A

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF EXPRESS MAILING

I hereby certify that this paper and the documents and/or fees referred to as attached therein are being deposited with the United States Postal Service on January 05, 1999 in an envelope as "Express Mail Post Office to Addressee" service under 37 CFR §1.10, Mailing Label Number **EL221766053US**, addressed to the Assistant Commissioner for Patents, Washington, DC 20231.

Michael L. Gough

01/05/99 JC571 U.S. PTO

Attorney Docket No.: SRI1P016

First Named Inventor:

CHEYER, Adam J.

JC618 U.S. PTO 09/225198 01/05/99

UTILITY PATENT APPLICATION TRANSMITTAL (37 CFR § 1.53(b))

Assistant Commissioner for Patents
Box Patent Application
Washington, DC 20231

☐ Duplicate for fee processing

Sir: This is a request for filing a patent application under 37 CFR § 1.53(b) in the name of inventors: Adam J. Cheyer and David L. Martin

For: **SOFTWARE-BASED ARCHITECTURE FOR COMMUNICATION AND COOPERATION AMONG DISTRIBUTED ELECTRONIC AGENTS**

Application Elements:

- ☒ 59 Pages of Specification, Claims and Abstract
- ☒ 16 Sheets of Drawings
- ☒ 01 Pages Combined Declaration and Power of Attorney

Accompanying Application Parts:

- ☒ Assignment and Assignment Recordation Cover Sheet (recording fee not enclosed)
- ☒ Return Receipt Postcard

Fee Calculation (37 CFR § 1.16)

	(Col. 1) NO. FILED	(Col. 2) NO. EXTRA	SMALL ENTITY RATE	SMALL ENTITY FEE	OR	LARGE ENTITY RATE	LARGE ENTITY FEE
BASIC FEE			$395	$	OR	$760	$ 760.00
TOTAL CLAIMS	89 -20 =	69	x11 =	$	OR	x18 =	$1242.00
INDEP CLAIMS	06 -03 =	03	x41 =	$	OR	x78 =	$ 234.00
* If the difference in Col. 1 is less than zero, enter "0" in Col. 2.			Total	$	OR	Total	$2236.00

Including filing fees and the assignment recordation fee of $40.00, the Commissioner is authorized to charge all required fees to Deposit Account No. 50-0384 (Order No. SRI1P016).

☒ The Commissioner is authorized to charge any fees beyond the amount enclosed which may be required, or to credit any overpayment, to Deposit Account No. 50-0384 (Order No. SRI1P016).

General Authorization for Petition for Extension of Time (37 CFR §1.136)

☒ Applicants hereby make and generally authorize any Petitions for Extensions of Time as may be needed for any subsequent filings. The Commissioner is also authorized to charge any extension fees under 37 CFR §1.17 as may be needed to Deposit Account No. 50-0384.

☒ Please send correspondence to the following address:

Brian R. Coleman
HICKMAN STEPHENS & COLEMAN, LLP
P.O. Box 52037
Palo Alto, CA 94303-0746

Tel (650) 470-7430
Fax (650) 470-7440

Date: 1/5/99

Brian R. Coleman
Registration No. **39,145**

Software-Based Architecture for Communication and Cooperation Among Distributed Electronic Agents

By:

Adam J. Cheyer and David L. Martin

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention is related to distributed computing environments and the completion of tasks within such environments. In particular, the present invention teaches a variety of software-based architectures for communication and cooperation among distributed electronic agents. Certain embodiments teach interagent communication languages enabling client agents to make requests in the form of arbitrarily complex goal expressions that are solved through facilitation by a facilitator agent.

Context and Motivation for Distributed Software Systems

The evolution of models for the design and construction of distributed software systems is being driven forward by several closely interrelated trends: the adoption of a *networked computing model*, rapidly rising expectations for *smarter, longer-lived, more autonomous software applications* and an ever increasing demand for *more accessible and intuitive user interfaces.*

Prior Art Figure 1 illustrates a *networked computing model* 100 having a plurality of client and server computer systems 120 and 122 coupled together over a physical transport mechanism 140. The adoption of the *networked computing model* 100 has lead to a greatly increased reliance on distributed sites for both data and processing resources. Systems such as the networked computing model 100 are based upon at least one physical transport mechanism 140 coupling the multiple computer systems 120 and 122 to support the transfer of information between these computers. Some of these computers basically support using the network and are known as *client* *computers* (*clients*). Some of these computers provide resources to other computers and are known as *server computers* (*servers*). The servers 122 can vary greatly in the resources they possess, access they provide and services made available to other computers across a network. Servers may service other servers as well as clients.

The Internet is a computing system based upon this network computing model. The Internet is continually growing, stimulating a paradigm shift for computing away from requiring all relevant data and programs to reside on the user's desktop machine. The data now routinely accessed from computers spread around the world has become increasingly rich in format, comprising multimedia documents, and audio and video streams. With the popularization of programming languages such as JAVA, data transported between local and remote machines may also include programs that can be downloaded and executed on the local machine. There is an ever increasing reliance on networked computing, necessitating software design approaches that allow for flexible composition of distributed processing elements in a dynamically changing and relatively unstable environment.

In an increasing variety of domains, application designers and users are coming to expect the deployment of *smarter, longer-lived, more autonomous, software applications.* Push technology, persistent monitoring of information sources, and the maintenance of user models, allowing for personalized responses and sharing of preferences, are examples of the simplest manifestations of this trend. Commercial enterprises are introducing significantly more advanced approaches, in many cases employing recent research results from artificial intelligence, data mining, machine learning, and other fields.

More than ever before, the increasing complexity of systems, the development of new technologies, and the availability of multimedia material and environments are creating a demand for *more accessible and intuitive user interfaces.* Autonomous, distributed, multi-component systems providing sophisticated services will no longer lend themselves to the familiar "direct manipulation" model of interaction, in which an individual user masters a fixed selection of commands provided by a single application. Ubiquitous computing, in networked environments, has brought about a situation in which the typical user of many software services is likely to be a non-expert, who may access a given service infrequently or only a few times.

Accommodating such usage patterns calls for new approaches. Fortunately, input modalities now becoming widely available, such as speech recognition and pen-based handwriting/gesture recognition, and the ability to manage the presentation of systems' responses by using multiple media provide an opportunity to fashion a style of human-computer interaction that draws much more heavily on our experience with human-human interactions.

PRIOR RELATED ART

Existing approaches and technologies for distributed computing include distributed objects, mobile objects, blackboard-style architectures, and agent-based software engineering.

The Distributed Object Approach

Object-oriented languages, such as C++ or JAVA, provide significant advances over standard procedural languages with respect to the reusability and modularity of code: *encapsulation*, *inheritance* and *polymorhpism*. Encapsulation encourages the creation of library interfaces that minimize dependencies on underlying algorithms or data structures. Changes to programming internals can be made at a later date with requiring modifications to the code that uses the library. Inheritance permits the extension and modification of a library of routines and data without requiring source code to the original library. Polymorphism allows one body of code to work on an arbitrary number of data types. For the sake of simplicity traditional objects may be seen to contain both methods and data. Methods provide the mechanisms by which the internal state of an object may be modified or by which communication may occur with another object or by which the instantiation or removal of objects may be directed.

With reference to Figure 2, a distributed object technology based around an Object Request Broker will now be described. Whereas "standard" object-oriented programming (OOP) languages can be used to build monolithic programs out of many object building blocks, distributed object technologies (DOOP) allow the creation of programs whose components may be spread across multiple machines. As shown in Figure 2, an object system 200 includes client objects 210 and server objects 220. To implement a client-server relationship between objects, the distributed object system

200 uses a registry mechanism (CORBA's registry is called an Object Request Broker, or ORB) 230 to store the interface descriptions of available objects. Through the services of the ORB 230, a client can transparently invoke a method on a remote server object. The ORB 230 is then responsible for finding the object 220 that can implement the request, passing it the parameters, invoking its method, and returning the results. In the most sophisticated systems, the client 210 does not have to be aware of where the object is located, its programming language, its operating system, or any other system aspects that are not part of the server object's interface.

Although distributed objects offer a powerful paradigm for creating networked applications, certain aspects of the approach are not perfectly tailored to the constantly changing environment of the Internet. A major restriction of the DOOP approach is that the interactions among objects are fixed through explicitly coded instructions by the application developer. It is often difficult to reuse an object in a new application without bringing along all its inherent dependencies on other objects (embedded interface definitions and explicit method calls). Another restriction of the DOOP approach is the result of its reliance on a remote procedure call (RPC) style of communication. Although easy to debug, this single thread of execution model does not facilitate programming to exploit the potential for parallel computation that one would expect in a distributed environment. In addition, RPC uses a blocking (synchronous) scheme that does not scale well for high-volume transactions.

Mobile Objects

Mobile objects, sometimes called mobile agents, are bits of code that can move to another execution site (presumably on a different machine) under their own programmatic control, where they can then interact with the local environment. For certain types of problems, the mobile object paradigm offers advantages over more traditional distributed object approaches. These advantages include network bandwidth and parallelism. Network bandwidth advantages exist for some database queries or electronic commerce applications, where it is more efficient to perform tests on data by bringing the tests to the data than by bringing large amounts of data to the testing program. Parallelism advantages include situations in which mobile agents can be spawned in parallel to accomplish many tasks at once.

