follow any phrase. This of course has the effect of increasing the perplexity, and at a certain point it becomes preferable to move to an n-gram LM. A second disadvantage of FSN language models is that as one moves towards a broader, more habitable user language, they very quickly become large, making the recognition time unacceptably long. The FSNs themselves become cumbersome and difficult to maintain.

N-gram language models, on the other hand, usually have a higher perplexity than FSNs, because all words may be legal followers of the current word, albeit many of them having low probabilities.

The disadvantage of n-gram LMs is that they are not as constraining as FSN LMs, which means that if the acoustic match accuracy is not good enough, the output of the recogniser may be so poor that the meaning extraction component has little chance of interpreting it, however robust its algorithm.

4.1.4 Loading the LM into the recogniser

The LM(s) may be loaded into the recogniser:

- once at the start of an application,
- repeatedly, according to where the user is in the dialogue immediately prior to recognition,
- during a recognition by the recogniser.

These three possibilities represent increasing degrees of sophistication. In the first case, there is just one LM for the whole application. In the second case, use is made of the fact that at different points in the dialogue the relative probabilities of different words will change greatly. For example, if the system asks: 'Which day do you wish to travel on?', there is higher probability than normal that the user's answer will contain a day of the week. Similarly, if it has already been established that the user wishes to travel from Ipswich, there is a higher probability than normal that the destination will be somewhere in the Anglia region.

In the third case, the LM is changing 'on the fly', during the decoding of the speech utterance. For example, if the first words in the sentence are 'I want to go from Ipswich to..', then the LM can increase the probabilities of stations which have direct trains from Ipswich before recognising the rest of the sentence.

The LM(s) passed to the recogniser may be complete models or modifications to the current model. A modification to a current model might consist of:

- a list of words to be added or deleted,
- modified probabilities for words which are already within the language model.

4.1.5 Output from the recogniser

If the job of a speech recogniser is to convert a speech utterance into a string of text, then, when someone says: 'What time does the flight leave?' it is hoped that the recogniser might come up with the string: 'What time does the flight leave'. In practice the recogniser's best guess may not be correct, but it may be able to give a number of scored alternative possibilities which the analysis module might use if the top choice does not make sense, for example:

- 1 'what time does the white leaf' 1245.6
- 2 'what time does the flight leave' 1250.1
- 3 'what time does a flight leave' 1252.3
- 4 'what time did the flight leave' 1270.1
- 5 'what time did a flight leave' 1272.3

The analysis module may then use its own rules to re-rank the list of possibilities or extract those portions that carry useful information (perhaps taking account of the recogniser scores or perhaps ignoring them).

There are, however, significant drawbacks in representing the recogniser output as a ranked list of word strings. The list can become very long indeed, containing large numbers of sentences that only differ by one or two words.

A much more compact representation of the same information can be achieved through the use of directed graphs (see Fig 7).

A third option that is sometimes used is the word lattice (see Fig 8). This consists of a lattice with entries, each of which specifies a possible word, its start time and its end time (but not which other words it follows or precedes). The benefit of the word lattice is also its weakness. It can offer in a compact form for storing a very large number of candidate sentences — sentences can be generated by connecting any sequence of words such that one word starts where the previous one finishes. Unfortunately this allows one to generate sentences that the recogniser did not consider very likely, e.g. 'what time did a flight leaf'.

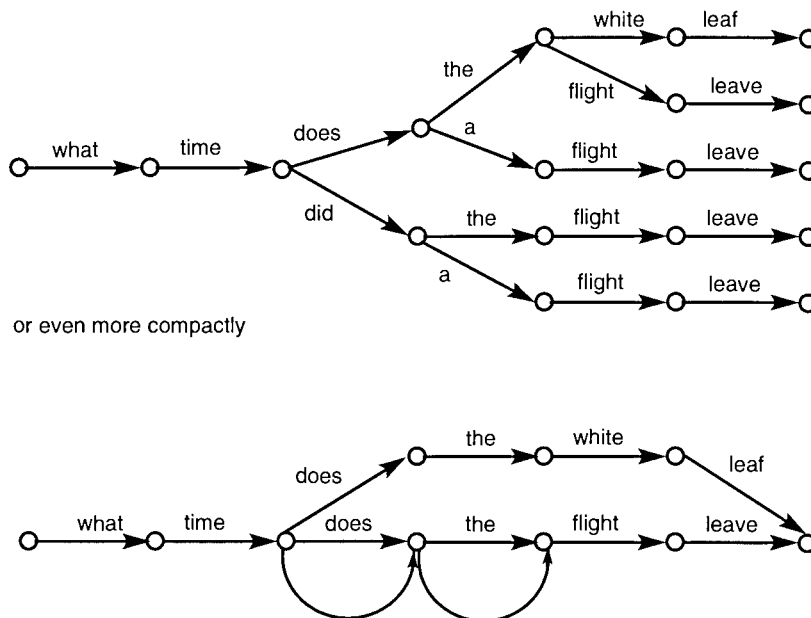


Fig 7 Directed graph for recogniser output.

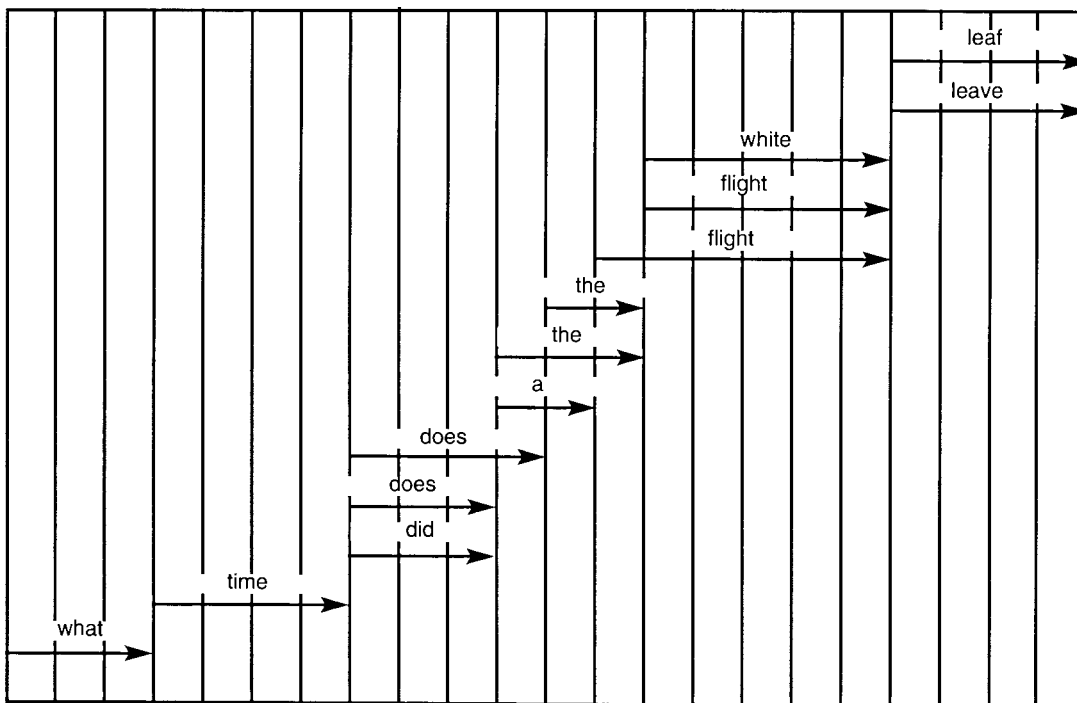


Fig 8 Word lattice for recogniser output.

4.2 Meaning extraction

4.2.1 Purpose of the meaning extraction component

The input to this component is a representation of the user's utterance that is the output from the speech recogniser. This representation may take one of several forms as described in the previous section. The meaning extraction component of the SLS converts the input to a representation of the meaning of the utterance, which is used to determine the next step in the dialogue.

The user's utterance will contain several pieces of information, some of which will need to be represented in the output from the 'meaning analysis' component. One piece is the type of utterance that the user made, e.g. an instruction, a statement or a particular kind of question. Another concerns the expectations present in the user's utterance. These are often expectations about the result of a query and are not usually intended to be used as constraints on the query itself (such as the use of a plural form to indicate that the user is expecting more than one item to satisfy a request). A third piece of information consists of

the entities referred to in the user's utterance, and the relationship between them. These will form the basis of the constraints on the database query. The purpose of the 'meaning extraction' component is to make some of this information explicit so that it may be more easily processed by subsequent components.

Although the role of this component can be described simply enough, the task is daunting. There are at least two main difficulties that need to be overcome during meaning extraction in SLS:

- ambiguity,
- ill-formed input.

Ambiguity [6] may be present in words as lexical ambiguity (e.g. 'saw' can either be a noun or a verb) or sense ambiguity (e.g. 'bank' may be a place in which to store money, or from which to fish), or it can be present in the relationship between phrases as structural ambiguity. To take the ubiquitous example:

'The man saw the boy with the telescope in the park.'

This sentence is structurally ambiguous in what was seen, the instrument of seeing, and the location of the person seeing. Often, this type of ambiguity cannot be resolved by the meaning extraction component, and has to be maintained in the meaning representation, to be deciphered by later components.

Ambiguity is inherent in all natural language, while ill-formedness can be caused by misrecognition in the speech recogniser, or by the user making an ungrammatical utterance (as defined by the system). Speech recognisers are becoming increasingly competent, but they still mishear quite often. This is particularly a problem with recognisers which use an n-gram language model which are less restricted in the ordering of the words they can recognise. For example, in a 'hotel enquiry' domain [7], the parser was faced with input such as:

'do the noise outsiders use your lake'

when the user actually said:

'there is a noise outside that keeps me awake'

It is doubtful that any meaning extraction component would be able to cope with such a discrepancy between the actual utterance and the recognised one, but there are mechanisms by which simpler ill-formed utterances can be analysed [8]. These will be explored in more detail below, but essentially they attempt to pick out those bits which can be analysed, and put them together in the best way possible.

If meaning extraction is viewed as performing a translation between an input text and its meaning, then, quite apart from the two particular problems described above, the difficulty of defining the translation is faced. This is a many-to-many mapping, since many English sentences can have essentially the same meaning (in the sense that they are true in the same situations) and some sentences can have more than one meaning. For instance, the following two sentences, to all intents and purposes, mean the same, but have a different structure:

'Romeo loves Juliet' and 'Juliet is loved by Romeo'

Assuming that all of the above problems can be overcome, a suitable representation for the output meaning must be chosen. The issues to be considered are:

- suitability for successive stages of processing — whether the representation encodes all the information necessary for further processing,
- explicitness versus conciseness — whether one representation can encode all unresolved ambiguity, while maintaining a clear meaning,
- extendibility — whether the representation will only cope with the particular sentences chosen for development.

The representation generally depends on the technology used, but various logics [9] are popular forms of representation. This is because they are well-understood, and fully-specified. Perhaps more importantly, a meaning represented in logic can be reasoned with, using the proof calculus of that logic. This is useful in database query systems [10]. The other main representation is to use a frame which encodes predefined meanings (such as the act of 'seeing') [11]. These frames will take arguments to fill in the details ('who', 'what', 'where', etc).

The rest of this section is devoted to a particular representation based on a logical representation. This representation, which is referred to as an extended logical form (ELF), has three fields corresponding to three types of information present in the user's utterance. An example ELF, representing the utterance 'read the message from John' is given in Table 3.

Table 3 Fields in an extended logical form representation.

Field name	Field content
Type	read(A),
Expectations	salient(A), singular(A)
Entities and relationships	message(A), named(B john), from (A,B)

The type field is used to indicate the type of the utterance, such as imperative, indicative, question, among others.

The expectations field contains a list of constraints that are the user's assumptions. In the example above, the user is presupposing that there is only one message from someone called John, therefore 'singular(A)' will be an item on the list of expectations. The constraint, 'salient(A)', is used to indicate that the user has assumed that there is some specific (usually recently mentioned) message that is the object of the instruction. This contrasts with 'read a message from John' where there would not be a 'salient(A)' constraint on the list.

The entities and relationships field of the ELF contains a list of conjunctive constraints which express the entities referred to in the user's utterance and the relationships between them.

The frame representation and the logical representation are indicative of the separation between different meaning extraction technologies. The logical form of representation tends to be used by grammar-based parsers, while the frame form tends to be used by caseframe parsers. These different types of parser will be described in the next two sections.

4.2.2 Grammar-based parsers

Traditionally, schoolchildren who are given the exercise of analysing English sentences according to a prescribed grammar of English have been doing parsing [12]. Parsing has the purpose of assigning a structure to an input utterance. This can be done by hand, but is now more often done using an automatic parser. More precisely, what has just been described would be a syntactic parser, performing a syntactic analysis. Here, the sentence is being analysed according to a syntactic grammar [13], where the relationship between nouns and verbs, for example, is made explicit. Syntactic analysis gives no representation of the meaning of the sentence. Indeed, the following two sentences would be assigned the same syntactic structure [14]:

'Bright red buses drive quickly' and 'Colourless green ideas sleep furiously'.

Of course, they clearly have a different meaning, and semantic analysis [15] is concerned with providing such a meaning, solely from the combination of the meaning of the individual words.

Grammar-based parsers take a linguistic grammar of, say, English, and assign a structure to the sentence based upon that grammar. Two stages are necessary — first, assign each word a part of speech (POS), such as noun, and, secondly, from these POS apply the grammar rules.

So, given the phrase:

'the man'

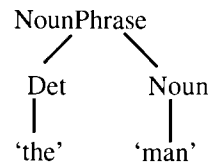
the lexicon which contains the following rewrite rules:

Det	→	'the'
Noun	→	'man'

and the grammar rule:

NounPhrase	→	Det	Noun
------------	---	-----	------

the parser would build the syntactic structure:



Syntactic analysis is not sufficient for meaning extraction as mentioned previously, and a semantic analysis stage is needed. This can either take place after the syntactic analysis or concurrently with the syntactic analysis. Indeed, semantics is often built up in parallel with the syntax, using the rule-rule hypothesis [15]. This basically states that syntactic and semantic rules should only exist in pairs, the semantic analysis being built by 'piggy-backing' on the syntactic analysis. This means that every part of the sentence contributes to its overall meaning, and that fine-grained syntax produces fine-grained semantics. Extremely complex grammars can be written [16], encapsulating fine points of language. These details may be important in some systems, and are thus represented in the final meaning representation.

The primary problem with grammar-based parsers, from the point of view of SLSs, is that they need to analyse the whole utterance before they can produce a representation. As has already been stated, there is quite a high likelihood of misrecognition, thereby producing input which is outside the scope of the grammar (ungrammatical input). One option is to constrain the speech recogniser using a finite-state network which mirrors the grammar of the meaning analysis component (section 4.1), thereby forcing the 'speech recognition' component to produce only those sentences which can be analysed using the grammar. An alternative is to dispense with the grammar-based approach altogether and use a caseframe parser.

4.2.3 Caseframe parsers

An alternative view of parsing has emerged as researchers have begun to use natural-language processing in practical applications. Caseframe parsers [17] do not use

an intermediate syntactic stage of processing, but attempt to go straight from the input representation to a representation of its meaning. This is done by assuming that phrases in the sentence have meaning, but only in the context of a predefined frame, or meaning template. An example of a frame is shown in Table 4. As many as possible of the slots of the frame are filled in, using simplistic syntactic analysis, from the phrases of the input sentence. This means that the phrases can appear in the middle of any amount of extraneous information, because only the particular phrases which are relevant to the frame are analysed. This approach is more robust than the grammar-based technique. For example, the following three sentences would be mapped on to the 'spilling' frame as shown.

'the waiter spilled the wine on the couple'

'It is the case, I believe, that earlier last week the waiter, what an appalling employee, spilled some of the wine which we use for only our best customers, on the couple visiting for the first time.'

'when waiter spilled the wine on the couple food'

Table 4 An example of a frame.

Spilling frame	
Actor	waiter
Patient	wine
Location	couple

However, there is a step in the above description that is obviously missing — how to pick the correct frame for the given utterance. This is usually done on the basis of the information contained in the sentence. One method of choosing the correct frame is to allow each piece of information within a sentence to vote for a particular frame. The frame used is the one which has the most votes for the given sentence.

A circular process can develop, however, where only the required information to fill the frame is analysed, but all the information needs to be analysed to pick the right frame. For this reason, caseframe parsers are generally only used in limited domains, where only a small number of frames are considered.

Caseframe parsers are not capable of the fine detail which a good grammar-parser can produce. This may be unnecessary in most applications, but could lead to confusion if the kind of input encountered really can only be distinguished using detailed analysis.

4.2.4 A hybrid approach

If grammar-based parsers are good at producing detailed analyses but poor at handling ill-formed input, while caseframe parsers are good with ill-formed input but unable to produce highly detailed analyses, then a combination of the two would seem ideal. Stallard and Bobrow [18] propose a two-stage process, combining a grammar-based parser followed by a caseframe parser, using a chart [19] as the intermediate representation.

Charts are used by a particular kind of grammar-based parser called a chart parser (this is something of a misnomer, since it is parsing words, not charts). This kind of parser produces all possible analyses of the text, storing intermediate results in a chart. This means that if there is no known analysis for the whole of the input, then there is the potential to look in the chart for incomplete analyses, which will represent those parts of the sentence which can be analysed by the grammar-based parser.

A nice way of combining these fragments is to build them into a frame. A caseframe parser can then be used to extract the meaning of the input utterance, not from the words, but from the chart produced in the first stage. This frame will then provide a meaning representation with as much detail as the grammar will allow, thus improving on the standard performance of caseframe parsers.

4.3 Dialogue manager

The dialogue manager (DM) is the central controller for an SLS. On receiving input from the speech recogniser, the DM manages and co-ordinates calls to the rest of the components in the system (see Fig 2).

The DM knows about the generic structure of conversations. It uses a conversational modeller to build up a dynamic model of the conversation between a user and the system as it progresses. It models conversational turns from both participants matching questions with their answers, instructions with acknowledgements, and so on. The syntactic and semantic analysis of individual utterances (or sentences) produced by the 'meaning extraction' component is used as a base and the DM goes beyond these single-sentence analyses to produce a dialogue model, which could be thought of as a grammar for the dialogue.

4.3.1 System-driven, user-driven, and mixed-initiative dialogues

Conversations with existing IVR dialogue systems, such as telephone banking systems, can be frustrating for callers because the conversation must always follow a strict pattern. A caller cannot say anything he or she wants, but is forced to give the correct reply at the appropriate time,

otherwise the interaction cannot proceed. In this kind of dialogue, the system takes all the initiative and callers are not allowed to ask questions, or say anything which is not specifically asked for. In other words, the dialogue is entirely system-driven.

At the other end of the scale are user-driven systems where the user takes all the initiative. The interface to the BT Business Catalogue (see section 3.1), operates in this way. The system accesses the World Wide Web (WWW) and waits until the user inputs a query.

Dialogue managers are now being developed that allow more natural conversations to take place between humans and machines. The intention is to produce a two-way flow of communication where the initiative can be taken either by a user, or by the system, and information may be given, asked for, and received by either party. Here is an example mixed-initiative conversation with an imaginary movie database query system of the future:

User: I'd like to see a film starring Meryl Streep.
 System: There are quite a few of them. Are you interested in thrillers, comedies or historical dramas?
 User: What about that one where she plays a Danish farmer.
 System: Do you mean 'Out of Africa'?
 User: Yes, that's the one.

The user instructs the system to find it a film starring Meryl Streep (user-initiative), but the system finds there is more than one and decides to ask the user what category of film (system-initiative). The user chooses not to answer this question directly, e.g. 'comedies', but gives instead some more information about a particular film (user-initiative). Now the system has enough information to pinpoint the film and the user confirms the system is correct.

4.3.2 Ellipsis and anaphoric reference resolution

Speakers use words or phrases to refer to things in the real world. Anaphoric references are a specific type of referring expression and are used when a speaker refers back to something mentioned earlier, e.g. 'he', 'him', 'her', 'that one', 'that way', 'the house' can all be anaphoric reference expressions:

User: Are there any messages from Peter?
 System: You have two messages from Peter Wyard.
 User: Read his second one.

In the above exchange, the user's second utterance contains two anaphoric references: 'his' and 'second one'. Assuming a grammar-based parser, it will represent 'Read his second one' as an ELF (see section 4.2). The ELF below shows that there is something salient and singular (A) which

is to be read, it is the second one, and it is from a masculine person of unknown name:

ELF:

```
read(A),
[salient(A),singular(A)],
[ord(A,2),named(B,C),gender(B,masculine),from(A,B)]
```

It is the reference resolution process which fills in missing information in the ELF. It does this by searching back through the preceding conversation to find something which can be read, and a masculine person. It is assumed that the last masculine person mentioned will be referred to as 'his', although this might not always be the case. The ELF output by the meaning extraction component is thus converted into a resolved extended logical form (RELF) where the missing information (message(A) and name(B,peter)) is filled in:

RELF:

```
read(A),
[salient(A),singular(A)],
[message(A),from(A,B),ord(A,2),name(B,peter),
gender(B,masculine)]
```

Ellipses occur when something is left out of an utterance which can be determined from what has gone before:

User: Do I have any messages from Peter?
 System: You have two messages from Peter Wyard.
 User: And David?

Here the user has left out: 'Do I have any messages from..' and has simply said: 'And David?' The 'meaning extraction' component will produce an extended logical form:

ELF:

```
A,
[],
[name(B,david)]
```

and the RELF will include the missing information which was found in the previous logical form:

RELF:

```
list(A),
[],
[message(A),from(A,B),name(B,david),
gender(B,masculine)]
```

4.3.3 Co-operative responses

Much of the ground work on the nature of conversation has been carried out by philosophers of language such as Grice. Grice devised a set of maxims, or co-operative principles, to which he considered people generally

conform when participating in a conversation [20]. Four of the most important maxims are of quality (be truthful and avoid statements for which you have too little evidence), quantity (be as informative as necessary, but do not give too much information), relation (do not give irrelevant information), and manner (do not present information in a way that is obscure, ambiguous or too lengthy). People tend to be co-operative in conversation for many reasons, for instance:

- to make the conversation flow more easily,
- to assist understanding,
- to pass information efficiently avoiding irrelevant information, false information, or too much detail,
- to promote good social relations.

If the maxims were ignored, then conversation would become very difficult. Of course there are many examples where the maxims are deliberately broken for the sake of humour, pathos or sarcasm. At present, these are outside the scope of this work and, unfortunately, the DMs do not have a sense of humour!

The DM tries to be as co-operative as it can in responding to the user — it tries to obey at least the maxims of quality, quantity and manner. It does this by being as brief in its responses as it can, by ensuring the database is accurate and up-to-date, and by supplying helpful information. The maxim of relation can be harder to satisfy in that the DM cannot always be sure that the information it is giving is maximally relevant to the user. For instance, if the database query finds nothing, then rather than just saying 'none found', the DM attempts to give some information which the user might find helpful. For instance:

User: Do I have any e-mails from Anna about aardvarks?
 System: Unfortunately you have no matching message,
 however, you have one message about aardvarks.
 User: OK, but do I have any e-mails at all from Anna?

Here there are no e-mails from Anna about aardvarks, but the system finds a message which is not from Anna but is about aardvarks. The DM does this by generalising the query it is making to the database. In order to do so it must decide which parts of the user's specification are the most important. Is it most important that the e-mail is from Anna or is it most important that the message is about aardvarks? Or are the two equally important? Here it wrongly assumes that the user is primarily interested in aardvarks. Our DM at present works from a priority list compiled from intuition about what might be most important to a user. The whole area of co-operation is one which we intend to investigate further in the future.

When the database query is generalised with a successful result, the DM produces another logical form, the co-operative extended logical form (CoopELF). For the above example, the RELF and CoopELF are as follows:

RELF:

```
list(A),
[plural(A)]
[message(A),from(A,B),name(B,anna),
gender(B,feminine),
about(A,aardvarks)]
```

CoopELF:

```
list(A),
[plural(A)] [message(A),about(A,aardvarks)]
```

Both logical forms are then sent to the response module (see section 4.5).

4.3.4 Error recovery

There are several reasons why a system may not understand a user's request — speech recognition errors, meaning extraction errors, and conversational modelling errors.

Unfortunately speech recognisers are not 100% accurate and a user does not always use phrases the computer can understand. It can be frustrating for a user if the system simply keeps repeating 'Sorry I didn't understand that'. Sometimes asking a user to repeat an utterance will result in a successful recognition, if, for example, a quiet utterance or background noise resulted in a recognition error. However, sometimes the user will need to rephrase the question, or speak later, or earlier, etc. It is necessary to have an error recovery dialogue which helps the user to try different strategies when difficulties occur, and which seems helpful rather than repetitive.

Sometimes conversational modelling errors occur; for instance, when a user says 'Read her e-mail' and the DM has no record of a female person being mentioned previously, the DM must then ask the user an appropriate question, e.g. 'Who do you mean?'

4.4 Database query

When the DM has prepared the query, it will be passed to the database query component. The database query component's purpose is to convert the query from the DM into one or more queries which can be used to find the required information from within the database. Having established the queries, the database query component then extracts the actual information from the database.

This is not as straightforward as it sounds and may involve a number of steps.

- Using general and domain-specific knowledge to attempt to satisfy the query — an example of this is the use of class taxonomies. If a user asked about ‘reptiles’, but the database knew about ‘snakes’, then the database query component could look down a reptiles taxonomy to realise that there is a link between query and data. Similarly, a user asking about ‘turtles’ from a database of ‘terrapins’ could have the query satisfied by the database query component if it looked sideways in a reptiles taxonomy.
- Optimising the query — by carefully arranging the order of execution of subqueries with a query, the speed of getting a result can be improved dramatically. To do this, the database query component will need to rank the severity of each of the constraints in the query. The query will execute optimally when the most severe constraints are applied first.

Of course, neither of these operations ensure that there will be a result from the query, i.e. there may be no answer to the query. Therefore the database query component will pass as much information to the dialogue manager as possible about a query which failed, so that the DM can decide what to do next.

Ultimately, the database querying module provides a means of separating the actual database query (in SQL, for example) from the internal representation in the DM. This should mean, in theory, that the dialogue manager need not be aware of the database used, and that using a different database management system only involves changing the internal mapping used by the database query component.

4.5 Response generation

In spoken language systems, responses are normally thought of as being provided using speech. However, in the case of multimodal systems, the response can also be in a graphical or pictorial form. This section is therefore divided into three parts — the generation of text, the synthesis of speech, and the generation of graphics.

4.5.1 Text generation

Text generation is the task of putting into words the information that is to be sent to the user. It can be as trivial or as complex a process as is appropriate for any particular SLS.

At the trivial end of the scale, text generation can be avoided altogether by using canned-text sentences. These work well if the application is very simple and there are

only a very limited number of things the system will need to say.

The next step up from this is to have carrier or template sentences where relevant information can be slotted. This allows a little more scope for personalising the messages to the user and has been used successfully in ELIZA [21], the famous artificial intelligence program which attempts to fool people into believing they are communicating with a psychologist. ELIZA recognises and scores certain patterns in the user’s input and uses the highest scoring pattern as a filler in its template output, transposing personal pronouns and possessives such as ‘I’ for ‘you’, and ‘my’ for ‘your’. For instance:

User: I am worried about my mother.
ELIZA: Tell me more about your mother.

The template output sentence is ‘Tell me more about...’ and ELIZA has transposed ‘my’ for ‘your’ in the pattern ‘my mother’ to give the filler ‘your mother’.

The most flexible method is to generate the most appropriate responses as and when necessary. Just as grammars can be used for analysis, most can equally well be used for generation. However, the flexibility of a text generator depends on the size of the grammar and lexicon used. If they are very small, the effect is little better than using template sentences.

Text generation is the reverse process of parsing. A parser converts a string of text to some kind of knowledge representation, whereas a text generator converts a knowledge representation to a string of text. For the e-mail assistant SLS, the RELF which has been instantiated after a successful database query is used for generation. If the initial database query was unsuccessful, then the uninstantiated RELF and the CoopELF are both used for generation. Furthermore, the presuppositions part of the RELF is used to reply in an appropriate way depending on the assumptions made by the user. Thus if the user was expecting more than one e-mail as in ‘Read me my messages from Peter’ and only one exists, then the system can reply ‘There is only one, the message reads:...’

4.5.2 Graphics generation

Multimodal systems are enhanced by graphs and/or pictures incorporated into the text. A graph or picture can show at a glance what would often take many paragraphs to describe in words.

BusCat (see section 3.1) builds WWW pages on-the-fly in order to answer a user’s query. The WWW pages include pictures of BT products relevant to the user.

A consideration when (automatically) collating material to be shown on a computer screen is size of images and graphs. It is desirable to try to fit the information within the limits of the screen rather than forcing the user to scroll down to view the whole page. The interface to the BT Business Catalogue uses different sized images to attempt to achieve this.

4.5.3 Speech synthesis or text-to-speech

BT's SLS systems all use the Laureate speech synthesiser [2] whenever TTS is required. When a text for synthesis is being generated automatically by the response module, information about the structure of the text will already be known. This opens up the possibility of providing Laureate with details of what kind of utterance it is (declarative, interrogative, or imperative), where the pauses should be, where the strongest emphasis ought to be placed, and so on. This is an area for future research.

5. Futures

There is a considerable amount of work to be done in taking the demonstration systems described in section 3 and turning them into applications which can be used by customers. To a certain extent this is the normal process of downstreaming, addressing issues such as thorough coverage of the application domain, robustness of software in the hands of naive users, and a well-designed user interface. However, there is also further work to be done on the underlying technology to make continuous spoken-language systems which will be really effective. Three such technology areas are highlighted in this final section. These technologies are all aimed at improving the quality of the system by making it more user-friendly and more intelligent.

5.1 Speech recognition

SLSs of the kind described in this paper place great demands on the recognition component, and these demands will increase as the coverage of the system is extended, because the recogniser must still work with sufficient speed and accuracy with a larger vocabulary and higher perplexity of language. In addition to these fundamental requirements, there are some other features which would help improve performance.

- Spontaneous speech recognition — this involves a number of different areas, each of them trying to enable better recognition of speech from naive users in a real application situation, as opposed to experts running a controlled demonstration. Non-speech sounds, such as breath noises, hesitations, coughs, etc, must be accommodated, together with out-of-vocabulary words (because it will never be possible to

create a recogniser vocabulary which covers every word to be spoken by real users), and the disfluencies, hesitations and restarts of spoken language.

- Confidence measures — the recogniser should be able to give the dialogue manager an idea of how confident it is that it has recognised an input utterance correctly, and beyond this, its confidence in particular parts of the recognised string. For example, the recogniser might say that it has recognised 'which films are UNK by Clint Eastwood', in which it is highly confident about everything except the unknown word UNK.
- Speaker adaptation — in many cases it is anticipated that SLSs will be used regularly by the same individual, in which case they may be willing to train a speaker-adaptive system to their voice before using the system. If this is not possible, the dialogues with SLS are usually long enough to make speaker adaptation during the dialogue a useful proposition.
- LM adaptation — it is important that the recogniser can efficiently update its language model during the course of the dialogue, as explained in section 3.1.
- Prosodic information — in the context of a spoken dialogue, there is considerable prosodic information in the user's utterances (stress, intonation, etc) which could help the recogniser, but is currently ignored.

5.2 User-centred language

The ideal would be for users to be able to speak to an SLS in a language which is natural to them, although it is realised that people will be prepared to adapt to the system to a certain extent. To do this the way real users actually say things must be studied, and incorporated as far as possible into the recogniser and grammar. However, user-centred language also requires more sophisticated processing in analysing the input sentence and translating it to a suitable database query. To take a very ambitious example, a user might say: 'How can we cut the cost of getting bulky documents from Edinburgh to London?', which would ideally be translated into a database query about faxes and ISDNs. This kind of translation requires considerable knowledge bases in the system, both domain-specific and general world knowledge, and it also requires a powerful inference engine to make effective use of the knowledge bases. In the shorter term, such queries may be beyond our capabilities, but users will not expect to have to phrase everything in exactly the terms encoded in the database which contains the required information.

5.3 *More flexible and intelligent dialogues*

The dialogues discussed are already considerably more flexible than the traditional prompt and response dialogues. However, there are two areas where dialogues could be enhanced:

- greater use of mixed-initiative dialogues, where the system sometimes takes the initiative — for instance to give the user some background information it has inferred they may be unaware of, or to make a ‘sales pitch’ for a particular product or service, if, for example, the user has been spending a lot of time asking questions about it,
- more use of user and life-style profiles — these are knowledge bases which the system has built up about the particular user, and the category to which the user belongs, the dialogue manager then being able to use this information both to help it interpret user input, and to decide on suitable initiatives to take, as discussed above.

6. Conclusion

Spoken language systems are likely to be of great importance in the future development of computer technology and its adoption by large numbers of people in the information age. This is because language is one of our primary means of communicating with each other, and unless humans change fundamentally in a short time, the use of natural language will be an important element in facilitating communication between humans and computers.

This paper has discussed the components of spoken language systems in general terms, and described three systems which are under development at BT Laboratories. There are a variety of spoken language systems, from pure speech-in/speech-out systems to multimodal systems which aim to combine spoken language with other modalities, such as typed text and mouse clicks, in order to achieve the most user-friendly interface possible.

As for the future of spoken language systems, the current state of the art is that they are on the verge of commercial deployment in some domains. However, it will be some time before we approach the scenarios envisaged in ‘2001: A Space Odyssey’ or ‘Star Trek’ where one can talk to a computer system in a completely natural way.

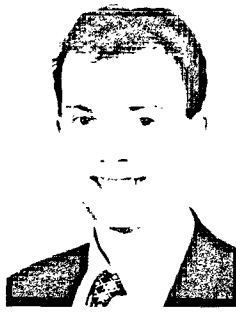
References

- 1 Scahill F et al: ‘Speech recognition — making it work for real’, *BT Technol J*, 14, No 1, pp 151—164 (January 1996).

- 2 Edgington M E et al: ‘Overview of current text-to-speech techniques: Parts I and II’, *BT Technol J*, 14, No 1, pp 68—99 (January 1996).
- 3 Bissell R A and Eales A: ‘The set-top box for interactive services’, *BT Technol J*, 13, No 4, pp 66—77 (October 1995).
- 4 <http://www.cm.cf.ac.uk.movies>
- 5 Power K J: ‘The listening telephone — automating speech recognition over the PSTN’, *BT Technol J*, 14, No 1, pp 112—126 (January 1996).
- 6 Gazdar G and Mellish C: ‘Natural language processing in Prolog’, Wokingham, England, Addison-Wesley, pp 169—174 (1989).
- 7 Rayner M and Wyard P: ‘Robust parsing of n-best speech hypothesis lists using a general grammar-based language model’, in 4th Eurospeech Conference on Speech Communication and Technology, EUROSPEECH ’95, pp 1793—1796 (1995).
- 8 Special Issue on ‘Ill-formed input’, *American Journal of Computational Linguistics* (1983).
- 9 Hodges W: ‘Logic’, Hamondsworth, England, Penguin Books (1977).
- 10 Warren D H D and Pereira F: ‘An efficient, easily adaptable system for interpreting natural language queries’, *American Journal of Computational Linguistics*, 8, No 3—4, pp 100—119 (1982).
- 11 Fillmore C: ‘The case for case’, in ‘Universals in Linguistic Theory’, New York, Holt, Rinehart and Winston, pp 1—90 (1968).
- 12 Beardon C, Lumsden D and Holmes G: ‘Natural language and computational linguistics’, Chichester, England, Ellis Horwood, pp 150—156 (1991).
- 13 Allen J: ‘Natural language understanding’, The Benjamin/Cummings Publishing Company, p 41 (1987).
- 14 Chomsky N: ‘Syntactic structures’, The Hague, Mouton Publishers, p 15, (1957).
- 15 Cann R: ‘Formal semantics’, Cambridge, England, Cambridge University Press, p 5 (1993).
- 16 Alshawi H (Ed): ‘The core language engine’, London, England, MIT Press (1992).
- 17 Carbonell J G and Hayes P J: ‘Robust parsing using multiple construction-specific strategies’, in Bolc L (Ed): ‘Natural language parsing systems’, Springer-Verlag, Berlin, pp 1—32 (1987).
- 18 Stallard D and Bobrow R: ‘Fragment processing in the DELPHI system’, *Proc of Speech and Natural Language Workshop, DARPA*, pp 305—310 (1992).
- 19 Earley J: ‘An efficient context-free parsing algorithm’, *Communications of the ACM*, 13, No 2, pp 94—102 (1970).
- 20 Grice H P: ‘Logic and conversation’, in Cole P and Morgan J L (Eds): ‘Syntax and Semantics’, 3, *SpeechActs*, pp 41—58, New York, Academic Press (1975).
- 21 Weizenbaum J: ‘ELIZA — A computer program for the study of natural language communication between man and machine’, *Communications of the ACM*, 9, No 1, pp 34—45 (1966).



Peter Wyard obtained an honours degree in theoretical physics from Cambridge University and an MSc in astronomy from the University of Sussex. After ten years teaching physics in India and the UK, he took an MSc in digital systems from Brunel University and joined BT Laboratories in 1988. He first worked on connectionist natural language processing. After a period as project leader for speech technology evaluation, he returned to the language group where he now leads the work on developing spoken language systems.



Ed Kaneen began work for BT in 1990 as a sponsored student, and joined the language group full-time in 1994, having gained an MEng in Computer Systems and Software Engineering from the University of York. He worked initially on prosody for text to speech, before returning to the work of his MEng, on the parsing of ungrammatical language. This has continued with the language group, focusing particularly on the interface with speech recognition.



Alison Simons joined BT in 1982 as a sponsored student. After receiving the degree of BSc in Computer Science from the University of Manchester in 1986, she joined BT Laboratories and began work in the speech synthesis research team.

She subsequently switched to speech recognition where she led a research team developing BT's first sub-word unit speech recognisers. She currently leads a team which is developing spoken language systems for accessing information sources and has recently completed the BT MSc in

Telecommunications Engineering.



Sandra Williams' early career was varied and included seismic survey analysis, customer relations work, and stained glass craftwork. She joined BT Laboratories in 1988 after graduating from Sussex University with a BA degree in Artificial Intelligence. She joined the natural language group to work on automatic text summarisation.

In 1992, she obtained an MPhil degree in Computer Speech and Language Processing from Cambridge University, which was sponsored by BT. Since 1992, she has been part of the language group in Systems Research. Her

specialist work interests include automatic text summarisation, and discourse and dialogue analysis. She is developing dialogue managers to handle ever more natural conversations between humans and computer.



Steve Appleby joined BT in 1983 to investigate methods for locating buried plant after gaining an honours degree in Physics with Musical Acoustics from the University of Surrey. After working in the copper access group on various problems related to copper network maintenance he joined the Systems Research Division to work on a wide variety of topics including fractal population modelling and mobile agents.

Currently, he works in the natural language group on semantic aspects of natural language processing and is completing a PhD

in fractal population modelling.



Keith Preston gained a BSc(Hons) in Applied Physics and Electronics from the University of Durham in 1978. After joining BT he was involved in pioneering work in optical communications, and has many publications and patents in this area. He was also responsible for the design and prototyping of a commercial range of advanced laser transmitter products. In 1992 he moved into the area of advanced software and now heads the natural language group at BT Laboratories.

Presenting
MAGIC CAP™

A Guide to General Magic's
Revolutionary Communicator Software



Barbara Knaster



Presenting
MAGIC CAP™

A Guide to General Magic's
Revolutionary Communicator Software



Barbara Knaster

tt



Addison-Wesley Publishing Company
Reading, Massachusetts • Menlo Park, California
New York • Don Mills, Ontario • Wokingham, England
Amsterdam • Bonn • Sydney • Singapore • Tokyo
Madrid • San Juan • Paris • Seoul • Milan
Mexico City • Taipei

distinguish their products are claimed as trademarks. Where those designations appear in this book, and Addison-Wesley was aware of a trademark claim, the designations have been printed in initial capital letters or all capital letters.

The author and publishers have taken care in preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

Library of Congress Cataloging-in-Publication Data

Knaster, Barbara.

Presenting Magic Cap : a guide to General Magic's revolutionary communicator software / Barbara Knaster.

p. cm.

Includes index.

ISBN 0-201-40740-X

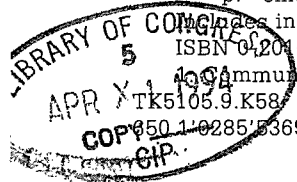
1. Communications software. 2. Magic cap. I. Title.

TK5105.9.K58 1994

950.1:0285'5369—dc20

93-46018

CIP



TK 5105
.9
.K58
1994

Copyright © 1994 by Barbara Knaster

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America. Published simultaneously in Canada.

Sponsoring Editor: Martha Steffen

Project Manager: Joanne Clapp Fullagar

Production Coordinators: Vicki Hochstedler • Gail McDonald
Jordan

Cover design: Grand Design/Boston

Set in 11 point Serifa Light by Total Concept Associates

1 2 3 4 5 6 7 8 9-ARM-9897969594

First printing, February 1994

Addison-Wesley books are available for bulk purchases by corporations, institutions, and other organizations. For more information please contact the Corporate, Government and Special Sales Department at (800) 238-9682.

Contents

	Preface: About Magic Cap	v
	Acknowledgments	xvi
Chapter 1	Getting Started	1
Chapter 2	Electronic Mail	19
Chapter 3	General Features	59
Chapter 4	Datebook	77
Chapter 5	Name Cards	113
Chapter 6	Phone	141
Chapter 7	Notebook	159
Chapter 8	File Cabinet	185
Chapter 9	Other Features	211
Chapter 10	Construction	233
	Index	249

*For my boys,
Scott and Jess*

Preface



About Magic Cap

Why Me?

I don't love technology. I don't hate it either, but I don't welcome it into my life unless I can figure out how it will make me happier or more efficient. I think of this as being very practical, but since I live in Silicon Valley, some people consider me almost primitive for having this attitude.

When I first heard about General Magic's dream of creating a personal communicator, I was impressed by the team of programming and user interface legends who had been assembled to build this portable box that kept you in touch all the time. At the same time, I wondered why most people would want one of these things. These communicators were going to let you send messages to anyone from anywhere, and they were going to be as easy to use as a telephone. Well, I already had a telephone, and I sometimes found it more intrusive than indispensable.

A personal communicator would also be an electronic datebook and notepad. There were plenty of electronic organizers already; would this one be just another expensive toy? I seemed to have no trouble at all being skeptical, wondering whether this idea of "reinventing telephony" would have much of an impact in the real world outside Silicon Valley, even with the impressive track record of the wizards at General Magic.

v

Of course, the list of companies that helped get General Magic rolling (Sony, Motorola, and Apple) and the others who joined along the way (AT&T, Philips, and Matsushita) added a lot to this tiny start-up company's credibility. Eventually, I became a guinea pig in official tests of the software in General Magic's lab and unofficial tests at home, courtesy of my husband, who joined the team as employee number 14. As the communicator shaped up, I started to see for the first time how it might fit into my life.

I also became intrigued by the powerful culture of this unique company: dedicated workaholics collaborated with engineers who had families, sharing an almost fanatical need to make magic. Watching this culture work to make practical tools helped convert my skepticism into enthusiasm. General Magic's dreams developed into two software platforms: Magic Cap and Telescript.

Origins of General Magic

The original idea for personal communicators sprouted in Apple's Advanced Technology Group. A research group led by Marc Porat observed three central trends that showed how people spent their work and personal time. First was the need to communicate easily and conveniently—with co-workers in the same office, with a spouse running errands, with clients in other parts of the world. Second, people increasingly require information on demand—stock quotes, movie schedules, how the home team fared. The third trend was remembering all this information—who to meet, where to be, when to be there. Porat also proposed the theory that people don't always clearly separate the personal and business parts of their day, which is the root of what he calls *whole person thinking*.

Creating a product for this model was the challenge. The code name Paradigm was chosen for the project (you can't do anything in Silicon Valley without a code name). When Apple realized that it couldn't devote enough resources to the Paradigm project, General Magic was created in 1990. The founders of the company were Marc Porat, the visionary who nurtured the ideas and put together the astonishing alliance of consumer electronics manufacturers and communications giants; Bill Atkinson, the legendary programmer and user interface designer who created HyperCard and the original graphics software in the Macintosh; and Andy Hertzfeld, the software wizard who programmed much of the original Macintosh. In subsequent years, many talented programmers and designers who worked on other successful products joined General Magic to form a world-class engineering group.

It's the Communication

The whole idea of Magic Cap is communication. A personal organizer is cool, but it's been done. An electronic datebook and address book combination is really useful, but it's not necessarily more special than its paper counterpart. Electronic mail and information services aren't just trendy, last-minute additions to Magic Cap; from the beginning of the project, everything in Magic Cap was designed around the idea of enabling people to communicate powerfully and easily.

When you turn on a Magic Cap communicator, you see a picture of a desk, laid out to simulate the way people work (see Figure P-1). There's a telephone, a datebook, a file for names and addresses, a notebook for writing and drawing. But right in the middle of the desk is a postcard and pencil, a subtle reminder of what Magic Cap is all

about. The in box and out box are also located in the center of the communicator's screen, their prominent positions drawing your eyes and attention.

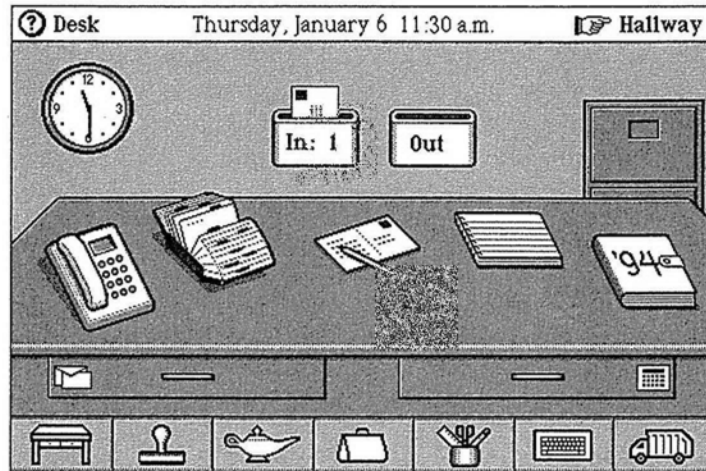


FIGURE P-1. The Magic Cap desk

Placing the postcard and other kinds of stationery in the desk drawer, Magic Cap encourages you to use electronic mail for every purpose: business letters to prospective clients, personal notes to your mom, invitations to meetings, birthday greetings to a friend in another state, and electronic orders for goods and services. You can use Magic Cap's electronic communication for many of the things you're used to doing with your telephone or postal service.

Every Magic Cap device is truly a communicator and not a message pad, an organizer, or a personal digital assistant. Every Magic Cap communicator comes ready to connect to a telephone line: there's a modem built in. Some also include two-way radios for wireless communi-

cation, but at the least, every Magic Cap communicator is only a standard telephone jack away from connecting you to the world of electronic networks.

Your communicator does a great job of replacing your address book, but it has another purpose for the names and addresses it knows: It can help you send messages to the people and companies it lists. Magic Cap also simulates your appointment book; and you can use it to contact the participants in your meetings to invite them to attend. Because Magic Cap's smart communication features connect you to the world outside, you can actually tell a news service what kinds of stories you're interested in, and those stories will be culled automatically and delivered for you to read at your convenience. Information providers may someday offer news, up-to-the-minute sports scores, movie and television schedules, road directions for many major cities, and lots of other services that somebody somewhere is dreaming up right now. It's like a home shopping network in the palm of your hand.

It's for Everybody

The engineers and interface designers at General Magic worked hard to make Magic Cap very easy to use. They wanted to make a consumer electronics product, not a computer, and so it had to be far easier to use than the simplest personal computer. There are lots of things that personal computers do very well, including keeping track of big chunks of data, publishing documents, and crunching numbers. Even though Magic Cap can handle some of those things, it was born to communicate.

Sending messages using AT&T's new PersonaLink service or conventional electronic mail and fax is intuitively simple with a communicator. Receiving mail is

also easy—a matter of simply connecting and collecting. Because Magic Cap communicators are electronics products designed for general consumer use, they may not be as powerful as many personal computers, but they are certainly more friendly.

Magic Cap's designers conducted scores of user tests to refine the way it works. Features that were confusing to novice users were tweaked or simplified. For example, at one time users could move objects around on the desk just by sliding them. This was disconcerting to people who didn't have computer experience—they just wanted to open the datebook, but it kept scooting away instead. The designers made a trade-off: Users had to enter an explicit mode to be able to slide the permanent features on their desks, so savvy users could redecorate, but beginners wouldn't be startled by something happening unexpectedly.

Computer veterans and other knowledgeable users had suggestions for ways to increase power, and many of those suggestions were added to Magic Cap, but never at the expense of friendliness. Magic Cap's inventors included many features that make it easier for power users, but those features are designed to stay out of the way of beginners.

Magic Cap's navigation system is an example of the designers' focus on simplicity. It lets users work with a desk, rooms in a hallway, and a downtown street. Novice users inevitably love Magic Cap's navigation, and pundits often criticize its appearance as too simplistic and playful. Many experts also criticized the friendly, playful Macintosh interface when it appeared in 1984. Now, of course, most personal computer users work with windows, icons, menus, and other elements that were disparaged on the first Macintosh. We'll have to wait and see what the eventual response is to Magic Cap's interface.

x

Designers of products that are supposed to be easy to use often talk about whether “your parents” could use it. Magic Cap may not be as easy as a telephone, but it’s designed so that most parents could quickly figure out how to use it to send a message to their children asking why they never write anymore.

Telescript Inside

Magic Cap integrates many tools that people use to communicate for work and play. At the core of this platform is Telescript, a communication-oriented programming language also developed by General Magic. There are several elements in the communication heart of Telescript. Foremost is its smart-messaging capability. As the foundation for sending and receiving electronic mail, Telescript actually turns each message into an agent, or independent program, that can carry personalized information with it.

Because each message is really a Telescript program, these “smart messages” can perform functions besides just expressing your words. If you use your communicator’s datebook to schedule a meeting with your colleague Tony, you can also automatically create an invitation for him to attend the meeting, send it to him, and then have the message complete a series of reactions based upon his response; he can use it to create and deliver his acceptance or regrets, and even schedule the meeting in his datebook.

Telescript messages travel in “smart envelopes,” which are Telescript programs that include a way to tell the message how to deliver itself. An electronic mail network based on Telescript can let you tell the message to wait in Tony’s mailbox until 5 P.M., and if he doesn’t pick it up by then, to fax it to him at home. The mailboxes that the

message passes through are also Telescript programs, meaning that they're also smart and can carry personal preferences.

Using a Telescript-based network, you can stamp your message to Tony as urgent. Meanwhile, because his mailbox is also a Telescript program, he has instructed it to let him know immediately when he receives an urgent message. A conventional electronic mail system may also have some of these "smart" features, but if they weren't built into the original engineering, it would be impossible for users to add them later. If a Telescript system needs to add features, users of the mail system can add and revise them.