Some of the disadvantages and inconveniences of the mobile agent approach include the programmatic specificity of the agent interactions, lack of coordination support between participant agents and execution environment irregularities regarding specific programming languages supported by host processors upon which agents reside. In a fashion similar to that of DOOP programming, an agent developer must programmatically specify where to go and how to interact with the target environment. There is generally little coordination support to encourage interactions among multiple (mobile) participants. Agents must be written in the programming language supported by the execution environment, whereas many other distributed technologies support heterogeneous communities of components, written in diverse programming languages.

Blackboard Architectures

Blackboard architectures typically allow multiple processes to communicate by reading and writing tuples from a global data store. Each process can watch for items of interest, perform computations based on the state of the blackboard, and then add partial results or queries that other processes can consider. Blackboard architectures provide a flexible framework for problem solving by a dynamic community of distributed processes. A blackboard architecture provides one solution to eliminating the tightly bound interaction links that some of the other distributed technologies require during interprocess communication. This advantage can also be a disadvantage: although a programmer does not need to refer to a specific process during computation, the framework does not provide programmatic control for doing so in cases where this would be practical.

Agent-based Software Engineering

Several research communities have approached distributed computing by casting it as a problem of modeling communication and cooperation among autonomous entities, or agents. Effective communication among independent agents requires four components: (1) a transport mechanism carrying messages in an asynchronous fashion, (2) an interaction protocol defining various types of communication interchange and their social implications (for instance, a response is expected of a question), (3) a content language permitting the expression and interpretation of utterances, and (4) an agreed-upon set of shared vocabulary and meaning for concepts (often called an *ontology*). Such mechanisms permit a much richer style of interaction among participants than can be expressed using a distributed object's RPC model or a blackboard architecture's centralized exchange approach.

Agent-based systems have shown much promise for flexible, fault-tolerant, distributed problem solving. Several agent-based projects have helped to evolve the notion of facilitation. However, existing agent-based technologies and architectures are typically very limited in the extent to which agents can specify complex goals or influence the strategies used by the facilitator. Further, such prior systems are not sufficiently attuned to the importance of integrating human agents (i.e., users) through natural language and other human-oriented user interface technologies.

The initial version of SRI International's Open Agent Architecture™ ("*OAA*®") technology provided only a very limited mechanism for dealing with compound goals. Fixed formats were available for specifying a flat list of either conjoined (AND) sub-goals or disjoined (OR) sub-goals; in both cases, parallel goal solving was hard-wired in, and only a single set of parameters for the entire list could be specified. More complex goal expressions involving (for example) combinations of different boolean connectors, nested expressions, or conditionally interdependent ("IF .. THEN") goals were not supported. Further, system scalability was not adequately addressed in this prior work.

SUMMARY OF INVENTION

A first embodiment of the present invention discloses a highly flexible, software-based architecture 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 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 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 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. Specific embodiments providing enhanced scalability are also described.

BRIEF DESCRIPTION OF THE DRAWINGS

Prior Art

Prior Art FIGURE 1 depicts a networked computing model;

Prior Art FIGURE 2 depicts a distributed object technology based around an Object Resource Broker;

Examples of the Invention

FIGURE 3 depicts a distributed agent system based around a facilitator agent;

FIGURE 4 presents a structure typical of one small system of the present invention;

FIGURE 5 depicts an Automated Office system implemented in accordance with an example embodiment of the present invention supporting a mobile user with a laptop computer and a telephone;

FIGURE 6 schematically depicts an Automated Office system implemented as a network of agents in accordance with a preferred embodiment of the present invention;

FIGURE 7 schematically shows data structures internal to a facilitator in accordance with a preferred embodiment of the present invention;

FIGURE 8 depicts operations involved in instantiating a client agent with its parent facilitator in accordance with a preferred embodiment of the present invention;

FIGURE 9 depicts operations involved in a client agent initiating a service request and receiving the response to that service request in accordance with a certain preferred embodiment of the present invention;

FIGURE 10 depicts operations involved in a client agent responding to a service request in accordance with another preferable embodiment of the present invention;

FIGURE 11 depicts operations involved in a facilitator agent response to a service request in accordance with a preferred embodiment of the present invention;

FIGURE 12 depicts an Open Agent Architecture™ based system of agents implementing a unified messaging application in accordance with a preferred embodiment of the present invention;

FIGURE 13 depicts a map oriented graphical user interface display as might be displayed by a multi-modal map application in accordance with a preferred embodiment of the present invention;

FIGURE 14 depicts a peer to peer multiple facilitator based agent system supporting distributed agents in accordance with a preferred embodiment of the present invention;

FIGURE 15 depicts a multiple facilitator agent system supporting at least a limited form of a hierarchy of facilitators in accordance with a preferred embodiment of the present invention; and

FIGURE 16 depicts a replicated facilitator architecture in accordance with one embodiment of the present invention.

BRIEF DESCRIPTION OF THE APPENDICES

The Appendices provide source code for an embodiment of the present invention written in the PROLOG programming language.

APPENDIX A: Source code file named compound.pl.

APPENDIX B: Source code file named fac.pl.

APPENDIX C: Source code file named libcom_tcp.pl.

APPENDIX D: Source code file named liboaa.pl.

APPENDIX E: Source code file named translations.pl.

DETAILED DESCRIPTION OF THE INVENTION

Figure 3 illustrates a distributed agent system 300 in accordance with one embodiment of the present invention. The agent system 300 includes a facilitator agent 310 and a plurality of agents 320. The illustration of Figure 3 provides a high level view of one simple system structure contemplated by the present invention. The facilitator agent 310 is in essence the "parent" facilitator for its "children" agents 320. The agents 320 forward service requests to the facilitator agent 310. The facilitator agent 310 interprets these requests, organizing a set of goals which are then delegated to appropriate agents for task completion.

The system 300 of Figure 3 can be expanded upon and modified in a variety of ways consistent with the present invention. For example, the agent system 300 can be distributed across a computer network such as that illustrated in Figure 1. The facilitator agent 310 may itself have its functionality distributed across several different computing platforms. The agents 320 may engage in interagent communication (also called peer to peer communications). Several different systems 300 may be coupled together for enhanced performance. These and a variety of other structural configurations are described below in greater detail.

Figure 4 presents the structure typical of a small system 400 in one embodiment of the present invention, showing user interface agents 408, several application agents 404 and meta-agents 406, the system 400 organized as a community of peers by their common relationship to a facilitator agent 402. As will be appreciated, Figure 4 places more structure upon the system 400 than shown in Figure 3, but both are valid representations of structures of the present invention. The facilitator 402 is a specialized server agent that is responsible for coordinating agent communications and cooperative problem-solving. The facilitator 402 may also provide a global data store for its client agents, allowing them to adopt a blackboard style of interaction. Note that certain advantages are found in utilizing two or more facilitator agents within the system 400. For example, larger systems can be assembled from multiple facilitator/client groups, each having the sort of structure shown in Figure 4. All agents that are not facilitators are referred to herein generically as *client* agents -- so called because each acts (in some respects) as a client of some facilitator, which provides communication and other essential services for the client.

The variety of possible client agents is essentially unlimited. Some typical categories of client agents would include application agents 404, meta-agents 406, and user interface agents 408, as depicted in Figure 4. Application agents 404 denote specialists that provide a collection of services of a particular sort. These services could be domain-independent technologies (such as speech recognition, natural language processing 410, email, and some forms of data retrieval and data mining) or user-specific or domain-specific (such as a travel planning and reservations agent). Application agents may be based on legacy applications or libraries, in which case the agent may be little more than a wrapper that calls a pre-existing API 412, for example. Meta-agents 406 are agents whose role is to assist the facilitator agent 402 in coordinating the activities of other agents. While the facilitator 402 possesses domain-independent coordination strategies, meta-agents 406 can augment these by using domain- and application-specific knowledge or reasoning (including but not limited to rules, learning algorithms and planning).

With further reference to Figure 4, user interface agents 408 can play an extremely important and interesting role in certain embodiments of the present invention. By way of explanation, in some systems, a user interface agent can be implemented as a collection of "micro-agents", each monitoring a different input modality (point-and-click, handwriting, pen gestures, speech), and collaborating to produce the best interpretation of the current inputs. These micro-agents are depicted in Figure 4, for example, as Modality Agents 414. While describing such subcategories of client agents is useful for purposes of illustration and understanding, they need not be formally distinguished within the system in preferred implementations of the present invention.

The operation of one preferred embodiment of the present invention will be discussed in greater detail below, but may be briefly outlined as follows. When invoked, a client agent makes a connection to a facilitator, which is known as its *parent facilitator*. These connections are depicted as a double headed arrow between the client agent and the facilitator agent in Figure 3 and 4, for example. Upon connection, an agent registers with its parent facilitator a specification of the capabilities and services it can provide. For example, a natural language agent may register the characteristics of its available natural language vocabulary. (For more details regarding client agent connections, see the discussion of Figure 8 below.) Later during task completion, when a facilitator determines that the registered services 416 of one of its client agents will help satisfy a goal, the facilitator sends that client a request expressed in the Interagent Communication Language (*ICL*) 418. (See Figure 11 below for a more detailed discussion of the facilitator operations involved.) The agent parses this request, processes it, and returns answers or status reports to the facilitator. In processing a request, the client agent can make use of a variety of infrastructure capabilities provided in the preferred embodiment. For example, the client agent can use *ICL* 418 to request services of other agents, set triggers, and read or write shared data on the facilitator or other client agents that maintain shared data. (See the discussion of Figures 9-11 below for a more detailed discussion of request processing.)