As more Telescript-based systems are created, they'll help extend the power of Magic Cap. A smart network would provide a handy way to interact with a store that has an electronic location downtown. If you wanted to send flowers to your Aunt Dorothy, you could visit the flower store downtown, then send a message to the florist that you wanted a bouquet of flowers for \$35 to be delivered today in Kansas. Your order could automatically attach your name for billing and your aunt's name and address for delivery information. The flower shop's mailbox could have special rules set up for receiving such orders that would expedite having the flowers delivered in time for her birthday. So far, Magic Cap's engineers haven't figured out how to have the flowers themselves come through a communicator, but just wait.

Families of Products

There are several different models of Magic Cap communicators from different manufacturers, and each one provides ways to communicate. Some Magic Cap communicators need to be plugged into a phone line; others

xii

take advantage of wireless transmission via radio waves. There may be additional options like telephone handsets that plug into your communicator or cellular phones that can be added. Every one of these, though, has one thing in common: Magic Cap. Communication à la Magic Cap is the foundation; your distinctive model of communicator provides the access.

Design Goals

General Magic engineers worked together with their alliance partners in designing various models of communicators that had to meet important goals. Communicators have to be small enough to be carried around all the time and easy enough for people to figure them out without hours of study. Magic Cap's designers compensated for small screen size by making items look simple and easily touchable. Performing various tasks in Magic Cap is intuitive and easy: touch the screen to activate items on a desk, go into a hallway of rooms filled with other features, or go to a downtown street with buildings representing remote services.

The immaturity of touch-screen technology provides another design challenge. Screens of current models are often difficult to see, another reason that the desk items are spaced far apart and well defined. Because it's hard to touch an exact point, Magic Cap allows for an imprecise touch to act precisely.

Magic Cap uses an on-screen keyboard as its main source of input; its interface doesn't require handwriting recognition. Because an on-screen keyboard is unwieldy, Magic Cap includes a large set of features to speed up typing. These features include trying to automatically complete words in well-known categories (names, cities, states, and so on), automatically guessing whether to

shift the keyboard to uppercase, and cross-referencing information (for example, learning which cities match which ZIP codes). Magic Cap interprets handwriting as ink and doesn't try to translate it into text.

Magic Cap Is for Communication

Magic Cap is a software platform designed specifically for communication, as shown in this classic message that helped inspire the Magic Cap team. In the spring of 1990, Bill Atkinson received this electronic postcard from his young daughter, Laura, who used an early software prototype to convey her thoughts simply and creatively, and her dad was able to read and enjoy her message at his convenience (see Figure P-2). This is what Magic Cap does best: personal communication.

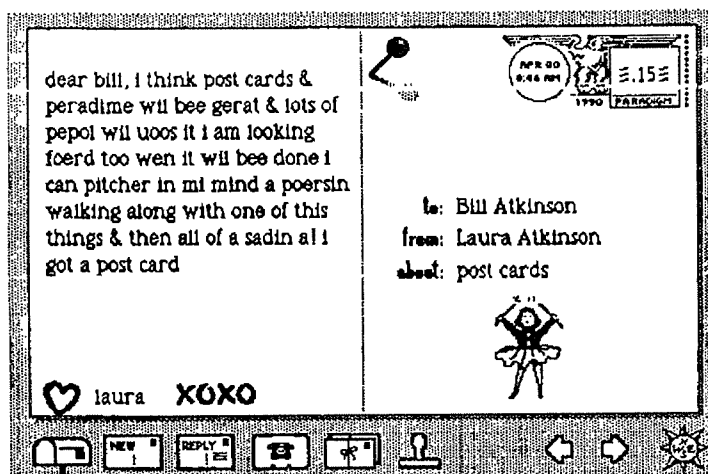


FIGURE P-2. "A! I got a postcard"

About This Book

By the time you read this book, you may already have a Magic Cap communicator, or not. You don't need to have a communicator for the book to be useful. If you have one, you can use this book as a complementary tutorial, and you might use the examples as a springboard for your own ideas. If you don't have a communicator yet, this book shows what Magic Cap can do and how you might be able to use it.

This book explains the concepts of Magic Cap and shows some practical examples of its use. Many of the scenarios are completely realistic and can be accomplished with the first Magic Cap communicators. You may find yourself using your communicator in exactly the same way, with only the names changed. Some other examples show how Magic Cap might develop over time, a kind of wishful thinking that could happen if communicators become popular. This book is pretty specific in stating what Magic Cap can do now, and what it might do in the future.

Magic Cap software was substantially done before this book was written, but some small details may have changed since then.

Acknowledgments

I didn't realize how many people were vitally important in publishing a book until I wrote one.

Everyone I worked with at Addison-Wesley was wonderful, even the people I never met who were responsible for the cover and art design. Martha Steffen was extremely helpful in explaining the publishing process, negotiating a breakneck schedule, suggesting ways to improve the manuscript, and calming down a nervous author. Every author should have an editor like Martha.

Joanne Clapp Fullagar managed all the production details so smoothly and completely from both coasts that the schedule seemed hectic but not impossible. I really appreciated her encouragement and suggestions.

Keith Wollman and Steve Stansel took a chance on publishing a book by an unproven writer about software still being written.

Tema Goodwin provided thorough and thoughtful copyediting that considerably improved the book.

Bill Fallon and Bob Garnet from AT&T reviewed parts of the manuscript, offering their suggestions and insight.

The folks at General Magic were encouraging and helpful. Curtis Sasaki and Jane Anderson helped get the project started, and Joanna Hoffman gave continuing support. Along with Curtis, Lynn Franklin, Susan Rayl, Terry Moody, Kevin Lynch, and John Sullivan also reviewed the manuscript to make sure I got things right.

xvi

David Hendler, a most literate writer, is also reviewing the book.

Bill Atkinson and Laura Atkinson gave me permission to reprint the wonderful postcard Laura sent to her dad. Bill also let me borrow his extensive collection of postcards sent and received in early 1990 to give me an excellent perspective on the evolution of Magic Cap.

The Magic Cap engineers offered their encouragement, and I thank them for answering all of my questions about how things worked (or didn't) and why they worked that way (or didn't). I am grateful to them for letting me share the magic.

My family, as always, was very supportive. Helen Schulman kept telling me that she always knew I would write a book someday, and I'm glad I could prove her right. Thanks, Mom.

Gene Schulman kept asking how the book was coming while he reminded me to take care of myself, and I know he is a very proud father right now. Thanks, Dad.

Louis and Jennifer Schulman came for a visit right in the middle of this frenzy. That weekend helped me keep my sanity.

Jess Knaster was very understanding about not getting to go anywhere for several weekends in a row because of "Mom's book." He waited as patiently as an eight-year-old can wait for his parents to take a break from work to play with him. He even let me use him in some examples in the book. Thanks, Jess. You're a great kid.

Scott Knaster was (and still is) my inspiration. He was the technical reviewer of this book, and he helped create

all the figures. He was also the source of the General Magic anecdotes. He micro-scheduled every page so this book could be finished in an amazingly short time, and he worked beside me on many late nights making this book better. He told me I could do it and then helped me actually do it. I'm so lucky to be married to my best friend.

Chapter 1

Getting Started

The First Time

You're probably interested in personal communicators because you've always been one of the first to get your hands on the newest technology—the industry calls you an *early adopter*, or a *heat-seeker*. Maybe your boss suggested that using one while you're traveling is good business, or you have an incredibly understanding spouse who bought you one for your birthday. On the other hand, maybe you haven't decided to take the plunge yet, but you want to know what it feels like to have one. Whatever the circumstance, you've joined the brave new world of personal communicators.

The first few minutes with a communicator are among the most important in your relationship, kind of like the experience between nervous job applicant and thorough interviewer. When you take it out of the box and start using it, you should feel good about the experience, not uncomfortable. You shouldn't feel overwhelmed by incomprehensible setup procedures or three different thick manuals, each of which says "read me first." General Magic and its alliance partners worked hard to make your beginning experiences pleasant, friendly, and reassuring. In this chapter, we'll go through the process of unpacking and setting up a new Magic Cap communicator.

1

Basics

Although Magic Cap communicators come in various models from several different manufacturers, they all have many features in common. Most important is that all Magic Cap communicators are operated by touching pictures of objects on the screen. You don't have to use lots of different gestures when touching the screen to make things happen. There are really just two actions you have to learn: touch and slide. To touch, just place your finger or stylus on an object, and then let go. To slide, touch any object and move it along the screen, as if you were sliding it aside. Everything in Magic Cap operates with those two actions.

To help you figure out what you're doing, Magic Cap creates a little world inside your communicator. This world is filled with familiar objects, such as a desk, a telephone, a datebook, an in box, and a clock. To learn to use Magic Cap, you start with what you already know about working with these and other familiar objects.

When you look at a Magic Cap communicator, you'll see that it comes with just one physical key, labeled *option*. If you hold down the option key while touching certain objects on the screen, you can make an alternate or advanced action take place. These optional movements are often used to take advantage of shortcuts for actions—they're never used for common or required functions.

Every Magic Cap communicator has a jack where you can plug in a telephone line. This is how you'll use your communicator to send and receive electronic mail, make phone calls, and send faxes. Some communicators also have built-in two-way data radios for sending and receiving information without having to connect a phone line.

Getting Started

Your first step should be installing the batteries in your communicator. Putting them in at the factory would drain some of their power during shipping and shelf time, so you get to have fresh batteries by installing them yourself. Every communicator has at least three sources of power: a main battery, a backup battery, and a wall adapter. Power is vitally important to your communicator—if it ever loses power completely, it will lose the information you entered! Magic Cap has an elaborate warning system to tell you when your main and backup batteries are running down.

Once you've installed the batteries, the next step is to turn on your communicator. The first images you'll see are the logo of the manufacturer and the Magic Cap rabbit-in-the-hat logo, and the provocative instruction to *Touch the screen to begin*. Your first action will be to teach the communicator about how hard your touch is and to fine-tune the screen's alignment. Magic Cap puts a bull's-eye target in the upper-left corner and asks you to touch it. When you touch it, the communicator's speaker sounds an approving *pop* and the target hops around to two other locations on the screen, calibrating your touch so that it will be more responsive to it.

While you're aligning the touch screen by tapping the targets, you're subtly experiencing three of the key elements of using Magic Cap. First, almost everything is accomplished by touching pictures you see on the screen. Second, when something changes its location on the screen, you'll usually see animation that makes it move rather than just having to figure out that it's gone from one place to another. This animation reinforces what you're seeing so that you're not surprised when the item appears in a new location.

Third, the targets make a popping sound when you touch them. In Magic Cap, most actions make sounds. As you get familiar with Magic Cap, these sounds will become reassuring and will help you confirm your actions. Of course, if you find the sounds annoying or you don't want to disturb people nearby, you can change them or turn them off completely.



Just a Touch. Magic Cap communicators come with a stylus, a sort of pen with no ink, but Magic Cap's hardware and software were designed to let you use your finger if you prefer. The stylus is required for only two functions: the alignment targets, since they need to be touched as precisely as possible to set the screen, and for handwriting, which is really tough to do well with your finger. You can do everything else with a stylus or your finger.

Magic Cap tries to be generous in deciding where you can touch things to activate them. Some items have an invisible halo around them so you can actually miss them by just a little when you use your fingertip to touch or slide. Some items require more precision, such as when you're typing on the keyboard; you can use your fingernail to get a better shot at them. As you use your communicator, you'll have a better idea of whether you want to use your finger or a stylus.

Magic Cap's manuals and information windows use two different words, *tap* and *touch*, to describe the action of placing your finger on an item on the screen and then removing your finger. Although *tap* is more appropriate for a button and *touch* is often used with other kinds of objects, the terms are completely interchangeable, and this book follows suit.

After you finish target practice, you get your first look at the desk and you see your first information window, as shown in Figure 1-1. The window suggests that you touch its *Getting Started* button to set up your communicator, but if you're not ready for that, you can touch the x in the upper-right corner of the window to close it and postpone the *Getting Started* stuff.



FIGURE 1-1. The desk with its information window open

You can always go back to *Getting Started* when you're ready by touching the circled question mark next to the word *Desk* in the upper-left corner of the screen, then tapping *Getting Started* in the window that appears. In fact, you can get information about any screen or window by touching that circled question mark. It's a good idea to run through *Getting Started* as the first thing you do with your new communicator.


There are three kinds of actions in the *Getting Started* process. First, there are vital setup tasks you must do to personalize your communicator before you can do almost

anything useful. The second kind are actions that you should do before proceeding but that aren't absolutely required. Third, there are instructive lessons that teach skills you might pick up on your own as you're using Magic Cap but that help cut down on any apprehension you might feel when you begin.

Starting the Lessons

There are two things you must do to personalize your communicator before getting much of anything done: You have to tell it who you are and where you are. *Getting Started* provides lessons that help you enter this important information. When you enter your name and location, it's important to recognize that these are not just practice examples; the information you give is saved and should be the real stuff.

What happens if you try to skip ahead and avoid entering your name or location before continuing? Many functions work fine whether they know your name or not: You can use the calculator, write in the notebook, or play a game without filling in your name. If you try to write a message, though, Magic Cap will gently remind you to enter your name first (it even says *please*).

 **Doing the Time Warp.** During Magic Cap's testing, many users thought that *Getting Started* was just a teaching tour of Magic Cap that let them practice doing things, not realizing that setup information entered there was very real. This led to lots of communicators being set up by people named Frank N. Furter and Bugs Bunny. Be sure to type the real information when you go through *Getting Started*.

If you touch *Getting Started*, instructions will lead you into the hallway, then down to the library where the *Getting Started* book waits for you. You'll be directed to tap the *Getting Started* book, as pictured in Figure 1-2, and you'll see the opening pages of the book.

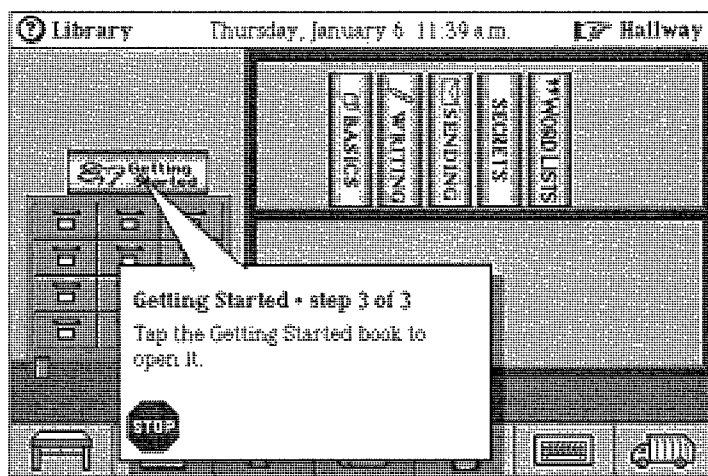


FIGURE 1-2. Just about to open the *Getting Started* book

When you start a lesson, you'll see the number of steps you'll need to follow to complete the lesson, and you'll know many you've done so far. The steps also tell you exactly what you're supposed to do and why, rather than just doing it for you. When a lesson step tells you to tap something, the step's window has a tail pointing to the item so you really get the idea behind the action as well as its consequences.

The first lesson is about the book itself, explaining each feature and action that you can expect. It introduces the buttons that you'll use to move from one step to another and the stop sign that all steps have in case you want to quit the lesson. If you stop a lesson, you will

go back to where you started. Even if you stop, you can complete the lesson at any time; in fact, when you go back to the library, the *Getting Started* book even remembers what page you were on.

The next four lessons are about general user features like the top and bottom of the screen and the keyboard. You can finish these lessons sequentially, which will help you feel more comfortable using your communicator by the time you reach the vital lessons that you must complete. If you want, you can skip directly to the lesson about what you must do to personalize your communicator—making a name for yourself.


Required Setup

Before you can make an appointment, create a message, or register for a mail service, the communicator needs to know who you are. There are nine steps to this process, but four of them are navigation steps that take you from place to place, and one step is just an explanation of what you're about to do, so it's easier than it sounds.

If you follow the lessons in order, you'll know about typing by the time you get to the lesson that helps you enter your name. You'll know that the on-screen keyboard appears on the screen when you need to type in some information, and that Magic Cap shows where you're going to type when you need to enter your name, address, and phone number. You may notice that some of your entries are finished automatically, because Magic Cap knows some words and tries to guess what you want to enter so you don't have to type as much. If Magic Cap guesses wrong or you don't even notice that it has guessed, you can continue to type without having

to do anything else—your typing will replace Magic Cap's guess. You can find out more about this automatic word-completion feature in Chapter 5.

As soon as you touch the *done* button after filling in your name, Magic Cap puts up an announcement telling you that it's personalizing the communicator for you and that it may take a couple of minutes. You'll also see a spinning hat, which tells you that the communicator is actually busy thinking, not just waiting for you to do something.

 **Spinning Its Wheels.** When Magic Cap personalizes your communicator, it often takes a couple of minutes. What on earth is it doing all that time? It's calculating a security code that will be used to prove that you're who you say you are when you communicate with other people and services. It takes so long because it's a very big number that's undergoing a lot of calculating, and Magic Cap communicators were designed to be great communicators, not super-fast number-crunching monsters. The good news is that it only has to do this calculation once when you set up your communicator.

Next you'll be directed to fill in your address, and then your phone number. You don't even have to put the parentheses around the area code or the hyphen after the first three numbers; it will take care of that for you. However, you do need to enter your area code. If you don't, Magic Cap will make an upset noise, insert the empty parentheses before your phone number, and place the typing point there for you.

Magic Cap's designers knew that an undersize on-screen keyboard is not the world's easiest way to enter

information, so they made sure that Magic Cap is engaged in a never-ending battle to help you enter information more quickly. For example, when you're asked to fill in your address, Magic Cap guesses that your address starts with a number, so it sets the keyboard to its number mode automatically. If your address is a post office box, or something else that begins with a letter, you can easily switch it back to see the standard keyboard.

In addition, the keyboard also makes smart guesses about capitalization. After you've typed the numbers, the keyboard not only switches back to showing letters, but it also shifts to uppercase for you. After you type the first letter, the shift is removed and the letters are lowercase again.

Before you can send a message or even make a phone call, there is one more must-do task: You have to tell the communicator where you're calling from so it knows whether to include area codes and country codes when dialing numbers. You should definitely follow the *Getting Started* lesson for this one, because it's not the most intuitive process in Magic Cap.

Set up dialing is the way to tell Magic Cap about the places you expect to be when you connect a phone line. Your communicator can feel when you plug in a phone line, and it will ask your location every time it gets connected. Phone numbers must be dialed differently depending on where you are; for example, when you dial your home number from outside your own area, you must dial the area code first; when you call from inside your own area, dialing your own area code first will prevent the call from going through.

Again, don't let the number of steps fool you—four of the seven steps are just navigation to get to and from the *Getting Started* book. While you're setting up the dialing locations, you'll get to see the stamper, which is one of

the buttons that's always available on the bottom of the screen. You'll learn that the stamper contains lots of rubber stamps that you can use throughout Magic Cap, plus special-use stamps like the ones that fill out name cards and locations.

The standard stamps for locations are *home*, *work*, and *hotel*, but each stamp lets you customize the name, so you can have more than one office, add more hotels as you travel, or come up with other locations. When you set up your dialing location, you'll be asked to choose a stamp that matches your actual location.

After typing the name of your location, you'll enter your country, area code, and whether you need to dial a prefix to get an outside line first. If you've already filled out your name card with a phone number, Magic Cap will guess that you're in that number's area code, an example of Magic Cap's smart integration. You can read more about this phone location business in Chapter 6.

After you touch *done*, you will have completed the only two tasks required to start using Magic Cap. Every other lesson is just enhancement and teaching, but you'll probably want to complete most of them anyway.

If these two setup tasks are so important, why aren't they the first lessons in *Getting Started*? Before Magic Cap lets you complete these setup lessons, it teaches you how to use the skills you'll need to do them. It wants to make sure you know how to type, use the stamper, and navigate from place to place before asking you to do real work with those skills.

More Setting Up

After you finish the two required lessons, setting the time and date is a good lesson to go through next. Your communicator will let you do everything it can do

without making you set the time and date, but it won't be very accurate. For example, it wouldn't be very useful to have an appointment set to the wrong date or to send mail that has the wrong time in its postmark.

You can turn the pages in the *Getting Started* book until you find the lesson entitled *Set the time and date*. Once again, the 15 steps listed make it sound tedious, but 4 are for moving around in Magic Cap, and each time you set a number or time zone, touching the *accept* button to move on counts as a step as well, so it'll seem much easier than that. Figure 1-3 shows the first step in the lesson that sets the time and date.

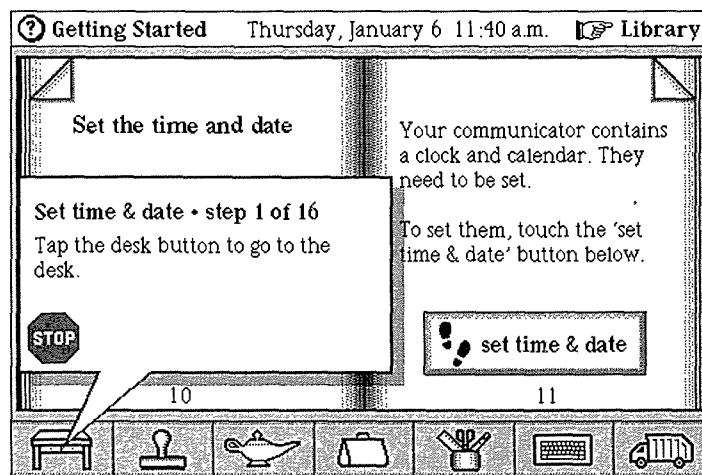


FIGURE 1-3. Starting to set the clock and calendar

As with some other lessons, the first thing you need to do is go back to the desk by touching the desk image on the bottom of your screen. By touching the clock behind the desk, it zooms up close enough for you to set it. The first thing you're asked to do is select a city in your time zone. Magic Cap wants to know this so it can adjust the

clock if you go to other cities. Your exact location may not show up in the list of cities you can choose from, so you can select any city that's in the same time zone. After the communicator knows what time zone it's in, it asks you to tell it what time it is.

After you set the time, you get to set the calendar to today's date. To set the date, you'll see a calendar page that lets you choose the year, month, and date. You'll use left and right arrows to help you move forward and backward in time. You finish up the lesson in the same way you've finished other lessons—you go back to the *Getting Started* book and tap *done*. Your communicator has learned a lot today; it knows who you are, where you're calling from, and the date and time.

The next lesson to go through signs you up for AT&T's PersonalLink service. This is another one that you should do, but you might choose to skip. As with all the others, you can either step through it with the help of the lesson, try its task yourself, or skip it altogether. This one is pretty straightforward, so if you're feeling really comfortable with your communicator, you might want to try flying solo on this one, going without the lesson. The only prerequisite is that you should be connected to a phone line to complete this process.

When you turn on your communicator, you'll find one or more messages in the in box above the desk—the exact number and the messages vary depending on which communicator you have. Some of these messages are offers from information networks to provide electronic mail and other services for your communicator, including one from AT&T.

When you tap an offer message to open it, you'll see that it has a button to request the signup materials from the service. If you're connected to a phone line, tap the button; the request will be answered with a return


message that includes the rest of the registration materials. Taking this lesson not only helps you register for a mail service that will give you a much more powerful communicator, it also helps you successfully and easily send your first message.

Lessons that Teach

We have now covered what you must do and what you should do to effectively and efficiently set up your new communicator. The other lessons found in the *Getting Started* book are more instructive than task-oriented: They teach you about the top and bottom of the screen and how to use the keyboard. By going through them, you'll have a better understanding of Magic Cap, its tools and commands, the places you can go in Magic Cap, and what you'll be able to do with your communicator.

The lesson about the top of the screen is short and sweet. It points out the areas along the top of the screen and describes each one. You'll learn that the top of the screen tells you where you are in the Magic Cap world, what the date is, and how much battery power you have remaining.

The lesson about the bottom of the screen is more detailed, pointing out the commands and tools that are available throughout Magic Cap. As each button is presented, it appears in its place along the bottom of the screen. The lesson describes each button in general terms: the desk button, the stamper, the lamp, the tote bag, the tool holder, the keyboard button, and the garbage truck. You can find out about all these things as you're working, but the lessons are like a shortcut to knowing how to use Magic Cap more effectively.

 **Making the Scene.** The space between the top and bottom of the screen is called the current *scene*. Common scenes include the desk, items on the desk such as the name file or a notebook page, the hallway and rooms in the hallway, the street downtown, or buildings on the street. For example, when you open the name file, a name card fills the screen; that's the name card scene. When you want to set the clock, you touch it and it zooms up close so that you can see it and nothing else; that's the clock scene. When you want to write a message, you touch the postcard at the center of the desk. This is like pulling the new card up so close that you actually see only the postcard scene. Magic Cap also includes windows, which are rectangles that float above the scene and are filled with items to help you perform actions or enter information. There's more about this stuff in Chapter 3.

The keyboard lesson has the most steps, but if you're going to be typing lots of information, the lesson is a few minutes' time well spent. As the keyboard opens, the lesson describes all the features built into the keyboard as it leads you step by step in typing a few words. You'll learn about the keyboard's smart capitalization and the switch that lets you change the keyboard from letters to numbers and symbols. The lesson also tells you about the typing point, the vertical line that shows Magic Cap where you want to begin typing.

The next lesson, *Add your signature*, is kind of fun to complete. You'll get a better feel for using the stylus as a writing tool on the screen as you sign your name. If you don't do this lesson right away, you can always add your signature at another time. Once you've completed this

lesson, you'll find stamps with your signature on them in the stamper when you're writing a message.

If you haven't added your signature but you decide you really want to sign your name on a message you're writing, you'll find empty signature stamps in the stamper. When you drop one of these stamps on your message, you'll go directly to the signature entry form, where you can sign on the dotted line. If you decide later that you're not happy with your signature, you can always repeat the lesson to change it.



Clip 'n' Save. Sometimes Magic Cap has to provide a visual representation of things that don't have a real physical metaphor, such as a few words of text or a particular style of shadow for an item on the screen. For such situations, Magic Cap provides coupons, complete with dotted-line borders, that represent those intangible things. For example, a coupon that represents a shadow style is good for one shadow in that style; you just drop it on an item and the item gets the new shadow. Before you sign your name, the signature stamps are really coupons that are redeemable for one stamp with your signature. When you put one on a message, Magic Cap will ask you to sign your name.

The last two teaching lessons in *Getting Started* actually step you through sequences that demonstrate the heart of Magic Cap, communication. One lesson helps you make a phone call to someone, and the other walks through the steps necessary to send a message. Once you're registered for the AT&T service, you can send a message to someone. You can use the phone lesson right away.

When you do the *Phone someone you know* lesson, you will be able to hear the call being dialed through the speaker in your communicator, but you'll have to use a handset or telephone to talk. After you've entered the phone number from the Magic Cap keypad, you can touch *dial* to have your communicator call for you. You'll get a *Phone status* window that lets you adjust the volume, and you'll see a timer that shows the duration of the call.

You can have a person-to-person conversation through the standard telephone, and then use the communicator to hang up. To complete the lesson, you can save the phone number by making a new name card for the person you called so you'll have the number handy the next time you want to call. Touch *done*, and you're back in the library ready to go on.

The last page of *Getting Started* congratulates you for finishing the lessons. If you tap *put away*, the book closes and hops back onto the shelf until you need to read it again. When you reach the end of *Getting Started*, you have finished setting up your communicator. You have also made a name card, used the phone, sent a message, written your name, and learned all the skills you'll need to get the most out of your communicator.

Lessons Learned

When you finish each lesson, the text on that page of the book is updated to reflect the completion of the series of steps, and the button changes to *repeat lesson*, which you can do whenever you want. After you finish the lesson, you will also get directions on what to do next.

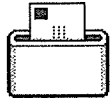
The lessons try hard to protect you from straying and doing anything else that might cause unnecessary frustration. If you touch the wrong item at any lesson step,

you'll hear a gentle reminder sound and see the proper location blinking on the screen. If you don't finish the required lessons, you can always complete the tasks on your own if you're the kind of person who learns best by experimenting, rather than by following a preset sequence.

Remember that if you don't complete the lessons, and you don't finish the setup manually, you won't be able to do most substantive tasks. When you try, a window appears and reminds you to personalize your communicator with your name or location; it even includes a button for returning you to the lesson with a single touch. And it says *please*.

The *Getting Started* book shows you how to use many of the tools and features available in your communicator and helps you set up vital information like your name and location. After completing the lessons, you'll feel more comfortable with the way Magic Cap works. Most important, if you've completed the *Getting Started* book, you've learned how to use your communicator for what it does best—communicate!

Chapter 2



Electronic Mail

Keep in Touch

Using technology to keep in touch used to mean picking up the phone, speaking to an operator, and asking her (yes, it was always her) to ring Grandpa Dave's house. As more people got telephones and the technology moved forward, customers felt more comfortable with phone numbers that looked like FLorida 5-2379 or KEystone 7-9855. Amazing advances let people use a telephone in their home to call someone in another city, or state, or even country. Telephones in public places worked by depositing coins, helping people stay in touch without having to stay at home.

Now, technology offers services that the venerable Alexander Graham Bell probably couldn't have imagined. We have phones that work without wires, giving you the freedom to call from your car, on a camping trip, or over the Rocky Mountains (although a lot of those conversations probably consist of "That's right—I'm calling from the airplane!").

As people became more mobile and used phones while they were away from home, they started to worry more that while they were away, someone might be calling them. Answering machines and voice mail were at first considered unacceptably rude—I used to think that if I wanted to call my friend, she should at least have the

decency to stay home and wait for my call, and I never liked leaving a message. Now many people consider it poor etiquette not to have voice mail or an answering machine, as an unanswered telephone inconveniences the caller.

Many people who resisted answering machines ultimately recognized that they give you freedom as both a caller and a receiver of calls. As a caller, you can fulfill your urge to communicate even if you can't reach anyone. When you're called, you can miss the call without missing its content.

This is the foundation of Magic Cap: power, flexibility, and convenience for the sender and the receiver. I want to communicate with you right now, but you may not be willing or able to hear from me right now. I can tell you what I need and go back to the rest of my life. You're free to get my communication at your convenience. Magic Cap expands this communication to cover electronic mail that contains digital information like words, pictures, sounds, cartoons, appointments, sketches, and more, along with the more-familiar phone calls and faxes.

This chapter covers four important scenes in Magic Cap: the in box, the out box, the AT&T building, and most important, the message-writing scene, which is where all new messages are created.

Communication at Heart

There are several elements at the heart of a Magic Cap communicator. The core of Magic Cap's communication features is the built-in Telescript language. While Telescript is transparent to you as a user, its communication features help Magic Cap deliver electronic mail features that you'll use in lots of different ways.

When you get a Magic Cap communicator, you can sign up for an electronic mailbox with one or more

services. In fact, AT&T has built a brand-new network service, AT&T PersonaLink Services, that is based on Telescript. Sending and receiving electronic mail is one of PersonaLink's key features.


Lots of people already have electronic mailboxes with other, existing services and networks, such as America Online, CompuServe, and Internet. Magic Cap lets you communicate with most of these existing services, even if you don't have mailboxes on those systems. It's kind of like this: You can use an automated teller machine that belongs to a bank where you don't have an account, as long as your bank has an agreement to talk to the other bank. As long as you're registered with PersonaLink, you can exchange mail with people on lots of other services. This communication between services happens through communication *gateways*.

Any time you enter a different world, which is what happens when you go through a gateway, you have to play by the rules of the new world. Sometimes, stuff gets left at the door and you can only send in what the other service can understand. For example, most electronic mail services don't know how to have pictures or sounds in their messages, so if you include those items in your messages, they probably won't get delivered to recipients on other services. Even with these limitations, being able to send mail to someone on another service without having to subscribe to it is very useful.

Some of the conventional mail services provide special software packages that let you use Magic Cap to connect to them directly. If you have one of these packages, you can use your communicator to talk to the service for mail and other information. Whether you're connected to PersonaLink or a conventional service, you can use Magic Cap to keep in touch with colleagues, family, and friends.

AT&T PersonaLink Services

Just what is PersonaLink? Why should you register for it? How can PersonaLink get in touch with people on other services? That's a lot of questions for some service from New Jersey. Users can send, receive, and store their messages with PersonaLink, and they can also use the gateways that PersonaLink offers to other conventional mail services. Electronic mail is only one feature of PersonaLink. AT&T's network service will take advantage of Telescript's intelligent agents to offer other kinds of communicating packages and information services.

 **Mostly Cloudy.** A Telescript-based service such as PersonaLink consists of one or more computers running one or more Telescript *engines*, or programs that understand Telescript. Engineers often refer to the computers and Telescript engines together as "the cloud." In honor of this nickname, the sky over Magic Cap's downtown always has a puffy cloud floating down the street.

When you first get your communicator, you'll find a message or two waiting in the in box when you turn it on—registration offers for PersonaLink and other services. After you've personalized your communicator with your name and phone number, you'll probably take a look at those messages. You'll find the signup procedure pretty straightforward. Depending on your model of communicator and the service you're signing up for, you might have to make sure you're connected to a phone line first. Then, tap *get signup form* at the bottom of the offer letter. A sample of one of these service offer messages appears in Figure 2-1.

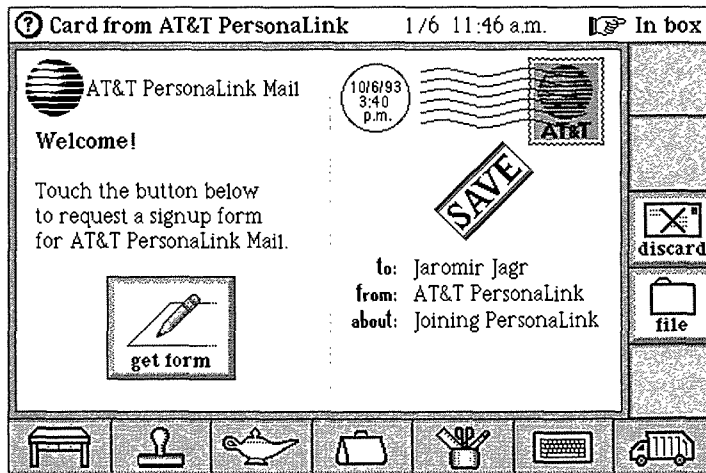


FIGURE 2-1. Request card for PersonaLink signup materials

When you tap the button, Magic Cap sends a message to the mail service requesting a signup form. You see the new message hop to the out box, and then watch a progress bar as the message is sent from your communicator to the service. When your communicator finishes, there's a new message in your in box—the signup form you asked for.

When you reply to the offer (again, you should plug into a phone line if your communicator and the service don't support wireless messages), your name card information is sent to the company providing the service so it can respond by sending you registration materials. If you're not connected to a phone line and you don't have wireless hardware when you respond, the message hops into the out box, waiting to be sent. Then, you can send the message when you connect to a phone line later.

Once you've signed up with a mail service, you can start sending messages right away.

Sending a Message

Because communicating is at the center of Magic Cap, the image of the postcard and pencil is right in the middle of the desk. To start making a new message, you just tap that postcard. There's also a desk drawer that holds a variety of stationery for other kinds of messages.

Magic Cap was designed around whole person thinking, which contends that people don't strictly separate their days into business and personal parts; instead, they weave them together, acting on impulses and needs as they come up. If Magic Cap succeeds, electronic mail will become popular for individuals as well as for businesses. Magic Cap provides clues to this focus. First, the devices are called *personal* communicators, and second, Magic Cap is factory-set to use postcards, not business letters, as the standard form of electronic mail, although you can easily switch to business letters instead.

With that in mind, let's go through a couple of electronic mail scenarios, a personal example and a business example.

Getting Personal

Imagine an old friend, Sheryl, whom you've known since grade school or maybe even kindergarten. Now, even though you live in different states, you still like to keep in touch beyond just birthday and holiday cards. A fact of your lives, though, is that while you might often think about her, it's difficult for you (and for her) to coordinate your time well enough to have a reasonable telephone conversation.

By writing an electronic message, you can jot down all the things you want to tell her about your life and ask her about hers, all while you're waiting for a staff meeting to

begin. When you connect to a phone line, you can let her know you've been thinking about her without having to worry about time zone differences. She can read your message and respond when her time is less complicated. Sheryl won't have to try to find a time when you're both home and not busy at the same time. In fact, this even works if she's out of town on business or vacation, because she'll certainly have her communicator with her.

To make the message, touch the postcard, and a new, blank message hops onto the desk and then zooms open to fill the screen. The new message also automatically opens the name chooser, a window that lists everybody in your name file, so you can choose your friend's name from the list (see Figure 2-2). If your friend isn't in your name file yet, you can add a new card for her by tapping *new*—you don't have to go to the name file to add a name. If you're not sure whom to address the message to right now, you can tap the *x* to close the name chooser.

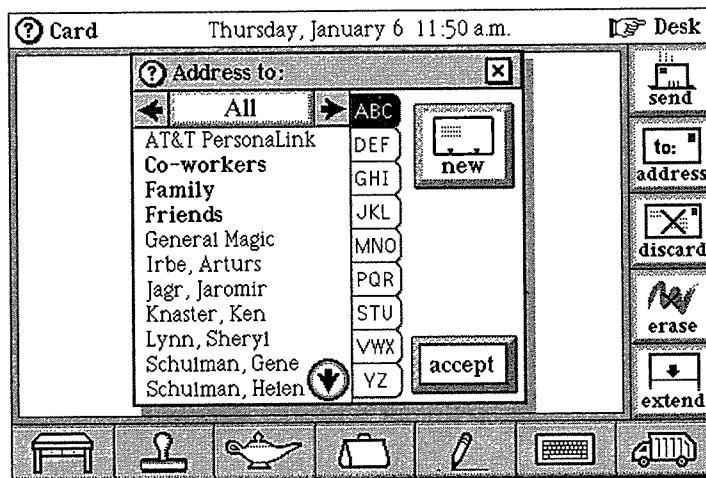


FIGURE 2-2. New message scene with name chooser

To address the message, touch your friend's name, and then touch *accept* to close the name chooser. You'll see that Sheryl's name has been added to the postcard as the addressee, and your name appears as the sender. You'll also find that Magic Cap has guessed that you'll want to start the message with a standard salutation, so it has typed *Dear Sheryl*, in the upper-left corner of the postcard. Of course, you can change this salutation if you want.

Magic Cap lets you choose between typing (for letter-perfect text) and writing (for drawings or a more personal feel). This is likely to be a fairly long letter, so you will probably want to type most of it. You can touch the keyboard image on the bottom of the screen to open the on-screen keyboard. If you find that you usually prefer to type messages rather than write them, you can customize Magic Cap so that it opens the keyboard instead of selecting a pencil when you make a new message.

When the keyboard opens, you can start typing directly below the salutation, or you can move to any other spot on the postcard where you might want to begin. Because it's been a while since the two of you spoke, you have a lot to tell your friend. After you've typed in your news and asked about her family, you might notice that Magic Cap has automatically enlarged the postcard for you so you can write as much as you need to without worrying about running out of room (see Figure 2-3). You can tell because an arrow pointing up appears when you reach the bottom of the card, but you can still type as if you had unlimited space, which you do. It would be great if paper postcards worked like that—then we wouldn't have to worry about writing so small. In the future, maybe there will be postcards for Magic Cap that have nice pictures on the back and we'll just send those instead.

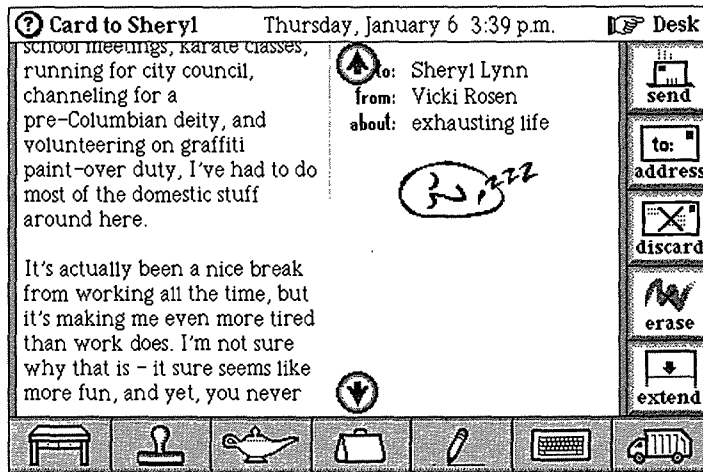


FIGURE 2-3. Longer text automatically extends the length of the card

When you're done typing, you can open the stamp drawer so you can stamp your signature and a sleeping face stamp (the theme of your postcard is "I'm working real hard and I'm tired"). If you want, you can touch *about:* to open the keyboard and fill in a description for the postcard, which would let Sheryl know if the card is just general news or if you really need something in particular, like her recipe for pickles.

Mailing the Card

As soon as you addressed the card, a stamp showing how the card will be delivered (that is, whether by fax, PersonaLink, America Online, or whatever) appeared in the upper-right corner. If you want to change the delivery choice, you can tap the stamp and choose another method. Magic Cap knows to list only the delivery choices that will actually work for the recipient; in other words, if

your friend doesn't have a CompuServe account, CompuServe isn't listed as a delivery choice. When you've made the right choice, tap *send*, and the postcard zooms down onto the desk and hops into the out box.

If you're connected to a phone line or have wireless access to the delivery choice you selected, Magic Cap will mail the postcard right away. If you can't connect, the mail will wait in the out box. Then, when a phone line is handy, you can open the out box, tap *mail*, and watch the message fly off to Sheryl. The next time she checks her mail, your message will be waiting for her.

Electronic mail has a different feel than a phone call. It's certainly not as interactive, but it can be more thoughtful, and you can communicate and keep in touch without having to find a time when you're both focused and uninterrupted. You get to write a letter with no pen, paper, envelope, stamp, or post office involved; it's delivered very quickly; and your friend gets to read and respond easily and at her convenience.

Replying and Forwarding

Of course, after Sheryl reads the message, you hope she'll want to respond. She can use a built-in shortcut for replying to your message. The command buttons along the right side of your message include choices for *reply* and *forward*. If you look at Figure 2-4, you can see these buttons along with the rest of the card as it looks when Sheryl receives it.

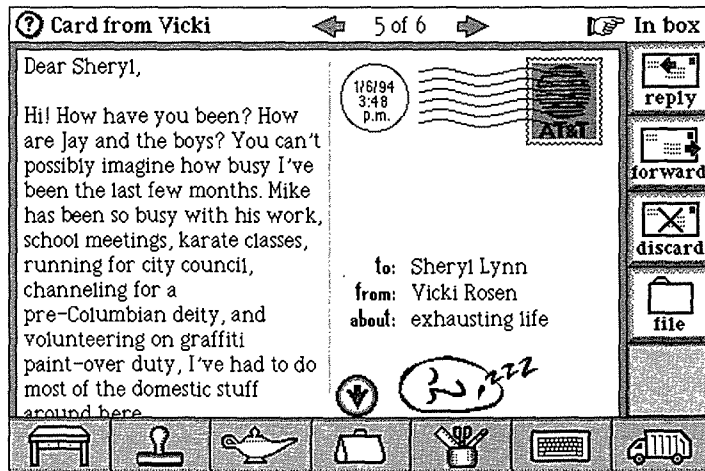


FIGURE 2-4. Message has been received and opened

When Sheryl is ready to respond, she can ask Magic Cap to create a new card by tapping *reply*. Because it's a shortcut, *reply* also fills in the address (back to you, the sender), guesses the salutation, opens the keyboard, and positions the typing point. A reply stamp appears at the bottom of the card, and an appropriate delivery choice is selected. Of course, she can change any of this stuff as she writes her reply. The next time you check your mail, her reply is there.

If the message wasn't just gossipy, but really was a request for the pickle recipe, you might also want to forward her reply to your friend Phil. To do that, tap *forward* on the right side of the screen while you're looking at her reply. Tapping *forward* makes a new card, opens the name chooser to let you address it, and attaches a copy of her message with the recipe. Once again, Magic Cap makes a delivery choice based on the information in the name file, and you can change the choice if you want.

Staying In Touch on the Road

One of the most frustrating situations in business is when you're out of your office and you need to communicate with someone else who is also out of the office. Your voice mail systems can have one-sided conversations, but there are times when communication needs to be more substantive and immediate. Electronic mail provides a thoughtful, reflective medium for getting your thoughts down just as you want them, a nice alternative to the ticking clock and live recording of voice mail.

Let's say you're the assistant director for an animation art gallery, and your gallery in Denver is just two days away from the opening of a big show. You're in Chicago, ready to accompany the animation cels from the corporate warehouse to Denver. The gallery director, Helen, is in Los Angeles, ready to accompany the artist on his flight to that show. Neither of you is in your gallery office, and neither of you is very reachable by telephone (just multiply the problems of getting messages from hotel operators by two). This is a job for Magic Cap.

Opening your communicator's desk drawer, you can begin a business letter to send to your boss by touching its image. When you touch the business letter, it hops out of the drawer and onto your desk, automatically opening the name chooser so you can address it to Helen. As with a postcard, you can always just touch the x to close the name chooser without picking an addressee right away.

The business letter automatically includes the sender's name in the upper-right corner of the letter (like letterhead), adds today's date, and then the salutation to the addressee, in this case *Dear Helen*. It also opens the keyboard and places the typing point for you. You need to ask her about which art pieces should be sent for the

show, as well as what kind of wine and hors d'oeuvres to serve at the opening, and when the artist will be available for interviews with the local press.

Consistent with other pieces of "paper" in Magic Cap, a business letter can be extended if you need more room at the bottom. After you finish your message, you close the keyboard. You can see the completed letter in Figure 2-5. If you've already addressed the letter and you agree with the delivery choice Magic Cap suggests, you can tap *send* to mail the letter.

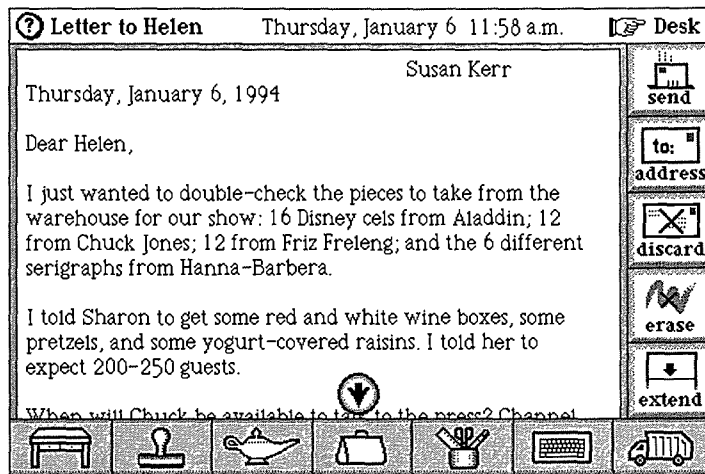


FIGURE 2-5. Business letter automatically adds heading information

Unlike the postcard, where you can see the postage stamp that tells how the message will be delivered, letters put that information on their envelopes. By touching *address* on the right side of the screen, you'll get the standard options to add or replace addressees, and you'll also be able to tap a special button labeled *show envelope*. If you do, you're switched to a different view that

resembles the front of an envelope with a company logo in the upper-left corner (see Figure 2-6). You can stamp anything you want on the envelope, or take it off completely by sliding it to the trash.

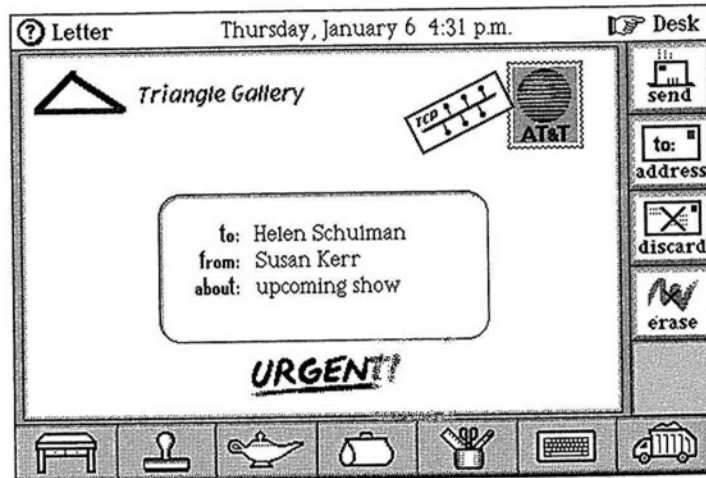


FIGURE 2-6. Envelope for the business letter

The envelope also has the postage stamp message in the appropriate place in the upper-right corner, and the recipient's name (as well as yours, and an *about:* description) in the center of the envelope. Here's where you can choose how to send your letter. When you communicate with Helen, you usually send a fax because you're typically out of the office while she's at the gallery. This time, you'll use her electronic mail address to send the letter, and you'll also stamp it urgent. You might guess that you should stamp *urgent* on the envelope and not the letter itself, but the urgent stamp will alert Helen's mailbox whether it's on the envelope or on the letter.

After you've switched delivery choices, you can touch *address* again, then *hide envelope* to switch back to the

letter. If you're done, you can touch *send* while you're looking at either the letter or the envelope to mail your message. Since Helen has set her in box to alert her when an urgent message comes in, and her communicator has a two-way radio for wireless access to electronic mail, she's able to read your message and respond within minutes. Your mind is eased much more quickly than if you had to wait for her to get back to her hotel, pick up your phone message (if she received it at all) and try to call you back in Chicago. You've helped ensure the success of the big event.



Following the Metaphor. The differences between Magic Cap letters and postcards are very similar to the differences between actual letters and postcards. Postcards give you a smaller amount of space to write in, your writing isn't hidden inside an envelope, and the addressee's name is printed on the right side of a line that bisects your message. Letters start with a plain piece of paper (although Magic Cap business letters automatically add items you would expect on a business letter), and the address is written on the envelope that holds your message. These are subtle differences, but they're important, especially if you recall Marshall McLuhan's adage about the medium being the message.

Group Therapy

As roads get clogged and gas prices increase, carpools are becoming more popular. Carpools that drive kids to school can be challenging enough to deal with, but for serious hassles, try coordinating a carpool for adults who work in the same office building. Let's see how Magic Cap might help you manage a carpool that takes five people to work each morning.

You're responsible for driving four other people to work one morning each week (because of your varying departure times, you each find another way home at the end of the day). You just found out that you will be out of the office for a week (starting tomorrow) overseeing the beginning of a client's construction project. You need to let the other people know that you'll miss your regular turn driving, as well as not needing a ride the other four days of the week. Everytime something like this comes up, it means trying to catch up with four other busy professionals to rearrange the carpool. You could make four different phone calls, one to each of them, but Magic Cap has a way to let you be more efficient.

Magic Cap helps you handle these kinds of problems by letting you collect name cards together into groups. Any set of names with something in common can form a group. We'll create a group to help manage our carpool. Tap the name file on the desk to open it, tap *new*, then *group*, and then type *Carpool* to name the group. Tap *add* to put people into the group. The result is a filled-out group card, as you can see in Figure 2-7. You can find out more about name cards in Chapter 5.