The functionality of each client agent are made available to the agent community through registration of the client agent's capabilities with a facilitator 402. A software "wrapper" essentially surrounds the underlying application program performing the services offered by each client. The common infrastructure for constructing agents is preferably supplied by an *agent library.* The agent library is preferably accessible in the runtime environment of several different programming languages. The agent library preferably minimizes the effort required to construct a new system and maximizes the ease with which legacy systems can be “wrapped” and made compatible with the agent-based architecture of the present invention.

By way of further illustration, a representative application is now briefly presented with reference to Figures 5 and 6. In the Automated Office system depicted in Figure 5, a mobile user with a telephone and a laptop computer can access and task commercial applications such as calendars, databases, and email systems running back at the office. A user interface (UI) agent 408, shown in Figure 6, runs on the user's local laptop and is responsible for accepting user input, sending requests to the facilitator 402 for delegation to appropriate agents, and displaying the results of the distributed computation. The user may interact directly with a specific remote application by clicking on active areas in the interface, calling up a form or window for that application, and making queries with standard interface dialog mechanisms. Conversely, a user may express a task to be executed by using typed, handwritten, or spoken (over the telephone) English sentences, without explicitly specifying which agent or agents should perform the task.

For instance, if the question "What is my schedule?" is written 420 in the user interface 408, this request will be sent 422 by the UI 408 to the facilitator 402, which in turn will ask 424 a natural language (NL) agent 426 to translate the query into *ICL* 18. To accomplish this task, the NL agent 426 may itself need to make requests of the agent community to resolve unknown words such as "me" 428 (the UI agent 408 can respond 430 with the name of the current user) or "schedule" 432 (the calendar agent 434 defines this word 436). The resulting *ICL* expression is then routed by the facilitator 402 to appropriate agents (in this case, the calendar agent 434) to execute the request. Results are sent back 438 to the UI agent 408 for display.

The spoken request "When mail arrives for me about security, notify me immediately." produces a slightly more complex example involving communication among all agents in the system. After translation into *ICL* as described above, the facilitator installs a trigger 440 on the mail agent 442 to look for new messages about security. When one such message does arrive in its mail spool, the trigger fires, and the facilitator matches the action part of the trigger to capabilities published by the notification agent 446. The notification agent 446 is a meta-agent, as it makes use of rules concerning the optimal use of different output modalities (email, fax, speech generation over the telephone) plus information about an individual user's preferences 448 to determine the best way of relaying a message through available media transfer application agents. After some competitive parallelism to locate the user (the calendar agent 434 and database agent 450 may have different guesses as to where to find the user) and some cooperative parallelism to produce required information (telephone number of location, user password, and an audio file containing a text-to-speech representation of the email message), a telephone agent 452 calls the user, verifying its identity through touchtones, and then play the message.

The above example illustrates a number of inventive features. As new agents connect to the facilitator, registering capability specifications and natural language vocabulary, what the user can say and do dynamically changes; in other words, the ICL is dynamically ***expandable***. For example, adding a calendar agent to the system in the previous example and registering its capabilities enables users to ask natural language questions about their "schedule" without any need to revise code for the facilitator, the natural language agents, or any other client agents. In addition, the interpretation and execution of a task is a distributed process, with no single agent defining the set of possible inputs to the system. Further, a single request can produce cooperation and flexible communication among many agents, written in different programming languages and spread across multiple machines.

Design Philosophy and Considerations

One preferred embodiment provides an integration mechanism for heterogeneous applications in a distributed infrastructure, incorporating some of the dynamism and extensibility of blackboard approaches, the efficiency associated with mobile objects, plus the rich and complex interactions of communicating agents. Design goals for preferred embodiments of the present invention may be categorized under the general headings of *interoperation and cooperation, user interfaces*, and *software engineering.* These design goals are not absolute requirements, nor will they necessarily be satisfied by all embodiments of the present invention, but rather simply reflect the inventor's currently preferred design philosophy.

Versatile mechanisms of interoperation and cooperation

Interoperation refers to the ability of distributed software components - agents - to communicate meaningfully. While every system-building framework must provide mechanisms of interoperation at some level of granularity, agent-based frameworks face important new challenges in this area. This is true primarily because autonomy, the hallmark of *individual* agents, necessitates greater flexibility in interactions within *communities* of agents. *Coordination* refers to the mechanisms by which a community of agents is able to work together productively on some task. In these areas, the goals for our framework are to *provide flexibility in assembling* *communities of autonomous service providers*, *provide flexibility in structuring cooperative interactions*, *impose the right amount of structure*, as well as *include legacy and "owned-elsewhere" applications.*

Provide flexibility in assembling communities of autonomous service providers -- both at development time and at runtime. Agents that conform to the linguistic and ontological requirements for effective communication should be able to participate in an agent community, in various combinations, with minimal or near minimal prerequisite knowledge of the characteristics of the other players. Agents with duplicate and overlapping capabilities should be able to coexist within the same community, with the system making optimal or near optimal use of the redundancy.

Provide flexibility in structuring cooperative interactions among the members of a community of agents. A framework preferably provides an economical mechanism for setting up a variety of interaction patterns among agents, without requiring an inordinate amount of complexity or infrastructure within the individual agents. The provision of a service should be independent or minimally dependent upon a particular configuration of agents.

Impose the right amount of structure on individual agents. Different approaches to the construction of multi-agent systems impose different requirements on the individual agents. For example, because KQML is neutral as to the content of messages, it imposes minimal structural requirements on individual agents. On the other hand, the BDI paradigm tends to impose much more demanding requirements, by making assumptions about the nature of the programming elements that are meaningful to individual agents. Preferred embodiments of the present invention should fall somewhere between the two, providing a rich set of interoperation and coordination capabilities, without precluding any of the software engineering goals defined below.

Include legacy and "owned-elsewhere" applications. Whereas *legacy* usually implies reuse of an established system fully controlled by the agent-based system developer, *owned-elsewhere* refers to applications to which the developer has partial access, but no control. Examples of owned-elsewhere applications include data sources and services available on the World Wide Web, via simple form-based interfaces, and applications used cooperatively within a virtual enterprise, which remain the properties of separate corporate entities. Both classes of application must preferably be able to interoperate, more or less as full-fledged members of the agent community, without requiring an overwhelming integration effort.

Human-oriented user interfaces

Systems composed of multiple distributed components, and possibly dynamic configurations of components, require the crafting of intuitive user interfaces to *provide conceptually natural interaction mechanisms, treat users as privileged members of the agent community* and *support collaboration.*

Provide conceptually natural interaction mechanisms with multiple distributed components. When there are numerous disparate agents, and/or complex tasks implemented by the system, the user should be able to express requests without having detailed knowledge of the individual agents. With speech recognition, handwriting recognition, and natural language technologies becoming more mature, agent architectures should preferably support these forms of input playing increased roles in the tasking of agent communities.

Preferably treat *users as privileged members* of the agent community by providing an appropriate level of task specification within *software* agents, and reusable translation mechanisms between this level and the level of *human* requests, supporting constructs that seamlessly incorporate interactions between both human-interface and software types of agents.

Preferably support *collaboration* (simultaneous work over shared data and processing resources) between users and agents.

Realistic software engineering requirements

System-building frameworks should preferably address the practical concerns of real-world applications by the specification of requirements which preferably include: *Minimize the effort* required to create new agents, and to wrap existing applications. *Encourage reuse,* both of domain-independent and domain-specific components. The concept of *agent orientation*, like that of object orientation, provides a natural conceptual framework for reuse, so long as mechanisms for encapsulation and interaction are structured appropriately. *Support lightweight, mobile platforms.* Such platforms should be able to serve as hosts for agents, without requiring the installation of a massive environment. It should also be possible to construct individual agents that are relatively small and modest in their processing requirements. *Minimize platform and language barriers.* Creation of new agents, as well as wrapping of existing applications, should not require the adoption of a new language or environment.

Mechanisms of Cooperation

Cooperation among agents in accordance with the present invention is preferably achieved via messages expressed in a common language, *ICL*. Cooperation among agent is further preferably structured around a three-part approach: providers of services register capabilities specifications with a facilitator, requesters of services construct goals and relay them to a facilitator, and facilitators coordinate the efforts of the appropriate service providers in satisfying these goals.

The Interagent Communication Language (ICL)

Interagent Communication Language (*"ICL"*) 418 refers to an interface, communication, and task coordination language preferably shared by all agents, regardless of what platform they run on or what computer language they are programmed in. *ICL* may be used by an agent to task itself or some subset of the agent community. Preferably, *ICL* allows agents to specify explicit control parameters while simultaneously supporting expression of goals in an underspecified, loosely constrained manner. In a further preferred embodiment, agents employ *ICL* to perform queries, execute actions, exchange information, set triggers, and manipulate data in the agent community.