Now that all the carpool members are reachable on electronic mail or via fax, you can send one message to four people detailing your change in plans.

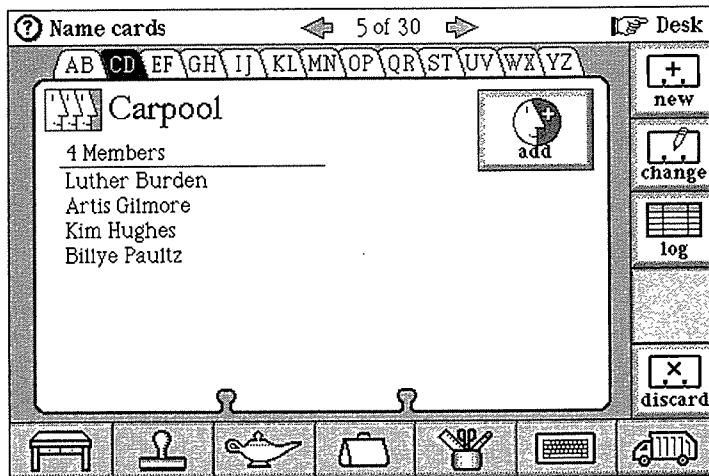



FIGURE 2-7. Group name card helps manage multiple addressees

After you tap the postcard to start writing a new message, you can pick *Carpool* as the addressee. The name chooser lists all groups in boldface to make them stand out. You can verify that all the recipients are included in the group by option-tapping *Carpool* in the addressee area, which opens a window listing the members of the group. If you then touch a name in the window, the delivery choice stamp will change to show how the message will be sent to that addressee. Magic Cap lets you have a different delivery choice for each member of a group.

When you tap *send*, the message will go to each member of the group using the delivery choice that Magic Cap guessed was right, or that you corrected. In this case, two copies will go by PersonaLink, one by America Online via a PersonaLink gateway, and one to a work fax number. There are four addressees and three different delivery choices, but you only have to send one message.

 **Testing Patience.** The General Magic team ran into an unexpected and embarrassing problem while testing PersonaLink. A General Magic tester plugged a communicator into a phone line that didn't go through General Magic's switchboard, but mistakenly instructed the communicator to dial 9 for an outside line before calling any numbers. When the communicator called AT&T's access number, which begins with 1-800, it actually started by dialing 9-1-800. Naturally, the telephone network responded with a message to "please dial a 1," determining that the communicator was calling a long distance number in area code 918.

The diligent tester took the message at its word. Thinking that Magic Cap had somehow forgotten to dial the initial 1, the tester cleverly told Magic Cap to dial a 9 for an outside line, then a 1, and then AT&T's number. The next time the communicator tried calling AT&T, it dialed the extraneous 9, then an equally inappropriate 1, followed by the proper 1-800 number. That makes 9-1-1, then 800 and the rest of the number. The PersonaLink service didn't answer, so the tester tried again a few times.

When the police officers and firefighters arrived, they were stern but understanding.

Choosing How Messages Are Delivered

Let's look at the many ways you can send a message. When you create a new message and address it to someone in your name file, Magic Cap puts a postage stamp in the upper-right corner of the message. The postage stamp shows how the message will be delivered to the addressee. That information is called the *delivery*

choice. If you've entered delivery information on the addressee's name card, such as a fax number or electronic mail address, the postage stamp shows one of the ways that you can get a message to the addressee.

If an addressee has more than one way to get mail, you can tap the postage stamp to see a choice box that lists the addressee's delivery choices. This works for multiple addressees, too; you can touch the name of an individual addressee to set the delivery choice for just that addressee. As you touch each addressee's name in turn, the postage stamp changes to show the delivery choice for that addressee.

How does Magic Cap decide which delivery choices to offer for a particular addressee? The easy answer is this: It offers all the valid ways of getting the message from you to the addressee. The full answer is more complex. To learn all the valid ways of sending the message, Magic Cap must consider which services you belong to and which services the addressee belongs to. For example, if the addressee has a Prodigy account, Prodigy will appear as a delivery choice only if you have some way of getting mail to Prodigy, either with your own Prodigy account or a gateway to Prodigy from a service you belong to.

So, when building the list of delivery choices, Magic Cap compares the services that you and the addressee belong to, also considering gateways that might get a message flowing between the two of you. But wait, there's more. If the addressee has any fax numbers, they're always listed as delivery choices, because every Magic Cap communicator can send faxes.

Finally, Magic Cap checks to see if you belong to any electronic mail services that know how to look up their members' addresses. Given an addressee's name, these services can check to see if the addressee has a mailbox

with them. This feature in an electronic mail service is called *directory lookup*. If you belong to any services that have directory lookup, Magic Cap will add those services to the list of delivery choices for all addressees. PersonaLink is an example of a service that does directory lookup, so once you're registered for PersonaLink, it will appear as a delivery choice every time you address a message.

Here's a summary of how Magic Cap makes its list of delivery choices:

1. Magic Cap checks your name card and the addressee's name card to see if there are any services that you both belong to. If so, those services are added to the list of delivery choices. For example, if you're both CompuServe members, *CompuServe* becomes a delivery choice.
2. Magic Cap checks your name card and the addressee's name card to see if any gateways offered by services you belong to can communicate with any gateways or services that the addressee belongs to. If so, those choices are valid too. For example, if you have an Internet account and the addressee has an MCI Mail account, Magic Cap can determine that MCI Mail has an Internet gateway, so *MCI Mail via Internet* will be added to the list of delivery choices.
3. Magic Cap checks the addressee's name card for any fax numbers. They're added to the list of delivery choices.
4. Magic Cap checks your name card to see if you belong to any services that offer directory lookup. If so, those services are added to the list of delivery choices. For example, if you're a PersonaLink member,

PersonaLink (trying directory) will be added to the list of delivery choices for this and every other addressee.

If you know that an addressee belongs to a particular service, but you don't have the addressee's account number, you can send a message to that addressee only if the service offers directory lookup. *PersonaLink*'s directory lookup feature is particularly powerful and provides several advanced features.

If you're registered with *PersonaLink* and you want to send a message to your cousin Arturs Irbe, who is also on *PersonaLink*, you can send the message and ask *PersonaLink* to look him up. When you address the message, you can tap the postage stamp and pick the *PersonaLink (trying directory)* delivery choice. As detailed, the delivery choices will also include any services that can get mail from you to him, including his fax numbers.

Because you picked the *PersonaLink (trying directory)* stamp, the message will be delivered to the big *PersonaLink* cloud in the sky. *PersonaLink* looks for any members it has who are named Arturs Irbe. It even looks phonetically, in case you're a bad Latvian speller. If *PersonaLink* finds nobody who matches, you'll get an electronic *return to sender* message back from *PersonaLink*. If there's exactly one person with that name, *PersonaLink* will go ahead and deliver the message, and also send you a new and improved name card with his account number.

If *PersonaLink* finds more than one match, which certainly could happen, *PersonaLink* returns the message to you along with a list of the matching names it found, including additional information like area codes to help identify the correct person. Because cousin Arturs lives in San Jose, the person with the 408 area code is most

likely to be him. After you decide which one is right, you can resend the message to him and Magic Cap automatically updates your name file. Hey, this is even better than calling directory assistance.

You don't have to send a message to someone in order to have PersonaLink look them up. You can also go downtown to the PersonaLink building and get a directory lookup form there. Fill out the information as completely as you can and send it in to PersonaLink. If there's a match, PersonaLink will send you the name card. Figure 2-8 shows a directory lookup form.

Letter 1/6 12:08 p.m. PersonaLink™ Center

Directory request

AT&T PersonaLink

name

If you know the area code or phone number, it will help find the right person.

phone

The address card for this person will be sent back to you the next time you get your mail.

send

to: address

discard

erase

extend

FIGURE 2-8. PersonaLink directory lookup form

Fax and Beam

Although the folks at General Magic and AT&T probably wish it weren't so, it will probably take a short while before you can reach everyone you know on electronic mail. Until then, you'll sometimes have to factor in more

traditional means of communication. Most offices, plenty of homes, and even a few cars have fax machines. Your communicator can easily send messages via fax. If you're telecommuting and you need to send something to the office, faxing it might be the easiest and fastest way to get it there.

If it's just a short message, write a postcard or letter and address it to the office fax machine. This is a good excuse for you to enter a name card for your office. Your office's fax number entered there will show up as a delivery choice, and your communicator will send the message just as if it were full-fledged electronic mail. You'll also find a fax command always available inside the lamp. If you sketch a floor plan in the notebook and want to send it to your architect for her advice, you don't have to attach it to a letter. Just tap the lamp, then fax, and you'll see the fax window, which lets you choose exactly what to send, who to send it to, and whether you want to include a cover page. When you're ready, tap *send fax* to deliver your sketch to the architect's fax machine. Figure 2-9 shows the *fax* window.

Magic Cap offers another way to send information that's fast and cheap: infrared beaming. Every Magic Cap communicator has an infrared transmitter and receiver that you can use to send messages, notebook pages, name cards, and other items. Of course, you have to be within a few feet of your recipient's communicator, but that's how infrared technology works.

To send via infrared beam, tap the lamp to open it, then tap *beam*. You'll get a list of what you can send and who you can send it to, as your communicator locates all recipients who are in range. Tap *send* to beam your information.

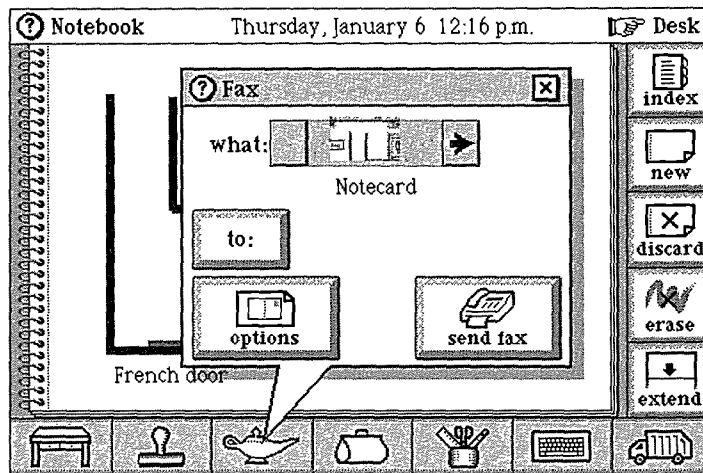


FIGURE 2-9. Fax is available everywhere in Magic Cap

Customizing for New Messages

When you make a new message, Magic Cap opens the name chooser and asks you to address the new message as it zooms open. If you don't want to address the message right away, you can touch the x to close the name chooser and address the message later. If you never want to address messages as soon as they're created, you can ask Magic Cap not to show the name chooser when you make a new message. To do this, tap the lamp, then *rules*, then tap rule 1, *Address new messages right after creating them*. to turn it off.

Every kind of stationery can be set to choose a particular tool when you create a new message on that kind of paper. For example, when you create a new postcard, Magic Cap chooses the thin pencil tool. If you want to change this behavior, you can set a rule to choose a

particular tool, such as the thick pencil, when you create a new postcard. If you prefer to type your postcards, you can turn the rule off.

When you're addressing a message, you might want to change the addressee, or add another addressee. When you tap the *address* button on the right side of the screen, you can add a new addressee to those already listed, or you can replace the entire list (see Figure 2-10). If you tap *replace addressees*, the name chooser appears and you can pick a new addressee.

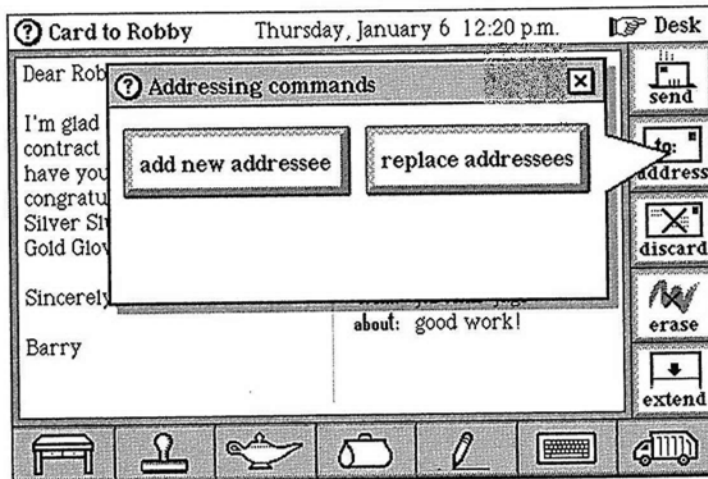


FIGURE 2-10. Addressing commands window

If you want to send a message to multiple addressees, you can tap *add new addressee* and pick from the name chooser. If you want to add more than one addressee at a time, here's a shortcut: Touch a name, and then option-touch *accept* to add that name as an addressee and still leave the name chooser open to add more names.

The name chooser also includes a choice box that lets you pick different addressee types. Choices include the

standard *to*; *cc*: for carbon copy (and it's amazing that this term is still around, since nobody actually makes carbon copies anymore); *bcc*: for blind carbon copy, when you want to send a message to someone without showing that it's going to that addressee; and *reply to*; which lets you specify someone else to receive replies to the message. If you're sending a letter rather than a postcard, the *Addressing commands* window also provides a button that lets you see the envelope and hide it once you've opened it.

Just because the plain postcard is located in the middle of your desk doesn't mean you have to use it every time. It's often appropriate for a quick message or a personal note, but if you're sending a business letter, you'll probably want to use more formal stationery. A postcard has the written message out in the open on the left side of the card, and the addressing information on the right side, just like a real postcard. If you tap the drawer on the left side of the desk, you'll find stationery inside. The letters, both plain and business, come with envelopes. The envelopes carry addressing information, a postage stamp, and a postmark. Letters and postcards look different in your in box, but when each is opened, its message appears. Take a look at Figure 2-11 to see what's in the stationery drawer.

When you make a new business letter, Magic Cap prints your name at the top of the page; it then adds the date, the name and address of the addressee, and the salutation, and also opens the keyboard. If you option-tap the business or plain letter, you can edit the stationery itself, not just a sheet of it. You can change the information you want to appear automatically when you make a new letter. For example, if you're sending a business letter, you might want to use your company's name rather than your own.

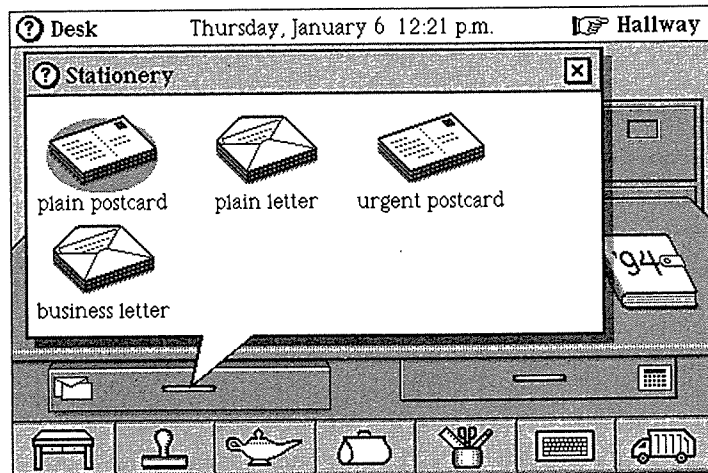



FIGURE 2-11. Different kinds of stationery in the desk drawer

When you option-touch *business letter*, you can see where it auto-types the sender name. If you change that to your company's name, every time you send a business letter, it will appear to be from your company. You can also edit the other information it automatically types, if you want.

To send a personal letter, you can touch *plain letter* in the stationery drawer. It's the image that resembles paper and an envelope. Letters appear in the stationery drawer as pieces of paper, not as postcards. There's also an urgent postcard if you want to make sure the message is read right away. The urgent postcard has *urgent* stamped on it, which will cause special handling when the addressee receives it. The urgent stamp is also available in the stamper, if you decide to make your message urgent after you've written it on a plain postcard. The *urgent* stamp on the postcard, letter, or envelope causes the special handling, whether it's automatically stamped or manually added.

If you use your communicator primarily for work, you might want to replace the plain postcard in the middle of your desk with the business letter. You can do this easily, and you can change it again whenever you need to. To change it, open the desk drawer where the stationery is kept and then slide a business letter out onto the desk. You can then option-slide the business letter onto the postcard in the middle of your desk, and the business letter replaces it as the standard kind of new message. Of course, you can always open the drawer and pick another kind of stationery.

 **Pay Attention.** If you watch closely when you tap the new message image in the center of the desk, you'll see that the message springs out of the stationery drawer instead of zooming out of the new message image itself as you might expect. This is intentional. The idea is to draw your attention to the stationery drawer so that you'll go exploring there. By having the new message hop out of the drawer, Magic Cap's designers hope to lure you into learning about the different kinds of stationery.

If you don't find the kind of stationery you need, you can make your own custom stationery to keep in the desk drawer, the ultimate way to personalize your letters. When you create your own stationery, you can make original drawings or scribblings, or you can use the goodies that Magic Cap provides, such as stamps and animations. Start with a plain piece of paper by tapping *plain letter* in the stationery drawer; then close the name chooser, leaving the new message unaddressed and blank. Design your stationery to look however you want.

You might add a nice face from the stamper's *faces* drawer. If you're really hip, you might choose to stamp your stationery with the animated character that sits, sleeps, hops, and spins. After you've finished, you can slide the letter to the stationery drawer and it will snap into place. You can option-tap the keyboard image to open the keyboard with label maker, then type a name for your stationery and drop the label on it to give it a name. You'll have a new kind of paper that you can use when you want to send a letter with a personal and funny touch—and when was the last time you could send someone a letter that hops?

Out Box: Where Messages Go

Let's go over what happens when a message is sent, besides just the communication from you to someone else. Your communicator has an out box, which is a launching pad for messages on their way out. If there are messages waiting in your out box, you can open it to see them, or slide them out to stop them from being sent. Tap the out box to see the messages that are inside it. Figure 2-12 shows you what the inside of an out box might look like.

Although you can open the out box to see the messages inside, and even open the messages themselves, it's not a great place to peruse your mail. That's because mail in the out box might be getting sent while you're looking at it, and it's liable to get filed away while you're in there. If you really want to look at mail that's in the out box, you're probably better off sliding it out onto the desk, which will prevent it from being sent while you're looking at it.

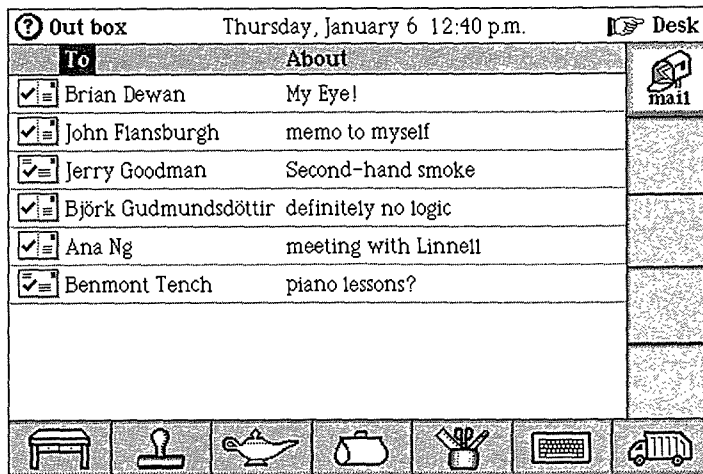


FIGURE 2-12. Messages in out box

You might think of the out box as a pretty simple place that just sends messages on their way, but Magic Cap and its Telescript components let you control your out box's behavior in a number of interesting ways. This control falls into two general areas: when messages in the out box are actually sent, and how they're handled after they're sent. These settings are controlled by rules in the out box.

When to Send Messages

Usually, you'd like messages that go to the out box to get the heck out of the communicator as soon as possible so they can get to their addressees. That's why the rule for when to empty the out box is set at the factory to send everything as soon as possible. If you're in the habit of making a bunch of messages at a time, you might want to change this rule so that you can write all your messages and then send them all at once.

If you're not plugged into a phone line and you don't have wireless hardware and mail access, this rule doesn't take effect until you're plugged in, and the out box holds all the messages until you're connected. Once you're plugged into a phone line, you can tap the out box to open it, then tap *mail* to send any messages that have been waiting around.

The out box also has a rule you can set that will empty the out box as soon as you send an urgent message. This one lets you keep the usual "send everything" rule turned off, but then sends everything when you fire off something urgent.

What Happens After Messages Are Sent

Whenever you send a message, the fact that you sent something is automatically logged with the time and date of sending, as well as a description of the message. The log is available by tapping *log* on the name card of the message's addressee. After you send the note, you could open the name card in the name file and then tap *log* to see when you sent the message and its description.

The out box includes several rules that determine what happens to mail after you send it. If you want to keep more than just a log, you can use rules to help file the messages themselves in folders in the file cabinet. There are three kinds of rules you can set for filing outgoing messages. You can file according to text in the message, you can file based on attributes set by stamps on the message (urgent, confidential, and so on), or you can set a catchall rule that will file everything not handled by the other rules. Figure 2-13 shows the three rules for out box filing that are set at the factory.

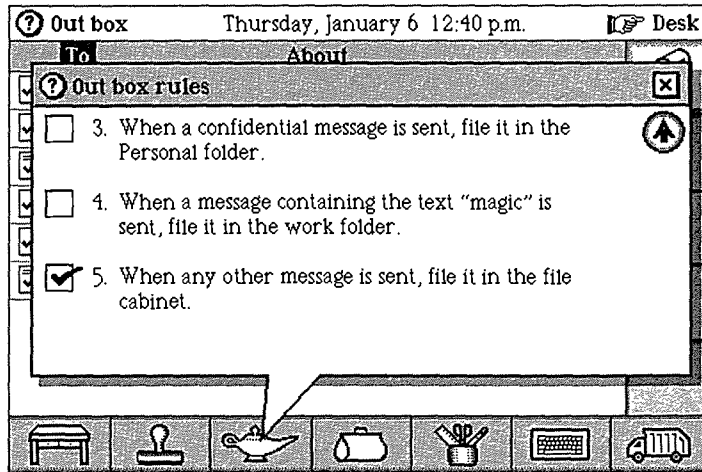


FIGURE 2-13. Out box rules for filing sent mail

You might make a folder for health information and then set a rule to file everything containing the word “health” in that folder. You could have a folder just for urgent mail, which will provide you with a file to explain to your boss why you deserve a great performance review and raise. If only you could find a person this efficient to file the rest of the stuff in your life!

In Box: Where Messages Arrive

When you collect your mail from any services that you belong to, your messages arrive at the in box. To see your mail, you tap the in box to open it, then touch a message to read it. When you’re looking at a message in the in box, you’ll see the familiar arrows at the top of the screen that let you move to the next or previous message, assuming you have more than one message in your in box. There’s a picture of this in Figure 2-14.

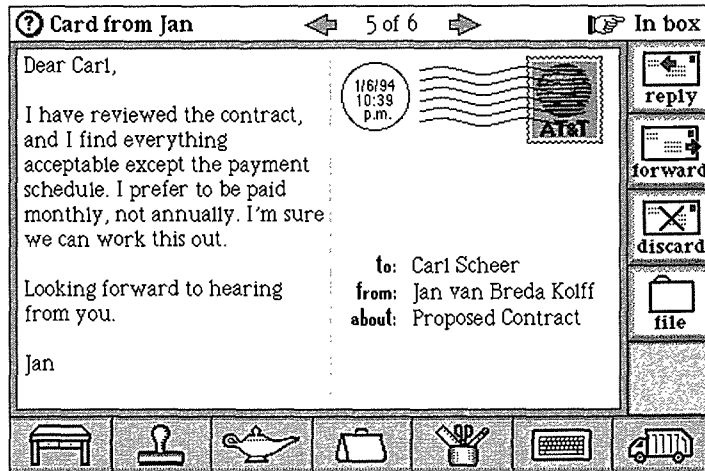


FIGURE 2-14. Open message shows how many messages are in the in box

When you open the in box, you can collect your mail by tapping *mail* along the right side of the screen. When you tap *mail*, Magic Cap connects to the mail service and gets your mail.

When you're looking at messages in the in box index, the messages' images change depending on whether they've been read and whether they're postcards or letters. Messages that have been read have an empty gray rectangle, while messages you haven't opened yet have a filled-in white image. You can sort the messages in your in box by any of the headings (sender, date received, and subject) just by touching a heading.

You can file all the messages at once by tapping *file all*. You might want to file messages individually as you read them, but *file all* is a great option for people who like to set up specific rules for where mail gets filed, which you can do in the file cabinet itself. When you file with *file all*, all the messages go into the received mail drawer in the

file cabinet, winding up in folders that you have set up to hold certain kinds of messages. See Chapter 8 for more information about setting up folders in the file cabinet.

If you're registered for PersonaLink, you can open the lamp and use the *summary* button to get a quick list of the mail waiting for you at the post office. You can choose which ones to have delivered to your in box, which to leave alone for a while, and which to throw away unopened. If you want to get rid of all your undelivered mail, the *clear* button does just that—it removes all the items addressed to you without bothering to get them.

In the real world, you don't have to bother with getting a summary of the mail that's waiting for you at the post office, and if you want to throw anything away, you can do so after it's delivered. Why do these commands exist? Delivering electronic mail costs you money and takes time, especially on a wireless network. This harsh reality requires commands like *summary* and *clear* that give you more control over which messages you receive.

Announcing Incoming Mail

Magic Cap's in box lets you exercise some power about how you want to handle incoming mail. There are two sets of rules that help you deal with your mail as it arrives. You use the first set to determine how Magic Cap informs you that you've got mail, and the second set lets you sort your incoming mail into folders in the file cabinet if you prefer to organize your mail before you read it.

You can set a rule to play a special sound for certain senders, with another rule playing a different sound when a message from just anybody arrives. You can set as many different "special sender sound" rules as you want. Another rule lets you instruct Magic Cap to display an

announcement when you get a message with a certain attribute, such as *urgent*.

Sorting Incoming Mail

If you like, you can get your in box to sort your mail before you even open it. You can tell your in box to look for mail with certain special attributes, like urgent or confidential, and file those messages in a particular folder as soon as they arrive. Setting this rule tells Magic Cap to file your mail as soon as it's collected, before you even read it. This automatic sorting and filing feature can be very useful to people who get a lot of mail and need to prioritize how they read it.

You can set a rule that looks for some particular text and puts messages that contain that text into specific folders. This works for text in the message as well as certain senders. It's interesting to note that one of the places you can automatically "file" mail is the trash, which might be something you'll do if you get inundated with junk mail.

If you're a PersonaLink member, you can ask Magic Cap to collect your mail automatically at a set time each day. This lets you plug your communicator into a charger/ phone line before you go to bed and tell your communicator to go to the post office for you and pick up your mail (of course, it really just calls up PersonaLink and gets the messages stored there for you, but saying it that way sounds more like real life and is more fun).

Going Downtown

In addition to telling your in box and out box how to behave, you can also go directly to the source and set rules for handling your mail before it gets to your communicator. You can go downtown to get to the AT&T

PersonaLink building (tap the upper-right corner to go to the hallway, and then tap it again to go downtown). Knock once on the building to open its doors and go inside. From the lobby of the building, you can perform several tasks, including checking or setting some mail-handling rules.

Telescript-based electronic mail services have a store-and-forward capability, which means that they hold onto your mail for you (that's the *store* part) and then send it along to you when you want it (that's *forward*). The mail service's building represents the place that does the storing and forwarding: It holds onto your mail until you're ready for it, at which point it sends mail to your in box, kind of like a post office would.

While you're in the PersonaLink building, you can set some preferences about how you want it to work for you. There are two sets of rules. One automatically forwards mail that meets certain criteria, and another automatically throws away certain mail. You can tap the mailbox rules sign to customize those settings. You can set a rule to forward copies of all messages from a designated sender to a designated recipient.

You can also arrange to forward messages about a certain topic to a specific recipient, such as forwarding a copy of all messages about baseball to Darin Adler. You can tell PersonaLink to discard certain messages before you've even collected them. If you're really sure that you don't ever want to read messages from a particular sender, or about a certain topic, you can set rules to toss them unopened. You'd better be really sure, though, because once they're discarded, they're gone. You can't even dig them out of the trash.

If you have a pager, you can set rules that tell PersonaLink to page you. For example, you might want

to be paged for a message stamped *urgent* or any message from a particular sender.

Downtown of the Future

If Magic Cap succeeds, downtown will be a swinging place where you can use your communicator to connect to all kinds of information services and stores. You might imagine being able to visit a travel agent, request information on flights to Cleveland for a sales conference, and also find out about taking your family to Walt Disney World afterward. You might still prefer to talk to your brother the travel agent during his business hours, but if you suddenly need to make your reservations at 10:30 at night, you could use your communicator to take care of it. A clever Telescript-based news agency downtown could know about your interests, collecting and forwarding just the news that you asked it to find.

You can imagine that if you wanted to get more information about health care reform, foreign investment in Germany, and Pierce Brosnan, you could ask the news service to keep an eye out for those topics and send only articles that cover those topics. Then, depending on the rules you set for your in box, you could have these articles filed automatically into folders set up just for them, letting you read them at your convenience.

If you're one of the millions of people who loves shopping with catalogs or televised shopping channels, you'll definitely want to try going to an online store to browse or buy things. Zarko's Department Store could have pictures and descriptions of dozens of items for sale and include an easy way to use the communicator to send an order or dial a customer service operator. You might also specify what items you're looking for, such as gold earrings or flannel nightshirts, and then ask the store to let

you know when those things go on sale or when new merchandise comes in.

Summary

Magic Cap was created for communication. Telescript, which is built into Magic Cap, offers smart messaging features that Magic Cap uses to make communicating easy. In addition, Telescript will be the foundation for information services, which will then be able to offer even more powerful communication features for Magic Cap users. One such service is AT&T PersonaLink, which takes full advantage of Magic Cap's features. PersonaLink also offers gateways to other conventional electronic services, such as CompuServe and Internet. If you subscribe to PersonaLink, you'll be able to contact anyone else on these and other services through the PersonaLink gateways. There will already be one or two messages waiting in the in box when your communicator first comes to life, and one will be a signup request for PersonaLink.

Making a new message is the central feature of Magic Cap. The postcard is the central item on the desk, and when you make a new message, it actually hops out of the desk drawer, showing you that there are other kinds of stationery for you to use. The stationery forms look very much like their counterparts in the real world. Postcards show the addressee on the right, and you can put your message on the left side (Magic Cap lets you extend the postcard's length as much as you need to). Letters can have business headings, and plain letters are just blank pieces of paper, but both have envelopes that handle the addresses and delivery choices.

Consistent with Magic Cap's strong integration of features, you can add an addressee to your name file while

you're making a new message, and even add a delivery route (fax number, electronic address) while you're there. You can also send messages to multiple addressees and groups from the name file. It's easy to choose different delivery choices for each addressee, even for different members of a group.

Lists of delivery choices are constructed from the contact information listed on an addressee's card, as well as information from your name card about which services you subscribe to and whether those services have gateways to other systems. All fax numbers will be listed as delivery choices, because all Magic Cap communicators can send a fax. If you know someone has an account on a system that you subscribe to (or can reach via a gateway) and you don't know the address, you may be able to use a directory lookup feature of that particular service. You can address the message to just a single person, to multiple addresses, or to a group, and you can also send copies. Messages can be carbon copied and blind carbon copied, and you can even have replies forwarded to someone else.

Although the image that sits in the middle of the desk is a postcard, you can change it to anything else that's in the stationery drawer or that you make yourself just by sliding the desired paper out of the drawer and dropping it onto the image of the postcard while you hold down the option key. You can choose from a plain postcard, urgent postcard, plain letter, or business letter. The business letter automatically adds business-like headings to the top of your letter, but you can edit this by option-tapping the stationery to change the automatic typing. You can also stamp your company logo, or write in your company name and address as letterhead at the top of the business letter or the envelope. You can even design

entirely new kinds of stationery to keep in the drawer for special occasions.

Your out box is customizable with rules that tell it when to send messages, as well as what happens to your copies of the sent mail. The in box also has rules you can customize. You can instruct it when to automatically collect your mail from information services, how to sort mail before or after it's read, and how to alert you to certain kinds of messages; for example, an urgent message should always tell you it's arrived—otherwise, urgent doesn't really mean anything.

PersonaLink acts as the post office, and it, too, has customizable rules for handling mail. You can ask it to sort your mail by certain senders or stamped attributes (urgent or confidential) or even to discard it before you've seen it. All of these different ways to handle mail may be a bit tedious when you're setting them up, but they let you tell your communicator when to "go" to the post office to collect the mail, to look for urgent messages and send them immediately, to watch for certain kinds of mail and file it directly into your junk mail folder, and to have the rest sorted by sender and delivered to your in box. The best part is that it all works just fine even if you don't change the factory settings.

You can take advantage of electronic information services that are available downtown. News retrieval services that can be customized to send you articles only about topics you have selected and retail and service outlets that offer you electronic shopping or airline reservations are just some of the possibilities available in a well-developed Magic Cap downtown, where everything revolves around the ease of communicating.

Chapter 3



General Features

What You See Is What You Get

I never heard this funny word *interface* until computers came into my life. Not only did personal computers change the way people work and spend their leisure time, the computer culture added many new words and new meanings of familiar words to the language. Well-known old words like *window*, *menu*, and *mouse* now need to be understood in context so that they're not confused with a glass opening, something to read at a restaurant, or a small furry rodent. The definition of *interface* in my slightly out-of-date dictionary reads, "a surface that lies between two parts of matter or space and forms their common boundary." Huh?

Nowadays, though, *interface* has spread beyond computers and is commonly used to refer to the part of a machine that connects with the person using it. Interface designers can be found everywhere there is a need to learn and understand how people use things, then integrate that knowledge into a product or process. Magic Cap was created using the premise of whole person thinking as discussed in the Preface, and every engineering decision along the way was made considering the effect it would have on the life of that whole person.

If you're reading this book from beginning to end, you may have wondered about the order of the chapters. The

chapter before this one expects you to understand and use electronic mail even though it comes before this general explanation of Magic Cap features and concepts. There are two reasons the chapters are presented in this order. First, you shouldn't be bombarded with definitions and explanations before you've even tried to do something real with your communicator. It's much better to get some practical experience first before trying to understand the theory.

Second, you don't have to know all the definitions and explanations to do something with your communicator. If you've finished the *Getting Started* lessons, or if you've just played around with some of the items on the desk, you've learned enough about Magic Cap to know how to move around to different places and how to use the keyboard to send a message or make an appointment. The ability to figure things out easily is a hallmark of a well-designed interface. Now that you have actually done something practical, reading about some of Magic Cap's features in greater detail will be more meaningful to you.

Navigation, Scenes, and Windows

Magic Cap was designed to imitate how and where people work and communicate in the physical world. As mentioned earlier, Magic Cap creates its own world inside your communicator to represent familiar objects and places, and it lets you move from place to place to work with objects. This movement is called *navigation*.

Magic Cap begins by showing you items on a desk. By touching any item, you can use it, send it, open it, or work with it in some way. You can slide some items to move them to other places. You'll also see that some items hop when they're moving to different places, like a new message on its way to the out box. You can leave the

desk and go down the hallway, seeing other rooms in the house. You gain access to those other rooms by opening the door with a touch. You can even leave home to find other buildings and storefronts outside on a street downtown.

Most of the screen (everything between the top bar and the bottom bar) is the current *scene*. The scene focuses your attention on the task you're doing right now by filling the screen. Some scenes correspond to places that you go in Magic Cap, such as the hallway or downtown. Other scenes are close-up views of items that let you work with them. For example, when you touch the clock to set it, the clock zooms up close and fills the screen; what you're seeing is the clock scene, as shown in Figure 3-1.

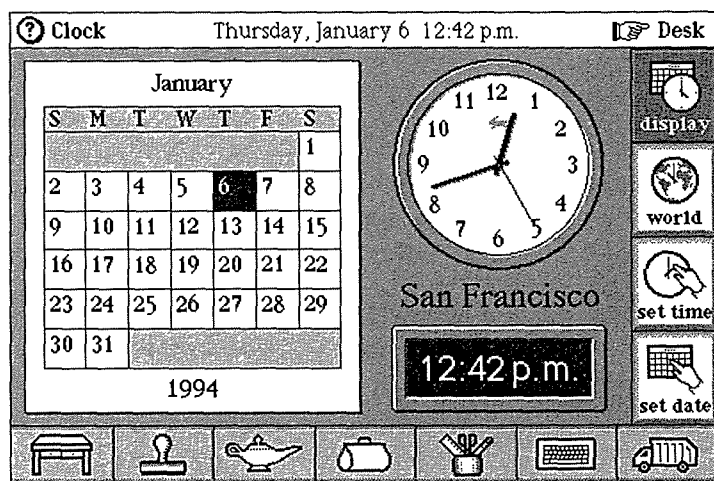


FIGURE 3-1. The clock scene is a close-up view of the clock

Some Magic Cap features must be made available without changing the scene. For example, when you're addressing a new message, you're looking at the message scene, but you also need to see the list of available names so you can choose your addressee. Magic Cap handles this situation by displaying a window that floats above the scene. Windows can perform additional actions or display information necessary to that scene. Figure 3-2 shows the name chooser, an important window that lists names.

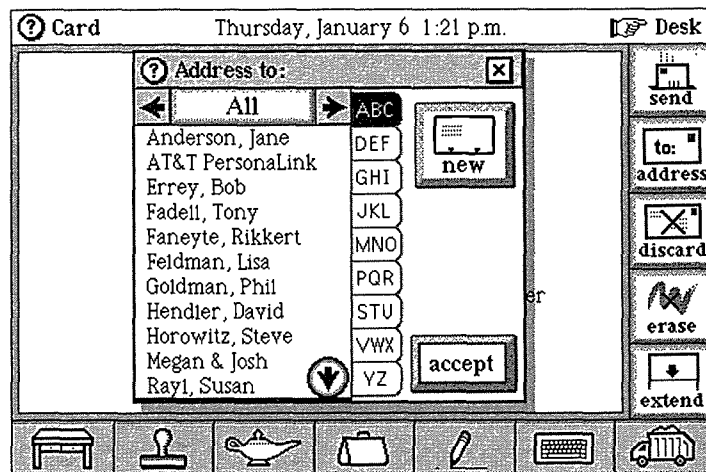


FIGURE 3-2. The name chooser window helps you address a message

The name chooser shows you all the names in the name file and even lets you add new ones when you're writing a new message. Another kind of window is for announcements, which tell you about events or situations that require attention. For example, when you get an urgent message, you'll see a window that announces

the message, as in Figure 3-3. When you see an announcement window, you just read the announcement and then close the window.

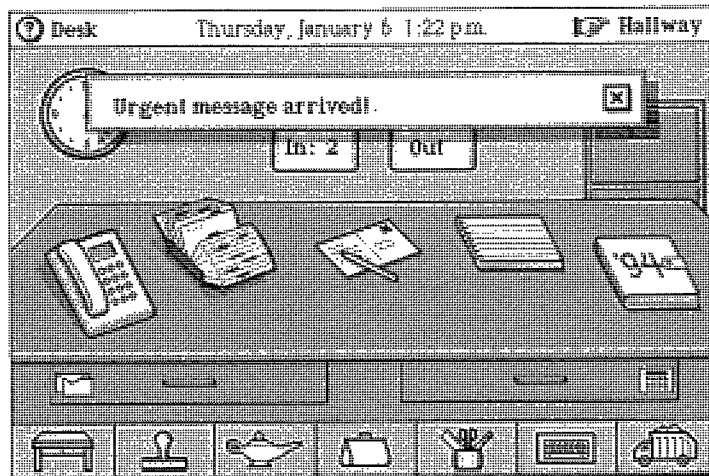



FIGURE 3-3. Announcing the arrival of an urgent message

You'll also see windows with confirmation announcements. These windows are shown when you're about to do something irreversible, like removing a name card from the file. You'll receive a warning in a window with fancy borders and buttons that give you the opportunity to change your mind. For example, when you discard a name card, you'll see a confirmation window like the one shown in Figure 3-4. If you'd rather avoid these confirmation messages, you can turn them off; see Chapter 9 for details.



FIGURE 3-4. Windows ask you to confirm destructive actions

 **Windowitis.** Windows are a fine solution for organizing lots of information on a personal computer screen, but a communicator's screen is so small that piling up lots of windows would quickly create a confusing mess. Magic Cap's solution is using scenes, rather than windows, as the foundation for each related set of features. The alternative to scenes would be stacking up a window each time the user moved around, which would make a big mess. Windows are used sparingly in Magic Cap.

Most windows close automatically when they're no longer needed. Many windows have a button labeled *accept* that lets you signal that you're done with it. When you tap *accept*, the window closes. Windows also have an x with a box around it in the upper-right corner. Touching this box closes the window. Because Magic Cap tries

to help you work faster, it often closes windows for you automatically, as when you switch scenes.

Magic Cap has another user interface goodie called a *choice box*. This one displays an option from a list and lets you pick what you want by touching arrows to move through the options one at a time. If you're impatient and you want to see all your choices at once, you can tap the label-like area in the middle that's surrounded by arrows, then touch the one you want. You already saw choice boxes in action when you selected different ways to send your messages in Chapter 2.

Smart Integration and Consistency

One of the main elements of Magic Cap is the tight integration of information across various scenes. If you find that you need to add a name to your list of contacts, you can do so without having to go to the name file scene. You can make new name cards while using the phone or the datebook, while making a new message, or from the name file itself. You can make a phone call while in any scene by using the *contact* button in the lamp, and then continue the call while you move on to other scenes. If you make an appointment for a meeting in the datebook, you can send a message inviting the participants without ever closing the datebook.

Magic Cap's designers worked hard to make features consistent and predictable. Things that look the same should behave the same, or users get confused and frustrated. Every time you learn about a feature, you will find that not only is its use consistent throughout Magic Cap, but you'll probably be able to guess about other elements of the feature by applying information you've already learned about other similar features.

Sound and Visual Effects

As soon as you start using your communicator, you'll notice that Magic Cap uses sound effects. These sounds help reinforce your actions as you take them, confirming that something has happened. Although most actions have accompanying visual effects, the sounds are a subtle reinforcement that you're getting what you expect. You hear a high-pitched sound when you make a new message and a pop when you touch the x to close a window. You hear the Magic sound at the beginning of the *Getting Started* lessons. When you discard something, there's a deep clunk. When you want to slide an object into the tote bag, you can be sure it's there when you hear the slurp sound.

As you use your communicator, you'll come to recognize the sounds that accompany familiar actions. Of course, you'll use your communicator in some settings that aren't appropriate for sounds, so you can easily turn down the volume. Because Magic Cap lets you customize your environment, you can also change the sound effects that are used for confirming actions.

In addition to sound effects, Magic Cap uses visual effects to confirm your actions. When you touch something, you can tell you hit the right spot because of the cross hair you see. Some objects highlight by displaying a kind of starburst effect that looks like cartoon motion lines. A simpler highlight effect, the one used by most buttons, is to invert the image—the background becomes lighter while the image becomes darker. While these highlighting effects sound strange when described, they're so intuitive that you'll probably understand their meaning right away without having to read about them first.

Magic Cap is filled with lots of other visual effects and animations that confirm your actions. In addition to mail that hops to the out box when it's sent, as already mentioned, discarded items often hop to the trash, filed mail hops into the file cabinet, and books hop onto shelves when you're done with them.

Top of the Screen

The very top and bottom of the screen contain things that you can always count on, no matter what scene you're in or what you're doing. The top provides various pieces of information about the scene and the communicator itself. The bottom has buttons that perform vital actions. This section presents a closer look at all the things that are available at the top and bottom of the screen.

The top-left corner of the screen always has the name of the current scene. There's usually a circled question mark next to the scene name that you can touch to get a description of that scene and instructions for the actions you can take there. The top-right corner of the screen always has a pointing hand and the name of a related scene. You can touch this hand or scene name to go there. Because tapping the hand or scene name often takes you back to the last scene you were in, this move is called *stepping back*.

As you become an expert navigator, you'll take advantage of a helpful shortcut built into the step-back hand. If you hold down the option key and touch the hand, you'll see a list of everywhere you've been (see Figure 3-5). You can get back to any scene instantly by touching its name in that list.

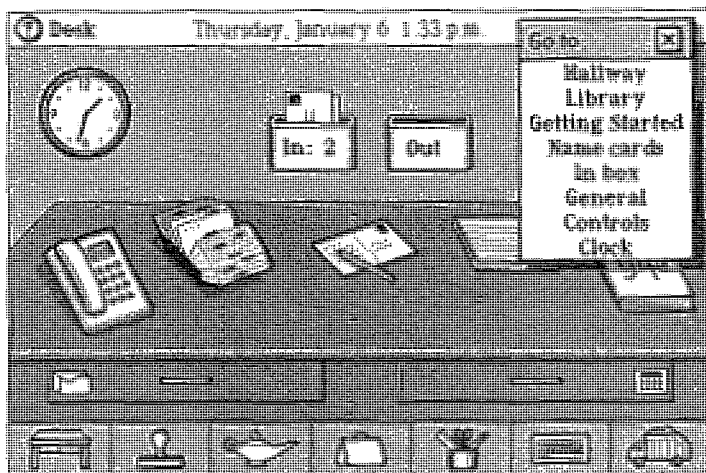


FIGURE 3-5. You can step back to any scene you've visited

In between the important areas at the left and right, the top center of the screen shows today's day and date, the current time, and the power level of your communicator's main battery. You can choose to display or hide any of these three items; details are in Chapter 9. If your communicator has some action that is ongoing, such as a message being sent or a phone call in progress, the top of the screen will also contain small images to remind you about those actions.

Bottom of the Screen

The bottom of the screen contains seven buttons that are also always visible, no matter where you are or what you're doing. Take a look at Figure 3-1, or virtually any other screen, to see the buttons at the bottom of the screen. The button in the bottom left corner has the image of a desk, and whether you're wandering around

aimlessly exploring Magic Cap or you remember that you need to check your datebook while you're browsing in the library, you can always touch this desk button to return quickly to the familiarity of your desk. Because the desk button is always there and it always takes you to the desk, you know that whatever you're doing in Magic Cap, you're always just one touch away from a comfortable scene.

Next along the bottom of the screen is the stamper, a button with an image of a rubber stamp. The stamper holds drawers filled with stamps, animated cartoons, sounds, and even songs that you can put on messages and other pages (see Figure 3-6). Some stamps in the stamper also perform actions or assign attributes, like the phone number stamps in the name file, or the urgent stamp on a message. Each scene can add its own custom drawers, like the signature stamps that are seen only when creating a new message, or the label drawers in the name cards scene.

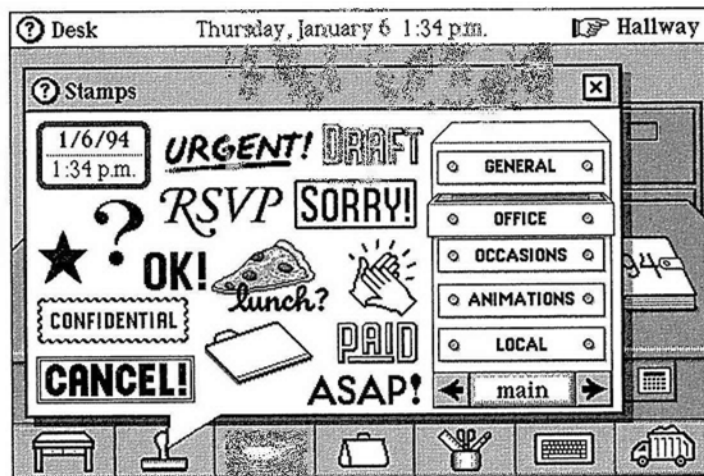


FIGURE 3-6. The stamper holds drawers of stamps

The lamp has buttons that help you communicate and do other important actions. Because the lamp is always there, you can perform these actions no matter what scene you're in (see Figure 3-7). The lamp includes *mail* and *fax* buttons to send the page you're seeing. You will also find the infrared *beam* button in the lamp, which you may remember from Chapter 2.

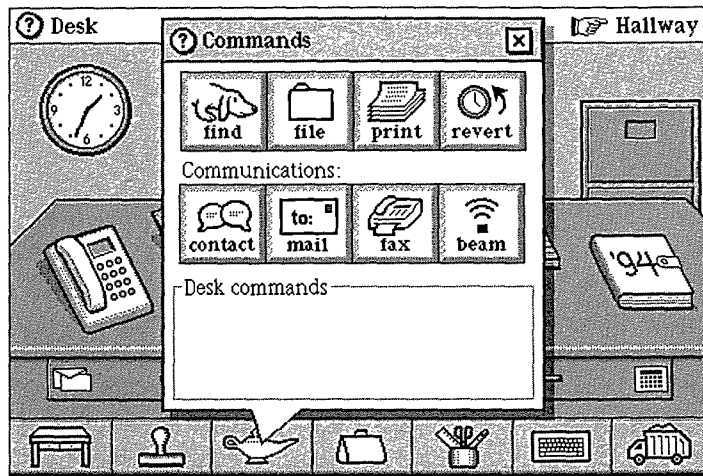


FIGURE 3-7. The lamp contains buttons that are always available

Another button in the lamp is *contact*, kind of a short-cut path to various means of communicating with someone. If you're in the middle of writing a letter to a client and you need to call your assistant to verify some information before you send it, you can touch *contact*, which then lets you use the phone or electronic mail, or you can quickly look up a name card without changing the scene. Like all other Magic Cap shortcuts, you don't need to know about or ever even use it; it's just another way to get there.

There are some other buttons in the lamp that perform general actions that aren't necessarily communication-related. The *find* button helps you track down information anywhere in Magic Cap. For example, after you've entered some appointments in the datebook (much more about that in the next chapter), you may want to go back and find one meeting in particular. If you don't remember the meeting's date, you can use *find* to search for it.

Touching *find* opens a window and the keyboard for you to type in what you're looking for. You can instruct it where to search and then unleash an animated basset hound (no, really!) to find it for you. The dog will search through the items on your desk until he sniffs it out in your datebook, opening to the correct date. When he's done, he hops back into the lamp until you call him again. If you just can't deal with an animated dog looking for your stuff, there's an option in the *Find* window that performs the search without showing the dog.



Sniffy the Wonder Dog. The dog that searches for your stuff is probably one of the most controversial aspects of Magic Cap. The designers of Magic Cap have a sense of humor but would never allow silliness to get in the way of the best possible experience for the user. People expect a dog to be able to search and find things, but not to be able to show human judgment; this expectation matches what happens when you search in Magic Cap, so the designers decided to use a dog. In practice, users either love or hate the dog, with very little gray area.

There is a *file* button in the lamp to move things into the file cabinet and onto memory cards; you can find out much more about filing in Chapter 8. The lamp also has a *print* button that lets you print information when your

communicator is connected to a printer directly or through a personal computer.

The final button in the lamp is called *revert*, and it's there to make a last-ditch effort to retrieve something you thought was irretrievable. The *revert* button removes any changes you made to the scene before your current visit to it, saving you if you've done something you want to reverse.