In a further preferred embodiment, a program element expressed in *ICL* is the *event*. The activities of every agent, as well as communications between agents, are preferably structured around the transmission and handling of events. In communications, events preferably serve as messages between agents; in regulating the activities of individual agents, they may preferably be thought of as goals to be satisfied. Each event preferably has a type, a set of parameters, and content. For example, the agent library procedure *oaa_Solve* can be used by an agent to request services of other agents. A call to *oaa_Solve*, within the code of agent *A*, results in an event having the form

ev_post_solve(Goal, Params)

going from *A* to the facilitator, where *ev_post_solve* is the type, *Goal* is the content, and *Params* is a list of parameters. The allowable content and parameters preferably vary according to the type of the event.

The *ICL* preferably includes a layer of conversational protocol and a content layer. The conversational layer of *ICL* is defined by the event types, together with the parameter lists associated with certain of these event types. The content layer consists of the specific goals, triggers, and data elements that may be embedded within various events.

The *ICL* conversational protocol is preferably specified using an orthogonal, parameterized approach, where the conversational aspects of each element of an interagent conversation are represented by a selection of an event type and a selection of values from at least one orthogonal set of parameters. This approach offers greater expressiveness than an approach based solely on a fixed selection of *speech acts*, such as embodied in KQML. For example, in KQML, a request to satisfy a query can employ either of the performatives *ask_all* or *ask_one*. In *ICL*, on the other hand, this type of request preferably is expressed by the event type *ev_post_solve*, together with the *solution_limit(N)* parameter - where *N* can be any positive integer. (A request for all solutions is indicated by the omission of the *solution_limit* parameter.) The request can also be accompanied by other parameters, which combine to further refine its semantics. In KQML, then, this example forces one to choose between two possible conversational options, neither of which may be precisely what is desired. In either case, the performative chosen is a single value that must capture the entire conversational characterization of the communication. This requirement raises a difficult challenge for the language designer, to select a set of performatives that provides the desired functionality without becoming unmanageably large. Consequently, the debate over the right set of performatives has consumed much discussion within the KQML community.

The content layer of the *ICL* preferably supports unification and other features found in logic programming language environments such as PROLOG. In some embodiments, the content layer of the *ICL* is simply an extension of at least one programming language. For example, the Applicants have found that PROLOG is suitable for implementing and extending into the content layer of the *ICL*. The agent libraries preferably provide support for constructing, parsing, and manipulating *ICL* expressions. It is possible to embed content expressed in other languages within an *ICL* event. However, expressing content in *ICL* simplifies the facilitator's access to the content, as well as the conversational layer, in delegating requests. This gives the facilitator more information about the nature of a request and helps the facilitator decompose compound requests and delegate the sub-requests.

Further, *ICL* expressions preferably include, in addition to events, at least one of the following: capabilities declarations, requests for services, responses to requests, trigger specifications, and shared data elements. A further preferred embodiment of the present invention incorporates *ICL* expressions including at least all of the following: events, capabilities declarations, requests for services, responses to requests, trigger specifications, and shared data elements.

Providing Services: Specifying "Solvables"

In a preferred embodiment of the present invention, every participating agent defines and publishes a set of capability declarations, expressed in *ICL*, describing the services that it provides. These declarations establish a high-level interface to the agent. This interface is used by a facilitator in communicating with the agent, and, most important, in delegating service requests (or parts of requests) to the agent. Partly due to the use of PROLOG as a preferred basis for *ICL*, these capability declarations are referred as *solvables.* The agent library preferably provides a set of procedures allowing an agent to add, remove, and modify its solvables, which it may preferably do at any time after connecting to its facilitator.

There are preferably at least two major types of solvables: *procedure* solvables and *data* solvables. Intuitively, a procedure solvable performs a test or action, whereas a data solvable provides access to a collection of data. For example, in creating an agent for a mail system, procedure solvables might be defined for sending a message to a person, testing whether a message about a particular subject has arrived in the mail queue, or displaying a particular message onscreen. For a database wrapper agent, one might define a distinct data solvable corresponding to each of the relations present in the database. Often, a data solvable is used to provide a *shared* data store, which may be not only queried, but also updated, by various agents having the required permissions.

There are several primary technical differences between these two types of solvables. First, each procedure solvable must have a handler declared and defined for it, whereas this is preferably not necessary for a data solvable. The handling of requests for a data solvable is preferably provided transparently by the agent library. Second, data solvables are preferably associated with a dynamic collection of facts (or clauses), which may be further preferably modified at runtime, both by the agent providing the solvable, and by other agents (provided they have the required permissions). Third, special features, available for use with data solvables, preferably facilitate maintaining the associated facts. In spite of these differences, it should be noted that the mechanism of *use* by which an agent requests a service is the same for the two types of solvables.

In one embodiment, a request for one of an agent's services normally arrives in the form of an event from the agent's facilitator. The appropriate handler then deals with this event. The handler may be coded in whatever fashion is most appropriate, depending on the nature of the task, and the availability of task-specific libraries or legacy code, if any. The only hard requirement is that the handler return an appropriate response to the request, expressed in *ICL*. Depending on the nature of the request, this response could be an indication of success or failure, or a list of solutions (when the request is a data query).

A solvable preferably has three parts: a *goal*, a list of *parameters*, and a list of *permissions*, which are declared using the format:

solvable(Goal, Parameters, Permissions)

The goal of a solvable, which syntactically takes the preferable form of an *ICL* structure, is a logical representation of the service provided by the solvable. (An *ICL* structure consists of a *functor* with 0 or more arguments. For example, in the structure a(b,c), `a' is the functor, and `b' and `c' the arguments.) As with a PROLOG structure, the goal's arguments themselves may preferably be structures.

Various options can be included in the parameter list, to refine the semantics associated with the solvable. The *type* parameter is preferably used to say whether the solvable is *data* or *procedure*. When the type is *procedure*, another parameter may be used to indicate the handler to be associated with the solvable. Some of the parameters appropriate for a *data* solvable are mentioned elsewhere in this application. In either case (procedure or data solvable), the *private* parameter may be preferably used to restrict the use of a solvable to the declaring agent when the agent intends the solvable to be solely for its internal use but wishes to take advantage of the mechanisms in accordance with the present invention to access it, or when the agent wants the solvable to be available to outside agents only at selected times. In support of the latter case, it is preferable for the agent to change the status of a solvable from private to non-private at any time.

The permissions of a solvable provide mechanisms by which an agent may preferably control access to its services allowing the agent to restrict calling and writing of a solvable to itself and/or other selected agents. (*Calling* means requesting the service encapsulated by a solvable, whereas *writing* means modifying the collection of facts associated with a data solvable.) The default permission for every solvable in a further preferred embodiment of the present invention is to be callable by anyone, and for data solvables to be writable by anyone. A solvable's permissions can preferably be changed at any time, by the agent providing the solvable.

For example, the solvables of a simple email agent might include:

```
solvable(send_message(email, +ToPerson, +Params),
        [type(procedure), callback(send_mail)],
          [])
solvable(last_message(email, -MessageId),
          [type(data), single_value(true)],
          [write(true)]),
        solvable(get_message(email, +MessageId, -
Msg),
           [type(procedure), callback(get_mail)],
      [])
```

The symbols `+' and `-', indicating input and output arguments, are at present used only for purposes of documentation. Most parameters and permissions have default values, and specifications of default values may be omitted from the parameters and permissions lists.

Defining an agent's capabilities in terms of solvable declarations effectively creates a vocabulary with which other agents can communicate with the new agent. Ensuring that agents will speak the same language and share a common, unambiguous semantics of the vocabulary involves *ontology*. Agent development tools and services (automatic translations of solvables by the facilitator) help address this issue; additionally, a preferred embodiment of the present invention will typically rely on vocabulary from either formally engineered ontologies for specific domains or from ontologies constructed during the incremental development of a body of agents for several applications or from both specific domain ontologies and incrementally developed ontologies. Several example tools and services are described in Cheyer et al.'s paper entitled "Development Tools for the Open Agent Architecture," as presented at the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 96), London, April 1996.

Although the present invention imposes no hard restrictions on the form of solvable declarations, two common usage conventions illustrate some of the utility associated with solvables.

Classes of services are often preferably tagged by a particular type. For instance, in the example above, the "last_message" and "get_message" solvables are specialized for email, not by modifying the *names* of the services, but rather by the use of the `email' parameter, which serves during the execution of an *ICL* request to select (or not) a specific type of message.

Actions are generally written using an imperative verb as the functor of the solvable in a preferred embodiment of the present invention, the direct object (or item class) as the first argument of the predicate, required arguments following, and then an extensible parameter list as the last argument. The parameter list can hold optional information usable by the function. The *ICL* expression generated by a natural language parser often makes use of this parameter list to store prepositional phrases and adjectives.

As an illustration of the above two points, "Send mail to Bob about lunch" will be translated into an *ICL* request send_message(email, `Bob Jones', [subject(lunch)]), whereas "Remind Bob about lunch" would leave the transport unspecified (send_message(KIND, `Bob Jones', [subject(lunch)])), enabling all available message transfer agents (e.g., fax, phone, mail, pager) to compete for the opportunity to carry out the request.