In addition to these buttons that are always available, each scene can add its own buttons in the lamp that work only inside that scene. For example, when you're looking at a message, the lamp includes a button that measures how much the card "weighs" in bytes, giving you an idea about how long it will take to send. The lamp also includes rules that set policies for each scene. For example, the datebook has rules that tell how to remind you when an appointment is coming up and when the communicator should throw away old appointments.

Next to the lamp along the bottom of the screen is the image of a tote bag (Magic Cap's designers considered using a suitcase, a satchel, and even a pocket before settling on a tote bag). Think of the tote bag as the place where you can put something if you need to carry it from one scene to another. When you put something in the bag, you'll see the bag bulging a bit, a graphic reminder that there's something in there. When you take everything out of the bag, it shrinks back to normal size until the next time you need to carry something.

The next item along the bottom of the screen is the tool holder, which represents all the different tools you can use for writing and drawing. This is where you can choose from among different pens that you use for writing on the screen, when you make a new message. There are even some magical pens that draw all sorts of

perfectly proportioned shapes, and others that draw lines that are always straight. Chapter 7 goes into greater detail about how to use the tools.

The next item is the button that opens Magic Cap's on-screen keyboard. You already know quite a bit about the keyboard if you followed the *Getting Started* lessons or if you've done any typing in any other scene. You've probably noticed some of the features that help you speed up your typing, such as smart capitalization and automatic completion of words. The keyboard lesson in *Getting Started* shows how to switch the keyboard from letters to numbers.


If you hold down the option key and touch this keyboard button, you'll get the extended keyboard, which lets you type letters, numbers, and special symbols like accented letters (Reneé) and international symbols (¡hola!). The extended keyboard also has a label maker that hangs off the top-right side. As you touch each key, the label prints out one letter at a time. When you're done, you can tear off the label and it becomes a text coupon that you can use to change an item's name. This is the process we used in Chapter 2 to name the custom stationery.

There are a few other keyboard goodies you haven't used yet, like the *expand* key. There will be more details about the *expand* key in Chapter 9, but briefly, you can use it to ask Magic Cap to guess the right word when you've only typed the first few letters. You can also set up abbreviations so you can touch a minimum number of keys while you're typing something, and then expand it to the full entry with the touch of one key.

The final button on the bottom of the screen shows the image of a garbage truck. As you might guess, this is where you throw things away when you don't want them

any more. You can trash an item by sliding it into the truck; it makes a slurping sound when you drop it in there. As a precaution, the last few items you throw away are stored in the trash (its image changes to show that it has something inside) until you empty the trash by opening the truck and touching its *empty* button. If you're really desperate to recover something you've tossed, refer back to the explanation of *revert* for more details on whether it's really gone.

You can hold down the option key when you touch some of the buttons on the bottom of the screen to make them do other things. When you option-touch the keyboard, you get the extended keyboard and label maker. Option-touching the lamp lets you set the volume or go directly to the control panel. If you option-touch the stamper or tool holder, the window will open to the same setting it had when you last closed it. For example, if you have the *faces* drawer of stamps open and you choose a face stamp, and then decide you want another face, you can option-touch the stamper to go directly to that drawer.

 **In the Corner Pocket.** The corners of the screen are precious real estate. Because you can figure out where they are without really looking, it's a good design idea to put very commonly used things there, and the desk button and the trash qualify. After you use Magic Cap for a while, you won't even think about the desk button—you'll just reach for the lower-left corner when you want to feel well grounded. Similarly, you'll get used to sliding things to the extreme lower right when you want to throw them away—and you won't even have to look to make sure they've reached the trash.

Construction

The engineers at General Magic had such a good time constructing the software that it seems they wanted you to be able to share in the fun. They realized that they couldn't possibly include everything that every user of a Magic Cap communicator might want to do, so they built a way for people to construct things on their own. The hallway includes a control panel that lets you put your communicator into *construction mode*. When you turn on construction mode, you'll get a magic hat that has an endless supply of goodies for building your own scenes: buttons, switches, choice boxes, sounds, borders, and lots more. If you just can't wait to find out about construction, check Chapter 10 for more details.

Summary

Magic Cap is designed to let you communicate easily with a minimum of instruction and background knowledge. The main scene is a desk, with several familiar items that help you communicate. You can go down a hallway to visit other rooms, like the library or the store-room. You can go beyond the hallway and wind up downtown, where you'll find AT&T's building and other storefronts. As services become available, downtown will have more stores selling merchandise and services that will provide you with customized information.

The space between the top and the bottom of the screen is filled by the current scene. You'll see windows that provide more information, announcements, or confirmation buttons that let you double-check before you do something destructive.

The information you enter into your communicator is available in all applicable scenes. For example, you can enter a new name while you're using the phone, or send a message while you're working in the datebook. In addition, features behave predictably and consistently throughout Magic Cap. Magic Cap uses sound and visual effects as cues to help confirm your action when you touch the screen.

The top of the screen displays information needed for navigation, including the name of the scene, the name of a related scene that you can go to, and possibly the time and date and battery level.

The bottom of the screen has buttons that are available in every scene. The stamper contains drawers of stamps that can be used for decoration or to assign attributes necessary for sending or filing. The lamp holds commands that can be used anywhere in Magic Cap as well as rules that set policies in every scene. The tote bag is used to carry items from place to place, and it alternately bulges and shrinks, depending on what's inside. The tool holder contains various implements for writing and drawing.

The keyboard is used for typing, your main way of getting information into Magic Cap. The keyboard can type letters, numbers, and special symbols. The keyboard can also type text onto a label maker, producing coupons that let you give names to objects on the screen. You can throw things away by sliding them into the garbage truck, making the image change from an empty truck to a full one. You can retrieve recently tossed items from the trash until you empty it.

Magic Cap includes construction mode, a way for advanced users to put together their own buttons, switches, and other fancy features.

Chapter 4



Datebook

The Last Datebook You'll Need

Before I started using Magic Cap, I already knew how much I depended on my datebook to keep my life running smoothly. I noted business meetings, doctor appointments, work deadlines, carpool schedules, and the delivery schedule for bottled water. You probably have lots of other reasons for trusting your calendar. Because datebooks are such personal things, appointments are often written cryptically in personalized secret code, with start or stop times that don't necessarily match the lines they're written on, or directions to places that seem to be on other planets. Magic Cap makes it easy (yes, even fun) to enter appointments in a datebook so they'll be descriptive and complete, yet still personal.

The folks at General Magic built lots of shortcuts into Magic Cap's datebook to make entering appointments a snap. Like everything else in Magic Cap, the datebook and its appointments are easily customizable, and you can arrange and add items that fit your needs. When you enter a new appointment, lists of choices are offered for every part of an entry. There are several different kinds of appointment types with various descriptions and options. You can add notes, stamps, and alarms to any kind of appointment. Each kind of appointment also has its own identifying image, which you can change if you want to.

When you shop for an old-fashioned datebook in a stationery store, you can choose the style that best fits how you work—seeing a day, week, or month at a time. You might like to see a week at a time if you have enough entries to need more space than a monthly book contains. If you're really busy, you might have to put up with the bulk of a daily appointment book. Magic Cap lets you switch between all of those options with a touch.

In Magic Cap, you can enter new appointments while looking at a day, week, month, or even year. Switching between different views is easy, and appointments entered in one view show up in the others, of course. Figure 4-1 shows the datebook's day view. The first thing many people do when they get a new datebook is go through and enter important dates—birthdays, work events, anniversaries, vacations. Let's do that now with Magic Cap.

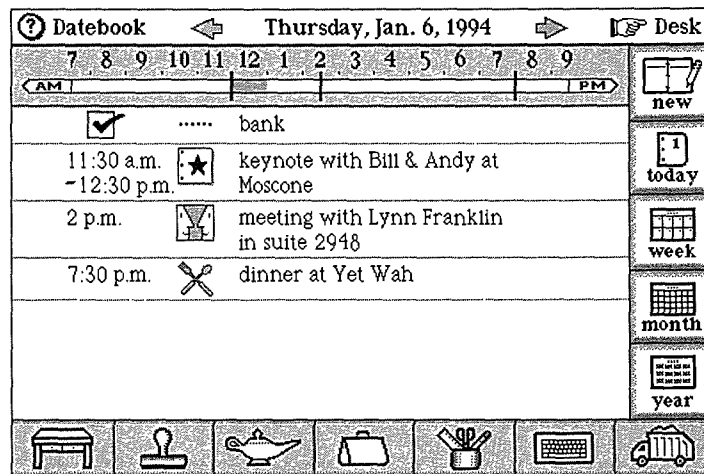


FIGURE 4-1. The datebook shows information about today's appointments

When you touch the datebook on the desk, Magic Cap automatically opens the datebook to today's page. When you touch *year*, Magic Cap displays the whole year, with today's date highlighted. The top of the screen shows the year, in case you've forgotten, and you can even wander off and look at other years if you want. When you touch a month, the month's view zooms open. As you might expect, the name of the month is also displayed at the top of the screen with arrows that move to the previous or following month. Figure 4-2 shows an example of a month view.

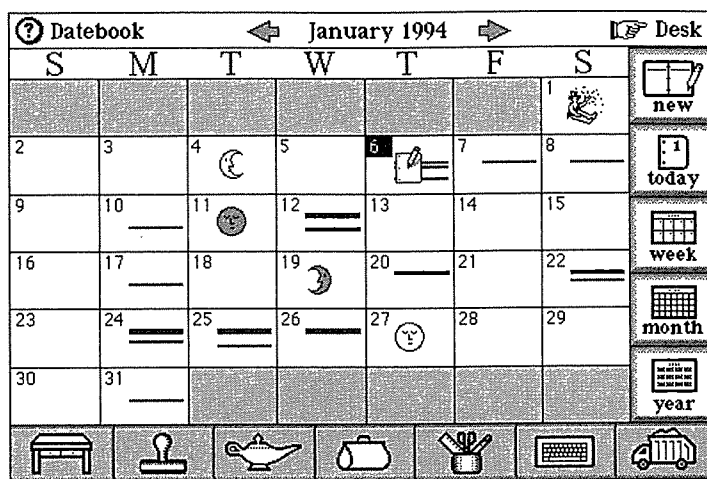


FIGURE 4-2. Month view in the datebook

Like most paper calendars, Magic Cap comes with some holidays already entered, both by name and by picture. You can change a holiday's picture or get rid of the occasion completely if you're a real curmudgeon. The datebook also includes appointments for turning daylight savings time on and off in the communicator's built-in clock. When the days arrive for starting or ending

daylight savings, the clock is automatically adjusted, untouched by human hands.

Holidays and daylight savings time aren't the only special days the datebook knows. For example, the datebook offers a friendly reminder of what you need to do on April 15. Finally, you can set the datebook to have the month view show the phases of the moon if you're lunarly inclined.

Entering Appointments

When you tap *new*, Magic Cap offers several choices for appointment types. You can tap *birthday* (which is represented by a little piece of cake with a candle) to see the month and date, then pick the birthday person from the name file. If you don't have that person in the file yet, you can make a new name card right on the spot without having to go to the name file. When you tap *accept*, the new person is filled in as the birthday boy or girl, and there's a name card for the new person, who also becomes the current contact. If you want, you can attach notes to the birthday entry about how old the person is or what kind of presents you should buy. When you tap *save*, the birthday goes into the calendar. Because it's a birthday, the datebook automatically repeats that entry for each following year.

There's probably no better way to break in your new datebook than by adding your mother's birthday. Let's assume her birthday is December 24. After you've added Mom, you can enter your spouse's birthday; let's say it happens to be two days later, on December 26. Here's a cool shortcut for doing that: Hold down the option key and slide Mom's birthday entry to the tote bag. By holding down the option key while sliding, Magic Cap makes a copy of the birthday while leaving the original birthday in its place. Figure 4-3 demonstrates this.



FIGURE 4-3. Option-slide the birthday to the tote bag to copy it

Next, tap twice on the right arrow at the top of the screen to move to December 26, watching the Christmas tree go by as December 25 passes. The bulging tote bag is a reminder that a copy of the birthday is still inside. When you slide the birthday out of the tote bag, it pops into place. It still has Mom's name, though, so you can tap it to open, and then tap *who* and give the birthday your spouse's name. Not that you'd ever forget Mom's or your honey's birthday, of course.

The next appointment to enter is a favorite anniversary, such as your wedding. Start by tapping *year*, then tap to get to the right month and date. Tap *new* to make a new appointment, then tap *special day*. The special day appointment includes a *what* button; when you tap it, you'll see a list that includes *business trip*, *holiday*, and the one you're looking for: *anniversary*. (It also includes *hibernation*, an appointment that might be more

appropriate for a bear's datebook.) After picking *anniversary*, just tap *save*, and the special day is set. You can see Magic Cap's list of special days in Figure 4-4.

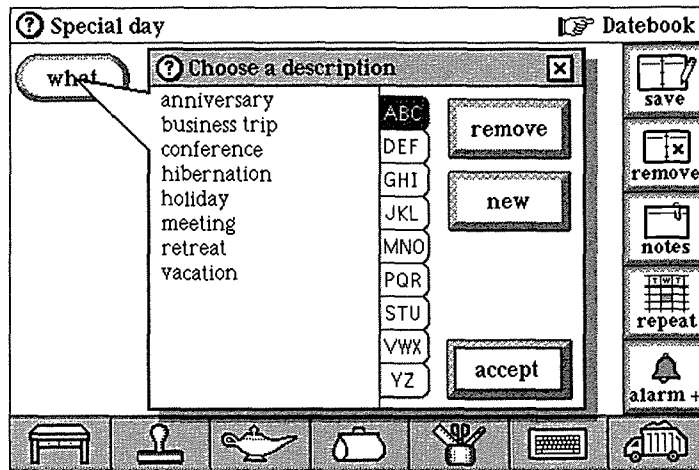



FIGURE 4-4. List of special days

 **Making a List.** The datebook has several situations where you get to pick an item from a list, as when you set a special day appointment. In most of these lists, you can pick one of the built-in options, or you can add your own, usually by tapping the *new* button. When you add a new choice, Magic Cap is smart enough to remember the new option and will list it every time that list appears in the future.

If you want to add a new option quickly and bypass the list of items, you can tap the keyboard image at the bottom of the screen. The keyboard opens and you can type the new option immediately. Of course, the new option you create will be added to the list for future

appointments. You can also remove options that you think you'll never use. If hibernation isn't in your plans, just tap it and *remove*; it won't bother you again.

Recurring Appointments

After adding all the birthdays and anniversaries to the datebook, you're ready to put in some of the other appointments that fill your time. You can start by adding the items that repeat, like staff meetings (twice a week) and school meetings (once a month). To make appointments that repeat, you start out by entering the first of the repeating events.

You'll enter appointments for staff meetings at the same time every Monday and Thursday. To put them in the datebook, you can just make one appointment for Monday and one for Thursday, setting each appointment to repeat every week. Go to the next Monday, tap *new*, and then tap *general purpose*, which is an appointment type that lets you pick the date, start and end times, what, where, and who. When you tap *what* and *where*, lists of choices appear; as usual, you can add your own. If you don't know (or don't care) about any of the information, such as location or end time, you can just leave it blank. (see Figure 4-5).

After you've filled in what you know, you're ready to set the appointment to repeat every week. To turn on the repeat feature, tap *repeat*. You can flip through the repeat choices until you see the right one: *repeat weekly*. When you save the appointment, it's entered for every Monday for the next year.

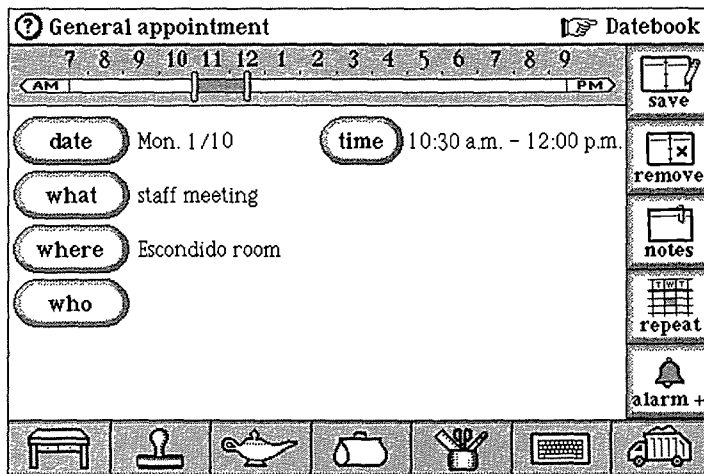


FIGURE 4-5. General appointment entry screen

Now you can set up the same appointment for every Thursday. You could just move to Thursday and enter the same appointment again, but let's try a shortcut. By holding down the option key, you can slide a copy of the appointment you just made into the tote bag. Then, move three days ahead, to Thursday. Slide the appointment copy out of the tote bag and just drop it on the day. That's it! Not only does it set up the new appointment, but Magic Cap is also smart enough to figure out that you want it to repeat every Thursday, since the original appointment was set to repeat every week.

School Daze

Now let's enter another repeating appointment, a school meeting. Once a month, on the third Wednesday, we'll enter a class meeting for the third grade class at Christa McAuliffe School. The meeting is always at 7 P.M. at school, and like many meetings, you never know when it will end. When you enter the appointment in the

datebook, fill in 7:00 as the start time and just don't bother with an end time. The datebook's designers understood that lots of appointments and meetings don't have a scheduled ending time, so you don't have to pretend that there is one.

Because this meeting happens on the third Wednesday of every month, you'd hope to use the datebook's repeating appointments feature to add the meeting to each month's calendar automatically. By using the *repeat monthly by day* option, the datebook understands that this meeting is always on the third Wednesday and schedules it there for future months. It suggests repeating the meeting for a year in the future, which you can adjust to May instead. In addition to monthly appointments that repeat by day, the datebook also lets you set up appointments that fall on the same date every month.

In this way, you can enter the meeting for a single Wednesday night and easily have it repeated for the whole school year. The subsequent meetings are each individual appointments, so you can change them or remove them as necessary without affecting any others. This makes the datebook flexible enough to handle the inevitable postponed or canceled meeting without messing up all the others.

Engraved Invitations

Because I work freelance, timely scheduling of meetings has a direct impact on my income. If you're like most modern communication cowpokes, telephone tag drives you crazy when you're trying to get together with someone. To help with this problem, Magic Cap includes a slick system for setting up meetings that is centered around the datebook. We'll use this system to get a meeting going.

To set up the meeting, tap *new*, then *meeting* to get the appointment entry scene. As with other kinds of appointments, you can set the date and time. The *where* button produces a list of possible locations for the meeting. As always, you can choose one of those listed, or add your own. You could choose *my home* or *my office*, but it's always more fun to have meetings at restaurants—after all, that means you get to eat.

When you're ready to choose the participants in the meeting, Magic Cap shows off how it integrates all the information in the communicator. When you tap *who*, you'll see a window listing the people you can choose for the meeting (see Figure 4-6). This isn't just any list of people, though—it's the same names that appear in the name file. Magic Cap uses the name file as the center for any kind of communication, including meetings, messages, phone calls, and faxes.

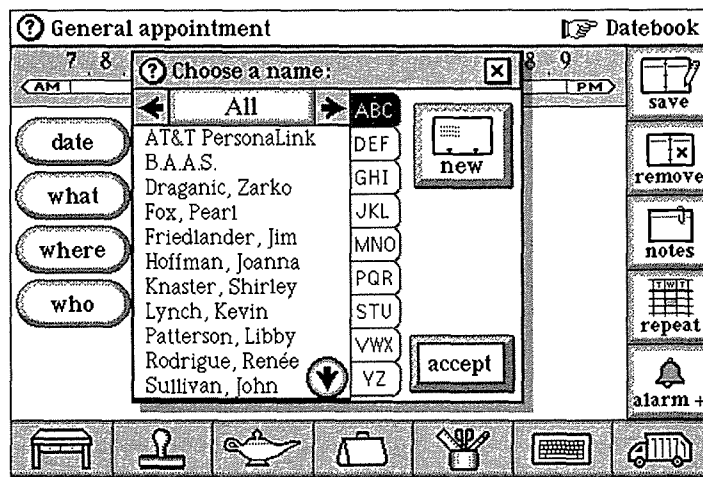


FIGURE 4-6. The name chooser is used to select names for an appointment

What if you need to make an appointment with someone who's not listed in the name file? The window that lists names includes a *new* button. You can tap that button to enter a new name card quickly, and you can then add that new name to the meeting. You even have access to the name file from the datebook, and you can add any name to the file for future lunch possibilities.

We'll set this up as a meeting at Seafood Sam's with three people, Martha, Steve, and Keith, and because this is a brand-new communicator, none of them is listed in the name file yet. By adding them now, you get to list them in the appointment, and you'll also have them in the name file for any future communication. To enter the new names, tap *new* and add the information for each person's name card in turn.



Choosing Names. The window that lets you pick from the name file is called the name chooser. It has a few other features that are especially useful as the list of names grows larger. There are tabs running down the right side of the list that let you jump quickly to any set of three letters—for example, if you tap the *STU* tab, the name chooser will make sure that the names starting with S (and T and U, if there's room) are on the screen.

The name chooser also includes a choice box that lets you see all the names, or only certain sets of names if you want. You can look at only the people, or just the companies, or you can choose to see only the members of a single group. One other note about the name chooser: The groups are shown in boldface so that you can spot them easily.

After you've entered the three names into the name file (without ever leaving the datebook and having to

open the name file), you're ready to add the three new folks to the meeting. Magic Cap remembers that Keith was the last name card entered, so Keith becomes the current contact—Magic Cap will have Keith's name selected when the name chooser opens. If that's who you want, you can just tap *accept*, or you can choose another name, which will then become the current contact.

The current contact feature is handy here, because you do want Keith in the meeting. When you tap *accept*, the window closes and Keith is added to the meeting. This is OK, but now you have to tap *who* again to add Steve and Martha to the meeting. To make things a bit faster, Magic Cap defines this nice shortcut: If you hold down the option key while tapping *accept*, the chosen person is added to the meeting, but the list of names stays open. This little trick lets you pick all the meeting participants quickly. That's a perfect example of an option-key shortcut: If you know about it, you can work a little more quickly; if you don't, you can accomplish the same task with a more obvious (but longer) list of steps.

As the last step in setting up the meeting, you can use the priority and status features to set the meeting as tentative, but high priority. The tentative setting adds a question mark on the meeting's display in the day view and the high priority is designated by an exclamation point.

When you're done adding the three people to the meeting and putting in all the details, all three names appear next to *who* on the appointment entry screen, as shown in Figure 4-7. Of course, if you're having a typically busy day, you may find that you won't be next to a telephone for more than 10 minutes at a time, so trying to reach three people, confirm the details, and possibly change them would be tough. The people you need to talk to might be somewhere on the road during a long com-

mute, or maybe staying at a hotel and having meetings of their own in several cities. It would be great if this new communicator could help you get in touch with everyone.

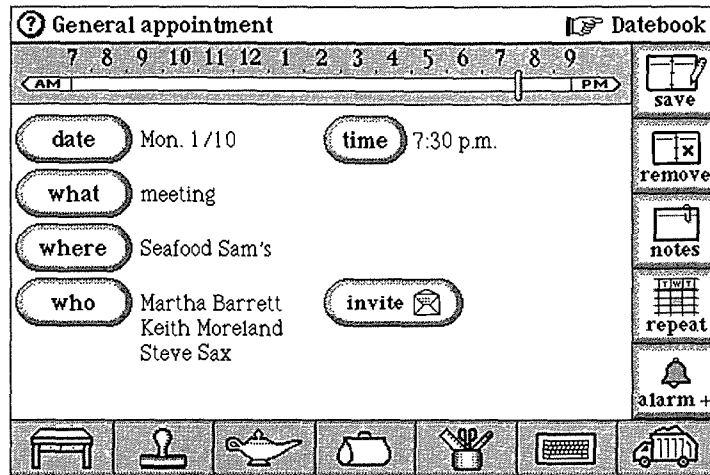


FIGURE 4-7. Meeting being entered with multiple participants

Here's how you can do it: Once you've chosen the meeting participants, a new button labeled *invite* appears on the appointment entry, with a suggestive little picture of an envelope. When you tap *invite*, Magic Cap automatically presents a new postcard that spells out the details of the meeting, addresses it to the people you want at the meeting, and asks for their response. The message uses the details of date, time, and place that you completed earlier, and even attaches any notes that you've entered along with the meeting information.

Not only does Magic Cap send a postcard separately to each invitee, but it also individually addresses each one, taking a good guess at the best way to contact each

person (that is, via PersonaLink, fax, America Online, or whatever). Best of all, Magic Cap makes handy *yes* and *no* buttons for the meeting invitees to use when responding. Figure 4-8 shows the postcard that the datebook creates.

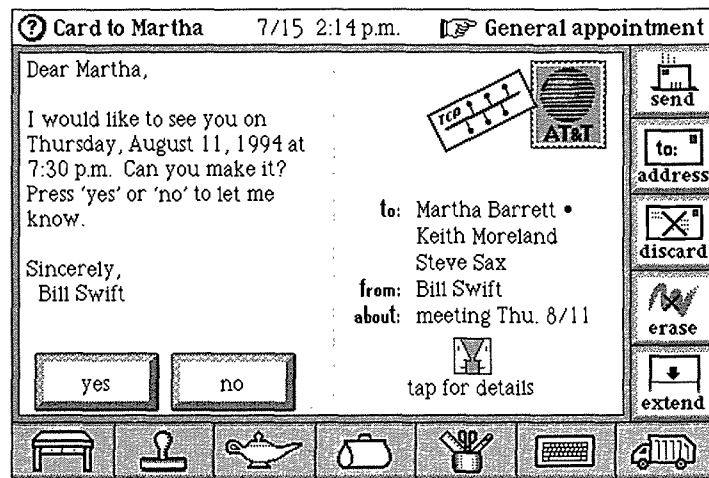


FIGURE 4-8. Automatically generated invitation for meeting

The postcard that Magic Cap creates is all written (very literately, I might add), addressed, and ready to go. If there's something about the message that you don't like, you can easily change it: You can write something with one of the writing tools, change the text by typing, or add a stamp or two. You can also change something more substantive, like the way that the card will be sent to any or all of the invitees.