Requesting Services

An agent preferably requests services of the community of agent by delegating tasks or goals to its facilitator. Each request preferably contains calls to one or more agent solvables, and optionally specifies parameters containing advice to help the facilitator determine how to execute the task. Calling a solvable preferably does *not* require that the agent specify (or even know of) a particular agent or agents to handle the call. While it is possible to specify one or more agents using an address parameter (and there are situations in which this is desirable), in general it is advantageous to leave this delegation to the facilitator. This greatly reduces the hard-coded component dependencies often found in other distributed frameworks. The agent libraries of a preferred embodiment of the present invention provide an agent with a single, unified point of entry for requesting services of other agents: the library procedure *oaa_Solve*. In the style of logic programming, *oaa_Solve* may preferably be used both to retrieve data and to initiate actions, so that calling a *data* solvable looks the same as calling a *procedure* solvable.

Complex Goal Expressions

A powerful feature provided by preferred embodiments of the present invention is the ability of a client agent (or a user) to submit compound goals of an arbitrarily complex nature to a facilitator. A compound goal is a single goal expression that specifies multiple sub-goals to be performed. In speaking of a *"complex goal expression"* we mean that a single goal expression that expresses multiple sub-goals can potentially include more than one type of logical connector (e.g., AND, OR, NOT), and/or more than one level of logical nesting (e.g., use of parentheses), or the substantive equivalent. By way of further clarification, we note that when speaking of an *"arbitrarily complex goal expression"* we mean that goals are expressed in a language or syntax that allows expression of such complex goals when appropriate or when desired, not that every goal is itself necessarily complex.

It is contemplated that this ability is provided through an interagent communication language having the necessary syntax and semantics. In one example, the goals may take the form of compound goal expressions composed using operators similar to those employed by PROLOG, that is, the comma for conjunction, the semicolon for disjunction, the arrow for conditional execution, etc. The present invention also contemplates significant extensions to PROLOG syntax and semantics. For example, one embodiment incorporates a "parallel disjunction" operator indicating that the disjuncts are to be executed by different agents concurrently. A further embodiment supports the specification of whether a given sub-goal is to be executed breadth-first or depth-first.

A further embodiment supports each sub-goal of a compound goal optionally having an address and/or a set of parameters attached to it. Thus, each sub-goal takes the form

Address:Goal::Parameters

where both *Address* and *Parameters* are optional.

An address, if present, preferably specifies one or more agents to handle the given goal, and may employ several different types of referring expression: unique names, symbolic names, and shorthand names. Every agent has preferably a unique name, assigned by its facilitator, which relies upon network addressing schemes to ensure its global uniqueness. Preferably, agents also have self-selected symbolic names (for example, "mail"), which are not guaranteed to be unique. When an address includes a symbolic name, the facilitator preferably takes this to mean that all agents having that name should be called upon. Shorthand names include `self' and `parent' (which refers to the agent's facilitator). The address associated with a goal or sub-goal is preferably always optional. When an address is not present, it is the facilitator's job to supply an appropriate address.

The distributed execution of compound goals becomes particularly powerful when used in conjunction with natural language or speech-enabled interfaces, as the query itself may specify how functionality from distinct agents will be combined. As a simple example, the spoken utterance "Fax it to Bill Smith's manager." can be translated into the following compound *ICL* request:

oaa_Solve((manager('Bill Smith', M), fax(it,M,[])), [strategy(action)])

Note that in this ICL request there are two sub-goals, "manager('Bill Smith',M)" and "fax(it,M,[])," and a single global parameter "strategy(action)." According to the present invention, the facilitator is capable of mapping global parameters in order to apply the constraints or advice across the separate sub-goals in a meaningful way. In this instance, the global parameter strategy(action) implies a parallel constraint upon the first sub-goal; i.e., when there are multiple agents that can respond to the manager sub-goal, each agent should receive a request for service. In contrast, for the second sub-goal, parallelism should not be inferred from the global parameter strategy(action) because such an inference would possibly result in the transmission of duplicate facsimiles.

Refining Service Requests

In a preferred embodiment of the present invention, parameters associated with a goal (or sub-goal) can draw on useful features to refine the request's meaning. For example, it is frequently preferred to be able to specify whether or not solutions are to be returned synchronously; this is done using the *reply* parameter, which can take any of the values *synchronous, asynchronous*, or *none*. As another example, when the goal is a non-compound query of a data solvable, the *cache* parameter may preferably be used to request local caching of the facts associated with that solvable. Many of the remaining parameters fall into two categories: feedback and advice.

Feedback parameters allow a service requester to receive information from the facilitator about how a goal was handled. This feedback can include such things as the identities of the agents involved in satisfying the goal, and the amount of time expended in the satisfaction of the goal.

Advice parameters preferably give constraints or guidance to the facilitator in completing and interpreting the goal. For example, a *solution_limit* parameter preferably allows the requester to say how many solutions it is interested in; the facilitator and/or service providers are free to use this information in optimizing their efforts. Similarly, a *time_limit* is preferably used to say how long the requester is willing to wait for solutions to its request, and, in a multiple facilitator system, a *level_limit* may preferably be used to say how remote the facilitators may be that are consulted in the search for solutions. A *priority* parameter is preferably used to indicate that a request is more urgent than previous requests that have not yet been satisfied. Other preferred advice parameters include but are not limited to parameters used to tell the facilitator whether parallel satisfaction of the parts of a goal is appropriate, how to combine and filter results arriving from multiple solver agents, and whether the requester itself may be considered a candidate solver of the sub-goals of a request.

Advice parameters preferably provide an extensible set of low-level, orthogonal parameters capable of combining with the *ICL* goal language to fully express how information should flow among participants. In certain preferred embodiments of the present invention, multiple parameters can be grouped together and given a group name. The resulting *high-level advice parameters* can preferably be used to express concepts analogous to KQML's performatives, as well as define classifications of problem types. For instance, KQML's "ask_all" and "ask_one" performatives would be represented as combinations of values given to the parameters *reply*, *parallel_ok*, and *solution_limit*. As an example of a higher-level problem type, the strategy "math_problem" might preferably send the query to all appropriate math solvers in parallel, collect their responses, and signal a conflict if different answers are returned. The strategy "essay_question" might preferably send the request to all appropriate participants, and signal a problem (i.e., cheating) if any of the returned answers are identical.

Facilitation

In a preferred embodiment of the present invention, when a facilitator receives a compound goal, its job is to construct a goal satisfaction plan and oversee its satisfaction in an optimal or near optimal manner that is consistent with the specified advice. The facilitator of the present invention maintains a knowledge base that records the capabilities of a collection of agents, and uses that knowledge to assist requesters and providers of services in making contact.

Figure 7 schematically shows data structures 700 internal to a facilitator in accordance with one embodiment of the present invention. Consider the function of a Agent Registry 702 in the present invention. Each registered agent may be seen as associated with a collection of fields found within its parent facilitator such as shown in the figure. Each registered agent may optionally possess a Symbolic Name which would be entered into field 704. As mentioned elsewhere, Symbolic Names need not be unique to each instance of an agent. Note that an agent may in certain preferred embodiments of the present invention possess more than one Symbolic Name. Such Symbolic Names would each be found through their associations in the Agent Registry entries. Each agent, when registered, must possess a Unique Address, which is entered into the Unique Address field 706.

With further reference to Figure 7, each registered agent may be optionally associated with one or more capabilities, which have associated Capability Declaration fields 708 in the parent facilitator Agent Registry 702. These capabilities may define not just functionality, but may further provide a utility parameter indicating, in some manner (e.g., speed, accuracy, etc), how effective the agent is at providing the declared capability. Each registered agent may be optionally associated with one or more data components, which have associated Data Declaration fields 710 in the parent facilitator Agent Registry 702. Each registered agent may be optionally associated with one or more triggers, which preferably could be referenced through their associated Trigger Declaration fields 712 in the parent facilitator Agent Registry 702. Each registered agent may be optionally associated with one or more tasks, which preferably could be referenced through their associated Task Declaration fields 714 in the parent facilitator Agent Registry 702. Each registered agent may be optionally associated with one or more Process Characteristics, which preferably could be referenced through their associated Process Characteristics Declaration fields 716 in the parent facilitator Agent Registry 702. Note that these characteristics in certain preferred embodiments of the present invention may include one or more of the following: Machine Type (specifying what type of computer may run the agent), Language (both computer and human interface).

A facilitator agent in certain preferred embodiments of the present invention further includes a Global Persistent Database 720. The database 720 is composed of data elements which do not rely upon the invocation or instantiation of client agents for those data elements to persist. Examples of data elements which might be present in such a database include but are not limited to the network address of the facilitator agent's server, facilitator agent's server accessible network port list, firewalls, user lists, and security options regarding the access of server resources accessible to the facilitator agent.

A simplified walk through of operations involved in creating a client agent, a client agent initiating a service request, a client agent responding to a service request and a facilitator agent responding to a service request are including hereafter by way of illustrating the use of such a system. These figures and their accompanying discussion are provided by way of illustration of one preferred embodiment of the present invention and are not intended to limit the scope of the present invention.

Figure 8 depicts operations involved in instantiating a client agent with its parent facilitator in accordance with a preferred embodiment of the present invention. The operations begin with starting the Agent Registration in a step 800. In a next step 802, the Installer, such as a client or facilitator agent, invokes a new client agent. It will be appreciated that any computer entity is capable of invoking a new agent. The system then instantiates the new client agent in a step 804. This operation may involve resource allocations somewhere in the network on a local computer system for the client agent, which will often include memory as well as placement of references to the newly instantiated client agent in internal system lists of agents within that local computing system. Once instantiated, the new client and its parent facilitator establish a communications link in a step 806. In certain preferred embodiments, this communications link involves selection of one or more physical transport mechanisms for this communication. Once established, the client agent transmits it profile to the parent facilitator in a step 808. When received, the parent facilitator registers the client agent in a step 810. Then, at a step 812, a client agent has been instantiated in accordance with one preferred embodiment of the present invention.

Figure 9 depicts operations involved in a client agent initiating a service request and receiving the response to that service request in accordance with a preferred embodiment of the present invention. The method of Figure 9 begins in a step 900, wherein any initialization or other such procedures may be performed. Then, in a step 902, the client agent determines a goal to be achieved (or solved). This goal is then translated in a step 904 into *ICL*, if it is not already formulated in it. The goal, now stated in *ICL*, is then transmitted to the client agent's parent facilitator 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, 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.

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 delegation is possible because agents' capabilities (solvables) 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.

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 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 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 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 *interpretation* of a goal involves the coordination of requests to the satisfying agents, and assembling their responses into a coherent whole, for return to the requester.

A further preferred embodiment of present invention extends facilitation so the facilitator can employ strategies and advice given by the requesting agent, resulting in a variety of interaction patterns that may be instantiated in the satisfaction of a request.

A further preferred embodiment of present invention handles the distribution of both data update requests and requests for installation of triggers, preferably using some of the same strategies that are employed in the delegation of service requests.

Note that the reliance on facilitation is not absolute; that is, there is no hard requirement that requests and services be matched up by the facilitator, or that interagent communications go through the facilitator. There is preferably support in the agent library for explicit addressing of requests. However, a preferred embodiment of the present invention encourages employment the paradigm of agent communities, minimizing their development effort, by taking advantage of the facilitator's provision of transparent delegation and handling of compound goals.

A facilitator is preferably viewed as a *coordinator*, not a controller, of cooperative task completion. A facilitator preferably never initiates an activity. A facilitator preferably responds to requests to manage the satisfaction of some goal, the update of some data repository, or the installation of a trigger by the appropriate agent or agents. All agents can preferably take advantage of the facilitator's expertise in delegation, and its up-to-date knowledge about the current membership of a dynamic community. The facilitator's coordination services often allows the developer to lessen the complexity of individual agents, resulting in a more manageable software development process, and enabling the creation of lightweight agents.

Maintaining Data Repositories

The agent library supports the creation, maintenance, and use of databases, in the form of data solvables. Creation of a data solvable requires only that it be declared. Querying a data solvable, as with access to any solvable, is done using *oaa_Solve*.

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 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 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 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 (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 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 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: "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 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 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 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."

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 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 *intra*agent and *inter*agent 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 *intra*agent with *inter*agent 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.

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.

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 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. 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.

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.

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:
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 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 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:
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.