You can mess around with the message for as long as you like. When you're finally happy with the message, you can send it. When you *send*, the message card hops into the out box, and a progress window shows what's going on as the message is sent. When the meeting

invitees get their messages, there's a copy of the appointment attached for them to examine. Each invitee can simply tap the *yes* or *no* button on the message card to respond.

If an invitee answers *yes*, several intelligent things happen. First, Magic Cap automatically creates and sends a return message telling you that that person can come, adding a thumbs-up stamp to the return card (see Figure 4-9). Next, the attached appointment that you created pops into place in the invitee's Magic Cap datebook, substituting your name for the invitee's (that is, it doesn't give Martha an appointment that says "meeting with Martha").

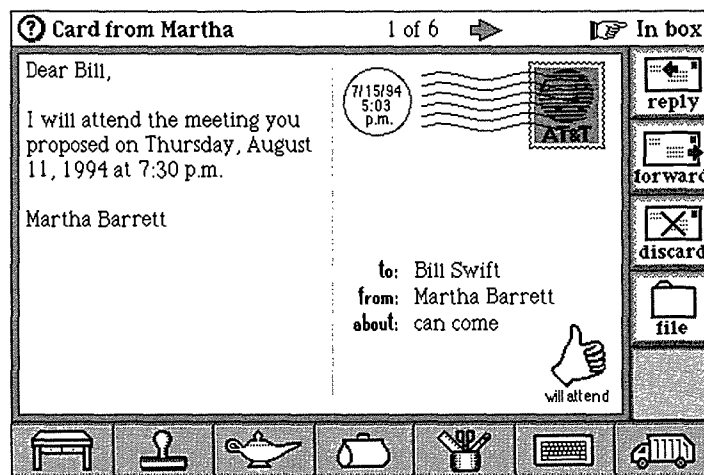


FIGURE 4-9. RSVP card for meeting with thumbs-up stamp

If all the invitees say that they're coming, the tentative setting for the meeting in the datebook changes to confirmed, and the tentative meeting's question mark is replaced with the regular face-to-face image that represents a confirmed meeting.

If an invitee taps *no*, Magic Cap throws away the invitee's copy of the appointment and then creates and sends a response telling you that that invitee can't come, stamping thumbs down on the response message. Of course, when you get the negative RSVP, the meeting stays tentative instead of being confirmed. If only one person responds yes, the meeting is confirmed, as Magic Cap errs on the side of caution, figuring that you'll want to go ahead with the meeting even if only one other person can make it.

This magical scenario of automatic response works great when both the sender and the recipient have Magic Cap, but what if one or more of the meeting invitees is using some other brand of communicator? If a message is received by a classic kind of mail, such as AppleLink or CompuServe, that doesn't know about buttons or datebooks, the automatic response and scheduling stuff obviously can't work—the invitee will have to send a response back manually. Magic Cap's electronic mail is smart enough to work wonderfully when communicating with advanced Telescript-based services like PersonaLink, and it works as well as possible when connecting to conventional services.

By using Magic Cap, you and the meeting's invitees don't need to rely on voice mail, answering machines, or hotel operators—the messages can be received and answered when it's convenient for the recipient. You don't even have to wait until the messages are sent before you can use other features, because Magic Cap lets you keep on working on other things while it's sending messages.

Also, the meeting invitees don't have to have Magic Cap to get the messages. As long as an invitee has one of the many electronic mail services that you can reach, or even just a fax machine, Magic Cap can get the word out. Now you can look forward to that dinner with your associates!

Integration

After all this inviting and responding, the appointment shows up in the datebook's day view as a *meeting with Martha and others* (it actually uses those human-sounding words) at Seafood Sam's at 7:30 P.M. This little exercise demonstrates one of the best features of Magic Cap—the amazingly tight integration of different kinds of information and communication. You opened the datebook and made an appointment, obviously something a datebook should be able to do. But while you were there, you added three people to the name file without having to actually open the name file. Then, you sent electronic mail messages to those people without having to go to the mail scene. This well-designed integration lets you get work done smoothly and quickly without having to fuss around with details of getting from one place to another.

To Do Lists

You probably don't have the luxury of being able to concentrate on just one thing at a time in your life—if you did, you wouldn't be interested in Magic Cap. Most people have a work life, a home life, a social life, and maybe a few other lives, and they need to remember different things for each one. Work life includes deadlines, schedules, and meetings; social life means sports, concerts, and parties; home life involves family, friends, personal finances, errands, and relaxation.

An old-fashioned appointment book is barely adequate for writing down where you have to be and when, and it usually has no place at all for keeping track of what you have to do. Many people like to make lists of what they're supposed to do and what they need to buy, and of course, they usually leave those lists at home when they need

them most. Magic Cap provides help in remembering what needs to be done without having to buy cases of sticky notes.

Magic Cap's datebook lets you build and maintain lists of tasks that you have to do. When you're in the datebook, tap *new*, then *to do*, to enter information about a new task to be done. Let's make a task that's a reminder to go to the bank during today's lunch hour to deposit the checks that Junior got for his birthday from his grandparents. When you tap *what*, there's a list of descriptions for the task, and it's the same list that the datebook always shows when you have to fill in a description for any appointment. There's nothing in the descriptions about going to the bank, so you can type in a new description, *bank*. As *bank* is entered for the task, this new description is added to the list that will be offered whenever you enter a new appointment.

After you fill in the description, you get to choose the starting date and the deadline for the task. The new task suggests today as the start date, but you can tap *start* to see the date chooser and then use the arrows and the calendar page to move to a different date. The new task suggests one year from today for the deadline, but that's a little too long for this one; surely your mother-in-law would call long before the year was up, wondering why her check hadn't cleared. Again, by tapping *deadline*, it's easy enough to change the month, year, and date with the arrows at the bottom of the date chooser. Let's set the deadline for tomorrow. You can see the finished product in Figure 4-10.

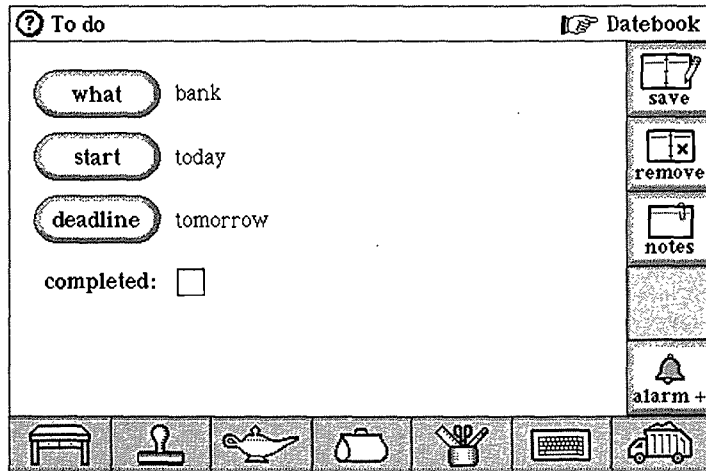


FIGURE 4-10. To do appointment entry

It might also be a good idea to add notes to remind yourself which account to put the checks in, plus reminders for other banking transactions you might want to make, like transferring money from the savings to the checking account to cover that check you wrote for car insurance. You can do this by tapping *notes* on the right to open the notes window. Then, write the notes, or tap the keyboard and type the notes if you're one of the many people who can't always read their own writing.

As you finish the notes and close the notes window, you can review the details of the task to make sure they're right. The task reveals another magical touch that's a welcome relief from the digital mindset of computers: The starting date for the task reads *today*, rather than giving today's date. (The datebook is also on a first-name basis with *tomorrow* and *yesterday*, but that's it—no *day after tomorrow* or *a month from next Tuesday*.) Like so much of Magic Cap, the datebook knows that most humans would use the word *today* when referring to an appointment that takes place on the current day.

You can make the same kind of entries for other errands and tasks, such as a trip to a customer's office to check on an order, a visit to the city government building to renew a business license, or a quick stop at the grocery store to get stuff for dinner. When a task is all filled in, you can tap *save* to put it into your datebook. The task is added to your list of appointments for the day, which you can see in Figure 4-11.

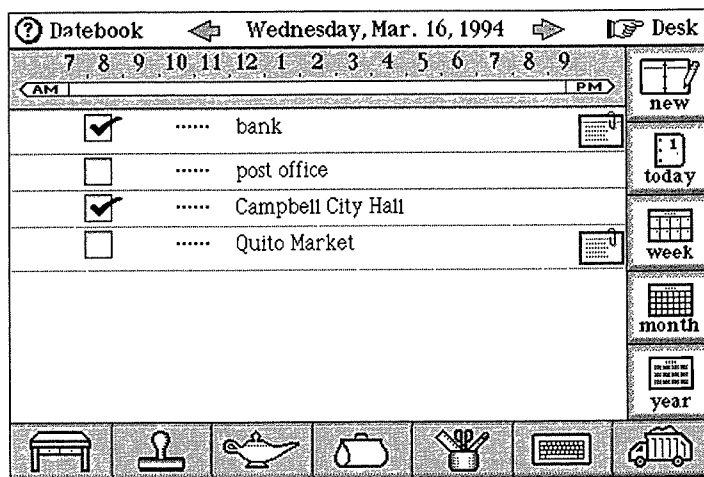


FIGURE 4-11. Day view of datebook with to do appointments

After you've completed a task, you can mark it off with the handy check box that appears next to the item in the datebook's day view. If you don't complete a task, it automatically reappears the next day, and the next, until the deadline comes. After the deadline passes, the task stops carrying forward. In the week and month view of the calendar, *to do* items are identified by a picture of paper and pencil at the top of the date, reminding you

that there are things that need to be done that don't really qualify as appointments. The space below the date is reserved for real appointments.



Seeing into the Future. If you're wondering why the suggested deadline for a task is a year away, here's the thinking of the datebook's designers. A *to do* item wouldn't be very helpful if you needed to keep adding it to your daily schedule until it was finished. The mission of a *to do* entry in Magic Cap is to move automatically from day to day, gently reminding you, as long as the task hasn't been completed and its check box is blank. When you reach the deadline and the task remains unchecked, it finally stops on that date. Since most people wouldn't have a simple task that takes a year to complete, the datebook can have 365 days to keep needling you about it, which should be plenty. You might imagine a future version of Magic Cap that warns you when an undone task is about to hit its deadline.

Day after Day

Magic Cap's datebook lets you schedule events that span multiple days. When you enter one of these, the appointment stretches out across all its days on the week and month views and appears at the top of each day's view during the event.

If you travel for business, you know that the best business trips are a mixed blessing, and the worst ones can be real nightmares. Let's say you have to travel to a customer's office on an extended trip, that is, one that will last more than just one day.

To enter the trip into the datebook, tap *new* and then *business trip* to get started. The entry screen for business trips is straightforward and very businesslike. You get to enter a description of the trip from a standard list of multi-day events, or (as always) you can add your own description, which will make it available again the next time you schedule a multi-day event. Then, you enter the date the trip starts and the date it ends. That's it. As with any appointment, you can add notes to remind you of any vital details. For this event, you can put your departing flight information in the notes.

The datebook knows about other kinds of appointments that span multiple days. When you tap *new*, you can choose from three different multi-day appointments: *business trip*, *vacation*, or *multi-day*, as shown at the bottom of the window in Figure 4-12. The first two, *business trip* and *vacation*, are really just multi-day appointments with the *what* information already filled in. If you choose the generic multi-day appointment, you can fill in the appointment description from choices that include *conference*, *holiday*, and *hibernation*, among others. Of course, you can add your own choices, too, and any new entries (as I'm sure you've already guessed) will be offered whenever you make a new multi-day appointment.

The day view has one image for *business trip*, one for *vacation*, and one for every other kind of multi-day appointment. The images are different only in the day view; week and month views simply show a line across the days of the event. All the multi-day appointments have space for a description, a starting date, and an ending date.

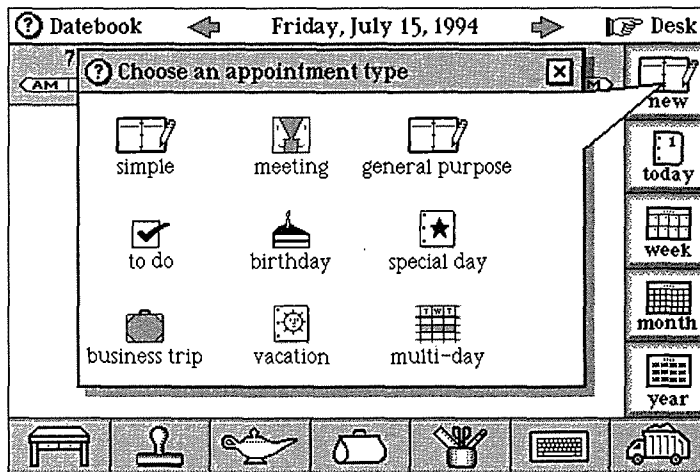


FIGURE 4-12. Three kinds of multi-day appointments

We've dealt with a business trip; now let's schedule a vacation. Let's imagine that your family is going to spend Thanksgiving at Disneyland. It's kind of a tradition for folks on the West Coast who find themselves far from their families for the turkey holiday. To get ready to enter this happy appointment, tap *year* to see the year view, then tap the calendar for November. You'll probably notice the image of the hapless, headless poultry on the fourth Thursday of the month—that's the week you're looking for (see Figure 4-13).

Now you can touch the day your vacation starts, and you'll see the view for that day. Tap *new*, and then *vacation*. The datebook suggests *today* as the starting date for your vacation—if only it were true. It's not, so you tap *from* to enter the starting date of your vacation, then tap *until* to enter the vacation's ending date. Both dates are entered with the same date chooser that you get used to seeing in Magic Cap every time you need to specify a date.

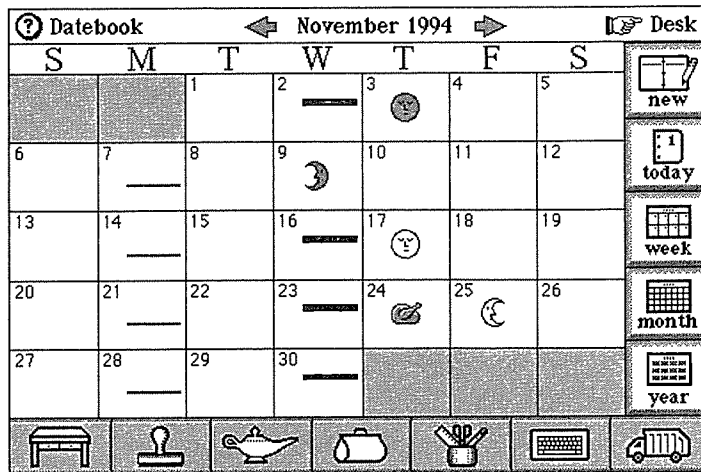


FIGURE 4-13. Month view for November

If you really enjoy your holiday trip to Disneyland, you might want the appointment to say *Disneyland* instead of *vacation*. Tap *what* to see the list of descriptions, then *new* to enter a new one. Magic Cap places a typing point in the right place and opens the on-screen keyboard, and you can type *Disneyland*. The new description appears with the appointment, and it'll be in the list the next time you make a multi-day appointment. Not only have you entered your Disneyland vacation, but the next time you make a multi-day appointment, Disneyland will be listed as a choice right beside the ones that are built into Magic Cap, as you can see in Figure 4-14.

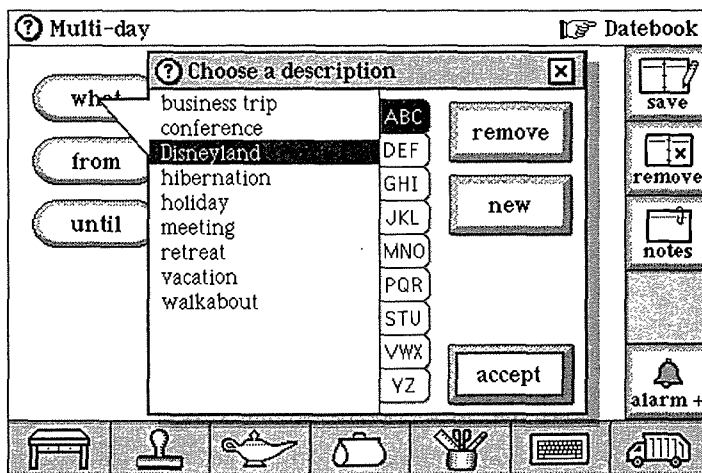



FIGURE 4-14. List of choices for multi-day appointments

Now you can add more detailed information about the trip by writing notes that include flight information and rental car confirmation numbers. You can get to the notes page by tapping *notes* and then typing or writing the notes on the screen. In a practically perfect future version of the datebook, you can imagine being able to attach a different note for each day, allowing you to enter specific information for each day and still enjoy the ease of a multi-day entry.

When you're done entering the vacation, you can tap *save* and see the appointment pop into the datebook. If you glance at the week view or month view, you'll see the vacation blocked out in November, right across that turkey image on Thanksgiving.

 **Busy Bodies.** Do you ever schedule appointments that overlap? Lots of people do, some more than others. Magic Cap's datebook intentionally lets you make overlapping appointments, a process called double-

booking. Making overlapping appointments is easy: Just do it. You can have as many appointments overlapping as your sanity will allow. Even if you're not in the practice of double-booking, you might find this feature handy for scheduling appointments during all day or multi-day events, such as a meeting during a business trip or a sky-diving session while you're on vacation.

Customizing

You probably have lots of favorite tricks for personalizing your paper datebooks. You'll find that Magic Cap caters to your creativity by letting you customize things to make it fit you better and more comfortably. You can do simple, helpful things, like changing the pictures that identify meetings, and you can also perform tricks that are a little fancier, like making a whole new kind of appointment.

Earlier in this chapter, you entered a repeating appointment for a monthly class meeting at school. The standard datebook image for a meeting, the face-to-face picture, doesn't really suggest a class meeting with the teacher and dozens of parents. You can fix this. While you're looking at the day view, you can tap the stamper and select the small picture of a blackboard. As you touch the blackboard, the window of stamps closes and you can slide the blackboard into place to replace the generic image for the meeting (see Figure 4-15).

You might remember that these meetings were entered as repeating appointments on the third Wednesday of each month. As you look at the class meetings through the forthcoming months, you'll see that they're now all represented by the blackboard image.

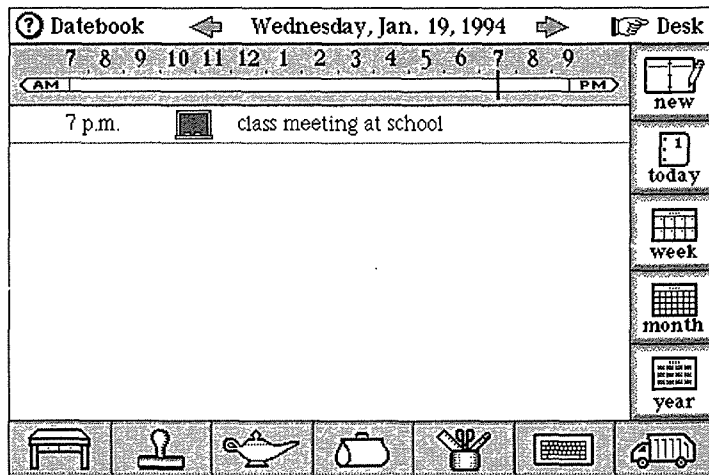


FIGURE 4-15. Customized image for a meeting

If you're a sports fan, you might spend a lot of time at the ballpark, stadium, or arena. When you go to a game, many of the details stay the same from one game to the next, just as with an appointment—*what* a game is; *where* the team's home is; and the *time*, which is probably the same for most games. (As fans of the San Francisco Giants, my family visits Candlestick Park for lots of afternoon games, but we avoid those frigid night games, so almost all our games start at 1:05 P.M.) The only things that change are the date of the game and the team they're playing. You can fill in *who* with the name of the opponent, if you don't mind having the names of hated teams in your name file.

To make it easier to schedule the games you attend each season, you can make a new general purpose appointment, fill in the *what*, *where*, and *time* parts of the appointment, and then tap *save*. When you see the new appointment in the day view, you can pick an

appropriate stamp from the leisure drawer and slide it onto the appointment. Stamps for baseball, football, basketball, and other favorite pastimes are provided. The appointment is pictured in Figure 4-16.

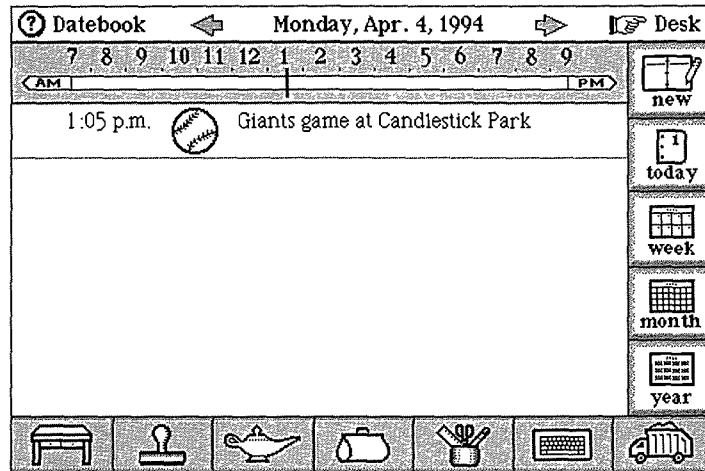


FIGURE 4-16. Day view with a baseball game appointment

Now for some magic: you can slide this appointment onto the *new* button, where it gets slurped up and added to the standard appointment types. From now on, whenever you tap *new*, you'll be able to schedule a day for going to see your favorite team play using a custom-made appointment type that has all the generic information filled in. Maybe a future version of Magic Cap will be able to tell you the score of each game before you attend.

Reminders

The datebook-makers at General Magic realized that appointments need to include a lot of special details. The datebook includes a button labeled *alarm+*, where you

can set up some miscellaneous details about an appointment, including the appointment's status, priority, and alarm.

Because personal communicators are much better at remembering things than people are, you can tell the datebook to remind you of appointments as they come up. Using the *alarm+* feature, you can ask the datebook to remind you about an appointment from a week before the event to the scheduled appointment time.

If you know you have a deadline coming up on a project, you can ask the datebook to remind you a day before the deadline so that you have time to make sure that everything is ready. To do this, make a new appointment for the project, tap *alarm+*, and then use the arrows on the *alarm* choice box to see the options until you come to *1 day early*, the one you want to set. When you save the appointment and look at it in the day view, you'll see, as shown in Figure 4-17, that it now has a little bell as a reminder of its alarm.

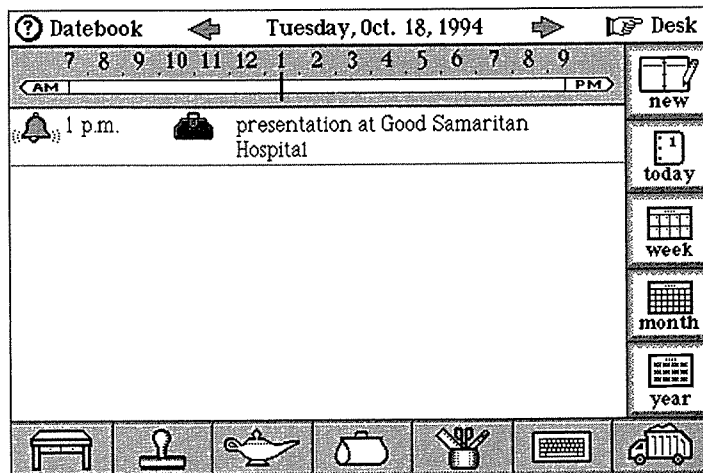


FIGURE 4-17. Day view with an appointment set with a reminder alarm

When you make a date to have lunch with a friend, you might worry about getting so wrapped up in working that you'll forget to look at the time, even though it's conveniently displayed at the top of your communicator's screen. When you set the lunch appointment, you can ask the datebook to remind you 30 minutes early so you won't be late—or at least you'll get to make an informed decision about being late.

Just exactly how will you be reminded about these appointments? As with many other features, Magic Cap supplies a standard way that this will happen, and it also lets you customize it if you want something different. For appointment alarms, the datebook will remind you in two ways: by playing the alarm sound (which sounds not unlike those irritating digital watches) and by displaying an announcement on the screen that will stay there until you close it. You can customize those reminders by changing the datebook's rules. See the "Customizing with Rules" section later in this chapter for more information about how to do that.

Appointment Priority and Status

Magic Cap can schedule appointments that might change or move around before they happen. Imagine that your father is planning a visit and you know the dates he's planning to come, but he might have to cancel due to a possible conflict. You can make a multi-day appointment in the datebook for the visit.

Because you're not sure if he'll be able to come, you'd like to make the appointment tentative. To do this, you tap *alarm+*, which gives you choices for alarm, status, and priority (see Figure 4-18). Use the *status* choice box to select *tentative*. When you save the appointment, it shows up in the day view with a wondering question mark to remind you of its precarious nature.