1.4 FIND ALL NEW MAIL MESSAGES.
There are 2 messages available.
Message 1, from Mark Tierny, entitled "OAA meeting."
1.5 NEXT MESSAGE
Message 2, from Jennifer Schwefler, entitled "Presentation Summary."
1.6 PLAY IT.
This message is a multipart MIME-encoded message. There are two parts.
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
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.".

```
print(Object, Parameters) {
   ' If Object is reference to "it", find an appropriate
document
   if (Object = "ref(it)")
      oaa_Solve(resolve_reference(the, document, Params,
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,
[], 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))
      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
     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
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 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 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 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 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, 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.
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.
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 hotel.
Photo appears.
2.7 [Points to another hotel]
Photo appears.
2.8 [Speaking] Price of the other hotel?
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]
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.

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.

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 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.
        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*.

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 MVIEWS 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 sequence of exchanges between two agents, the facilitator assist 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 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.

IN THE CLAIMS:

1. A computer-implemented method for communication and cooperative task completion among a plurality of distributed electronic agents, comprising the acts of:

registering a description of each active client agent's functional capabilities, using an expandable, platform-independent, inter-agent language;

receiving a request for service as a base goal in the inter-agent language, in the form of an arbitrarily complex goal expression; and

dynamically interpreting the goal expression, said act of interpreting further 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, based on a match between the sub-goal being dispatched and the registered functional capabilities of the selected client agent.

2. A computer-implemented method as recited in claim 1, further including the following acts of:

receiving a new request for service as a base goal using the inter-agent language, in the form of another arbitrarily complex goal expression, from at least one of the selected client agents in response to the sub-goal dispatched to said agent; and

recursively applying the last step of claim 1 in order to perform the new request for service.

3. A computer implemented method as recited in claim 2 wherein the act of registering a specific agent further includes:

invoking the specific agent in order to activate the specific agent;

instantiating an instance of the specific agent; and

transmitting the new agent profile from the specific agent to the facilitator agent in response to the instantiation of the specific agent.

4. A computer implemented method as recited in claim 1 further including the act of deactivating a specific client agent no longer available to provide services by deleting the registration of the specific client agent.

5. A computer implemented method as recited in claim 1 further comprising the act of providing an agent registry data structure.

6. A computer implemented method as recited in claim 5 wherein the agent registry data structure includes at least one symbolic name for each active agent.

7. A computer implemented method as recited in claim 5 wherein the agent registry data structure includes at least one data declaration for each active agent.

8. A computer implemented method as recited in claim 5 wherein the agent registry data structure includes at least one trigger declaration for one active agent.

9. A computer implemented method as recited in claim 5 wherein the agent registry data structure includes at least one task declaration, and process characteristics for each active agent.

10. A computer implemented method as recited in claim 5 wherein the agent registry data structure includes at least one process characteristic for each active agent.

11. A computer implemented method as recited in claim 1 further comprising the act of establishing communication between the plurality of distributed agents.

12. A computer implemented method as recited in claim 1 further comprising the acts of:

receiving a request for service in a second language differing from the inter-agent language;

selecting a registered agent capable of converting the second language into the inter-agent language; and

forwarding the request for service in a second language to the registered agent capable of converting the second language into the inter-agent language, implicitly requesting that such a conversion be performed and the results returned.

13. A computer implemented method as recited in claim 12 wherein the request includes a natural language query, and the registered agent capable of converting the second language into the inter-agent language service is a natural language agent.

14. A computer implemented method as recited in claim 13 wherein the natural language query was generated by a user interface agent.

15. A computer implemented method as recited in claim 1, wherein the base goal requires setting a trigger having conditional functionality and consequential functionality.

16. A computer implemented method as recited in claim 15 wherein the trigger is an outgoing communications trigger, the computer implemented method further including the acts of:

monitoring all outgoing communication events in order to determine whether a specific outgoing communication event has occurred; and

in response to the occurrence of the specific outgoing communication event, performing the particular action defined by the trigger.

17. A computer implemented method as recited in claim 15 wherein the trigger is an incoming communications trigger, the computer implemented method further including the acts of:

monitoring all incoming communication events in order to determine whether a specific incoming communication event has occurred; and

in response to the occurrence of a specific incoming communication event satisfying the trigger conditional functionality, performing the particular consequential functionality defined by the trigger.

18. A computer implemented method as recited in claim 15 wherein the trigger is a data trigger, the computer implemented method further including the acts of:

monitoring a state of a data repository; and

in response to a particular state event satisfying the trigger conditional functionality, performing the particular consequential functionality defined by the trigger.

19. A computer implemented method as recited in claim 15 wherein the trigger is a time trigger, the computer implemented method further including the acts of:

monitoring for the occurrence of a particular time condition; and

in response to the occurrence of a particular time condition satisfying the trigger conditional functionality, performing the particular consequential functionality defined by the trigger.

20. A computer implemented method as recited in claim 15 wherein the trigger is installed and executed within the facilitator agent.

21. A computer implemented method as recited in claim 15 wherein the trigger is installed and executed within a first service-providing agent.

22. A computer implemented method as recited in claim 15 wherein the conditional functionality of the trigger is installed on a facilitator agent.

23. A computer implemented method as recited in claim 22 wherein the consequential functionality is installed on a specific service-providing agent other than a facilitator agent.

24. A computer implemented method as recited in claim 15 wherein the conditional functionality of the trigger is installed on a specific service-providing agent other than a facilitator agent.

25. A computer implemented method as recited in claim 15 wherein the consequential functionality of the trigger is installed on a facilitator agent.

26. A computer implemented method as recited in claim 1 wherein the base goal is a compound goal having sub-goals separated by operators.

27. A computer implemented method as recited in claim 26 wherein the type of available operators includes a conjunction operator, a disjunction operator, and a conditional execution operator.

28. A computer implemented method as recited in claim 27 wherein the type of available operators further includes a parallel disjunction operator that indicates that disjunct goals are to be performed by different agents.

29. A computer program stored on a computer readable medium, the computer program executable to facilitate cooperative task completion within a distributed computing environment, the distributed computing environment including a plurality of autonomous electronic agents, the distributed computing environment supporting an Interagent Communication Language, the computer program comprising computer executable instructions for:

providing an agent registry that declares capabilities of service-providing electronic agents currently active within the distributed computing environment;

interpreting a service request in order to determine a base goal that may be a compound, arbitrarily complex base goal, the service request adhering to an Interagent Communication Language (ICL), the act of interpreting including the sub-acts of:

determining any task completion advice provided by the base goal, and

determining any task completion constraints provided by the base goal;

constructing a base goal satisfaction plan including the sub-acts of:

determining whether the requested service is available,

determining sub-goals required in completing the base goal,

selecting service-providing electronic agents from the agent registry suitable for performing the determined sub-goals, and

ordering a delegation of sub-goal requests to best complete the requested service; and

implementing the base goal satisfaction plan.

30. A computer program as recited in claim 29 wherein the computer executable instruction for providing an agent registry includes the following computer executable instructions for registering a specific service-providing electronic agent into the agent registry:

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, the new agent profile defining publicly available capabilities of the specific agent; and

registering the specific agent together with the new agent profile within the agent registry, thereby making available to the facilitator agent the capabilities of the specific agent.

31. A computer program as recited in claim 30 wherein the computer executable instruction for registering a specific agent further includes:

invoking the specific agent in order to activate the specific agent;

instantiating an instance of the specific agent; and

transmitting the new agent profile from the specific agent to the facilitator agent in response to the instantiation of the specific agent.

32. A computer program as recited in claim 29 wherein the computer executable instruction for providing an agent registry includes a computer executable instruction for removing a specific service-providing electronic agent from the registry upon determining that the specific agent is no longer available to provide services.

33. A computer program as recited in claim 29 wherein the provided agent registry includes a symbolic name, a unique address, data declarations, trigger declarations, task declarations, and process characteristics for each active agent.

34. A computer program as recited in claim 29 further including computer executable instructions for receiving the service request via a communications link established with a client.

35. A computer program as recited in claim 29 wherein the computer executable instruction for providing a service request includes instructions for:

receiving a non-ICL format service request;

selecting an active agent capable of converting the non-ICL formal service request into an ICL format service request;

forwarding the non-ICL format service request to the active agent capable of converting the non-ICL format service request, together with a request that such conversion be performed; and

receiving an ICL format service request corresponding to the non-ICL format service request.

36. A computer program as recited in claim 35 wherein the non-ICL format service request includes a natural language query, and the active agent capable of converting the non-ICL formal service request into an ICL format service request is a natural language agent.

37. A computer program as recited in claim 36 wherein the natural language query is generated by a user interface agent.

38. A computer program as recited in claim 29, the computer program further including computer executable instructions for implementing a base goal that requires setting a trigger having conditional and consequential functionality.

39. A computer program as recited in claim 38 wherein the trigger is an outgoing communications trigger, the computer program further including computer executable instructions for:

monitoring all outgoing communication events in order to determine whether a specific outgoing communication event has occurred; and

in response to the occurrence of the specific outgoing communication event, performing the particular action defined by the trigger.

40. A computer program as recited in claim 38 wherein the trigger is an incoming communications trigger, the computer program further including computer executable instructions for:

monitoring all incoming communication events in order to determine whether a specific incoming communication event has occurred; and

in response to the occurrence of the specific incoming communication event, performing the particular action defined by the trigger.

41. A computer program as recited in claim 38 wherein the trigger is a data trigger, the computer program further including computer executable instructions for:

monitoring a state of a data repository; and

in response to a particular state event, performing the particular action defined by the trigger.

42. A computer program as recited in claim 38 wherein the trigger is a time trigger, the computer program further including computer executable instructions for:

monitoring for the occurrence of a particular time condition; and

in response to the occurrence of the particular time condition, performing the particular action defined by the trigger.

43. A computer program as recited in claim 38 further including computer executable instructions for installing and executing the trigger within the facilitator agent.

44. A computer program as recited in claim 38 further including computer executable instructions for installing and executing the trigger within a first service-providing agent.

45. A computer program as recited in claim 29 further including computer executable instructions for interpreting compound goals having sub-goals separated by operators.

46. A computer program as recited in claim 45 wherein the type of available operators includes a conjunction operator, a disjunction operator, and a conditional execution operator.

47. A computer program as recited in claim 46 wherein the type of available operators further includes a parallel disjunction operator that indicates that disjunct goals are to be performed by different agents.

48. An Interagent Communication Language (ICL) providing a basis for facilitated cooperative task completion within a distributed computing environment having a facilitator agent and a plurality of autonomous service-providing electronic agents, the ICL enabling agents to perform queries of other agents, exchange information with other agents, set triggers within other agents, 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 disjunctive operator parallel disjunctive operator that indicates that disjunct goals are to be performed by different agents.

49. An ICL as recited in claim 48, wherein the ICL is computer platform independent.

50. An ICL as recited in claim 48 wherein the ICL is independent of computer programming languages which the plurality of agents are programmed in.

51. An ICL as recited in claim 48 wherein the ICL syntax supports explicit task completion constraints within goal expressions.

52. An ICL as recited in claim 51 wherein possible types of task completion constraints include use of specific agent constraints and response time constraints.

53. An ICL as recited in claim 51 wherein the ICL syntax supports explicit task completion advisory suggestions within goal expressions.

54. An ICL as recited in claim 48 wherein the ICL syntax supports explicit task completion advisory suggestions within goal expressions.

55. An ICL as recited in claim 48 wherein each autonomous service-providing electronic agent defines and publishes a set of capability declarations or solvables, expressed in ICL, that describes services provided by such electronic agent.

56. An ICL as recited in claim 55 wherein an electronic agent's solvables define an interface for the electronic agent.

57. An ICL as recited in claim 56 wherein the facilitator agent maintains an agent registry making available a plurality of electronic agent interfaces.

58. An ICL as recited in claim 57 wherein the possible types of solvables includes procedure solvables, a procedure solvable operable to implement a procedure such as a test or an action.

59. An ICL as recited in claim 58 wherein the possible types of solvables further includes data solvables, a data solvable operable to provide access to a collection of data.

60. An ICL as recited in claim 58 wherein the possible types of solvables includes data solvables, a data solvable operable to provide access to a collection of data.

61. A facilitator agent arranged to coordinate cooperative task completion within a distributed computing environment having a plurality of autonomous service-providing electronic agents, the facilitator agent comprising:

an agent registry that declares capabilities of service-providing electronic agents currently active within the distributed computing environment; and

a facilitating engine operable to parse a service request in order to interpret a compound goal set forth therein, the compound goal including both local and global constraints and control parameters, the service request formed according to an Interagent Communication Language (ICL), the facilitating engine further operable to construct a goal satisfaction plan specifying the coordination of a suitable delegation of sub-goal requests to complete the requested service satisfying both the local and global constraints and control parameters.

62. A facilitator agent as recited in claim 61, wherein the facilitating engine is capable of modifying the goal satisfaction plan during execution, the modifying initiated by events such as new agent declarations within the agent registry, decisions made by remote agents, and information provided to the facilitating engine by remote agents.

63. A facilitator agent as recited in claim 61 wherein the agent registry includes a symbolic name, a unique address, data declarations, trigger declarations, task declarations, and process characteristics for each active agent.

64. A facilitator agent as recited in claim 61 wherein the facilitating engine is operable to install a trigger mechanism requesting that a certain action be taken when a certain set of conditions are met.

65. A facilitator agent as recited in claim 64 wherein the trigger mechanism is a communication trigger that monitors communication events and performs the certain action when a certain communication event occurs.

66. A facilitator agent as recited in claim 64 wherein the trigger mechanism is a data trigger that monitors a state of a data repository and performs the certain action when a certain data state is obtained.

67. A facilitator agent as recited in claim 66 wherein the data repository is local to the facilitator agent.

68. A facilitator agent as recited in claim 66 wherein the data repository is remote from the facilitator agent.

69. A facilitator agent as recited in claim 64 wherein the trigger mechanism is a task trigger having a set of conditions.

70. A facilitator agent as recited in claim 61, the facilitator agent further including a global database accessible to at least one of the service-providing electronic agents.

71. A software-based, flexible computer architecture for communication and cooperation among distributed electronic agents, the architecture contemplating a distributed computing system comprising:

a plurality of service-providing electronic agents; and

a facilitator agent in bi-directional communications with the plurality of service-providing electronic agents, the facilitator agent including:

an agent registry that declares capabilities of service-providing electronic agents currently active within the distributed computing environment;

a facilitating engine operable to parse a service request in order to interpret an arbitrarily complex goal set forth therein, the facilitating engine further operable to construct a goal satisfaction plan including the coordination of a suitable delegation of sub-goal requests to best complete the requested service.

72. A computer architecture as recited in claim 71, wherein the basis for the computer architect is an Interagent Communication Language (ICL) enabling agents to perform queries of other agents, exchange information with other agents, and set triggers within other agents, the ICL further defined by 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 disjunctive operator parallel disjunctive operator that indicates that disjunct goals are to be performed by different agents.

73. A computer architecture as recited in claim 72, wherein the ICL is computer platform independent.

74. A computer architecture as recited in claim 73 wherein the ICL is independent of computer programming languages in which the plurality of agents are programmed.

75. A computer architecture as recited in claim 73 wherein the ICL syntax supports explicit task completion constraints within goal expressions.

76. A computer architecture as recited in claim 75 wherein possible types of task completion constraints include use of specific agent constraints and response time constraints.

77. A computer architecture as recited in claim 75 wherein the ICL syntax supports explicit task completion advisory suggestions within goal expressions.

78. A computer architecture as recited in claim 73 wherein the ICL syntax supports explicit task completion advisory suggestions within goal expressions.

79. A computer architecture as recited in claim 73 wherein each autonomous service-providing electronic agent defines and publishes a set of capability declarations or solvables, expressed in ICL, that describes services provided by such electronic agent.

80. A computer architecture as recited in claim 79 wherein an electronic agent's solvables define an interface for the electronic agent.

81. A computer architecture as recited in claim 80 wherein the possible types of solvables includes procedure solvables, a procedure solvable operable to implement a procedure such as a test or an action.

82. A computer architecture as recited in claim 81 wherein the possible types of solvables further includes data solvables, a data solvable operable to provide access to a collection of data.

83. A computer architecture as recited in claim 82 wherein the possible types of solvables includes a data solvable operable to provide access to modify a collection of data.

84. A computer architecture as recited in claim 71 wherein the planning component of the facilitating engine are distributed across at least two computer processes.

85. A computer architecture as recited in claim 71 wherein the execution component of the facilitating engine is distributed across at least two computer processes.

86. A data wave carrier providing a transport mechanism for information communication in a distributed computing environment having at least one facilitator agent and at least one active client agent, the data wave carrier comprising a signal representation of an inter-agent language description of an active client agent's functional capabilities.

87. A data wave carrier as recited in claim 85, the data wave carrier further comprising a signal representation of a request for service in the inter-agent language from a first agent to a second agent.

88. A data wave carrier as recited in claim 85, the data wave carrier further comprising a signal representation of a goal dispatched to an agent for performance from a facilitator agent.

89. A data wave carrier as recited in claim 88 wherein a later state of the data wave carrier comprises 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.

Software-Based Architecture for Communication and Cooperation Among Distributed Electronic Agents

ABSTRACT

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 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 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 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.

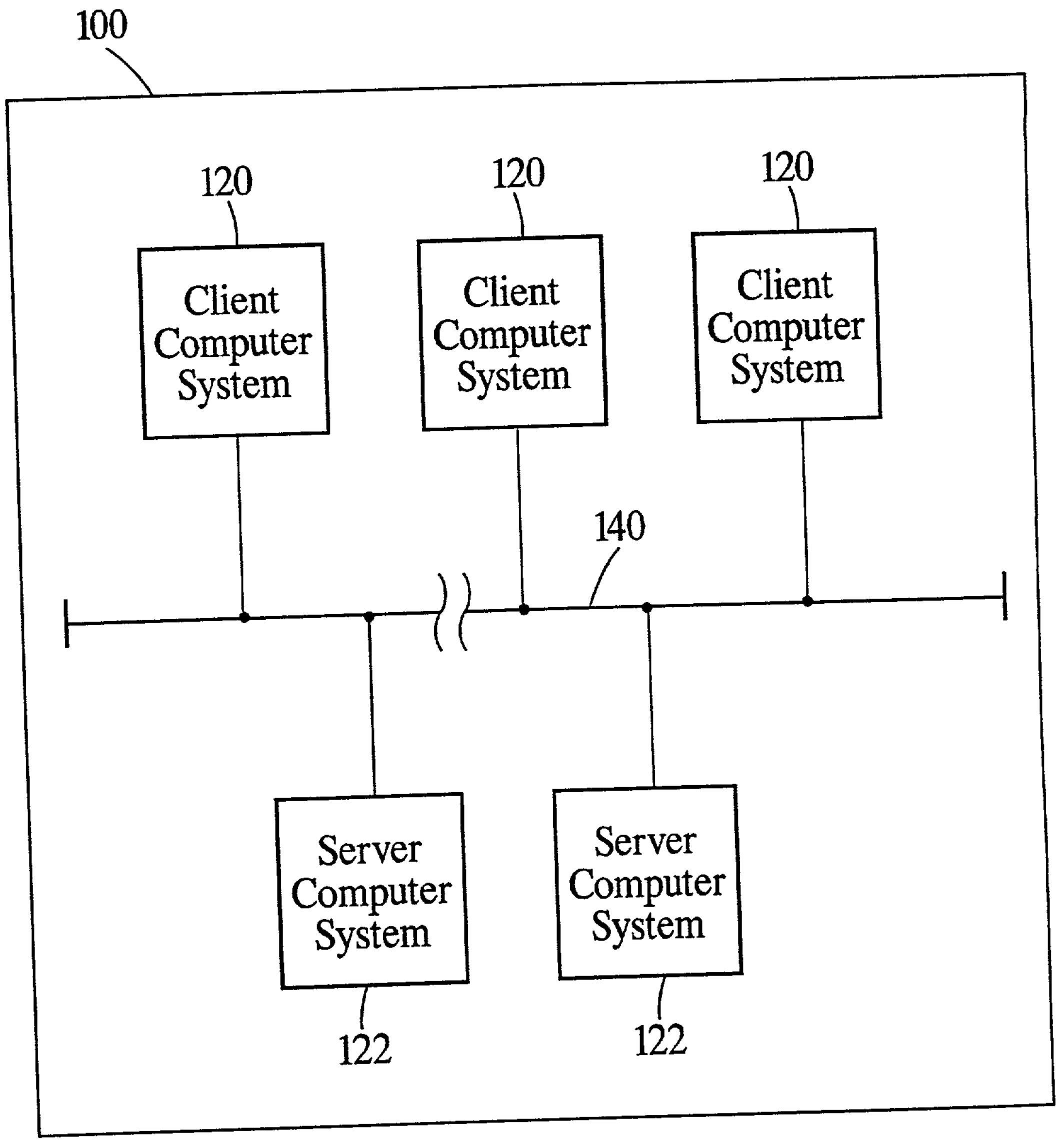


Fig. 1
(Prior Art)

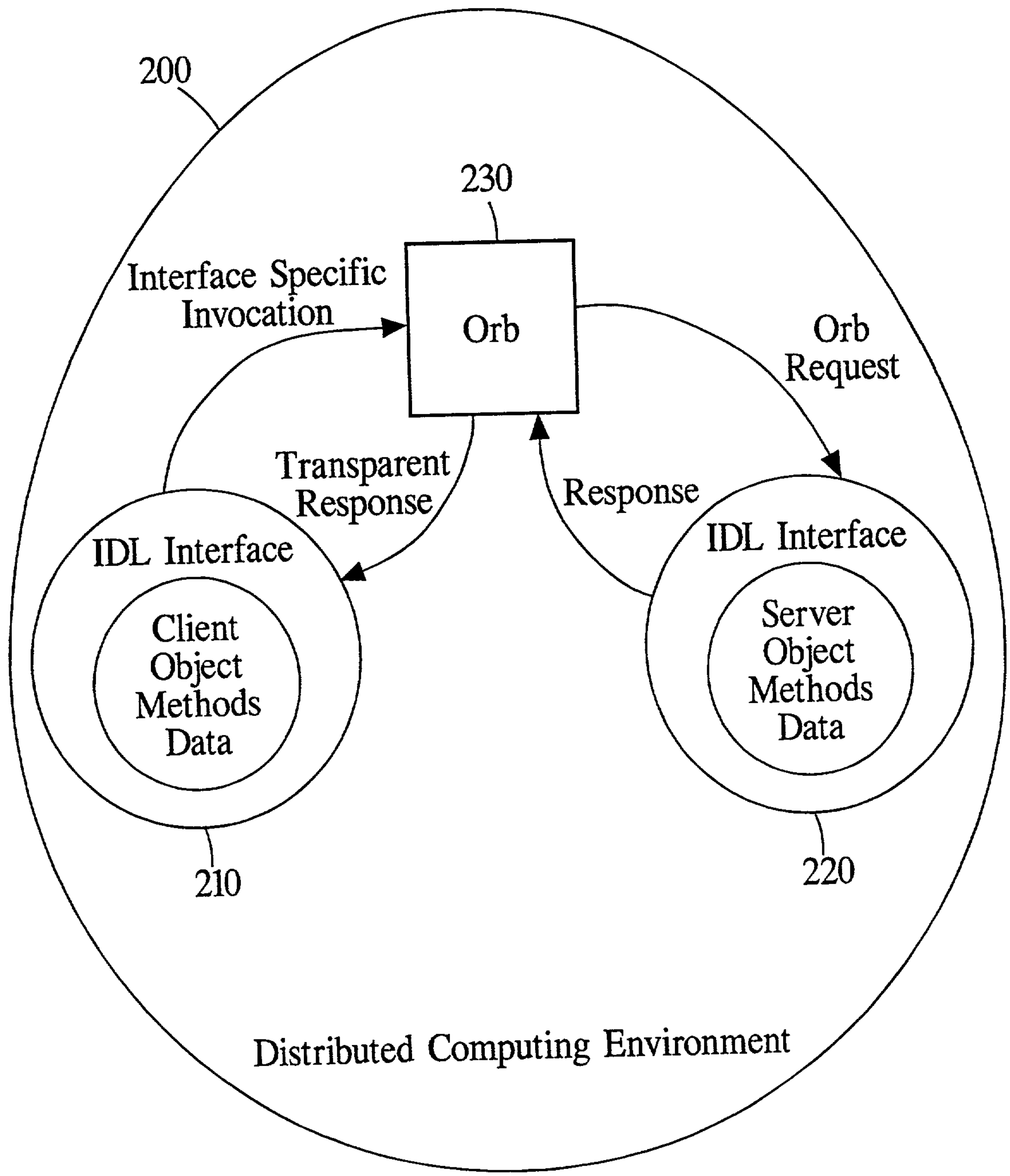


Fig. 2
(Prior Art)

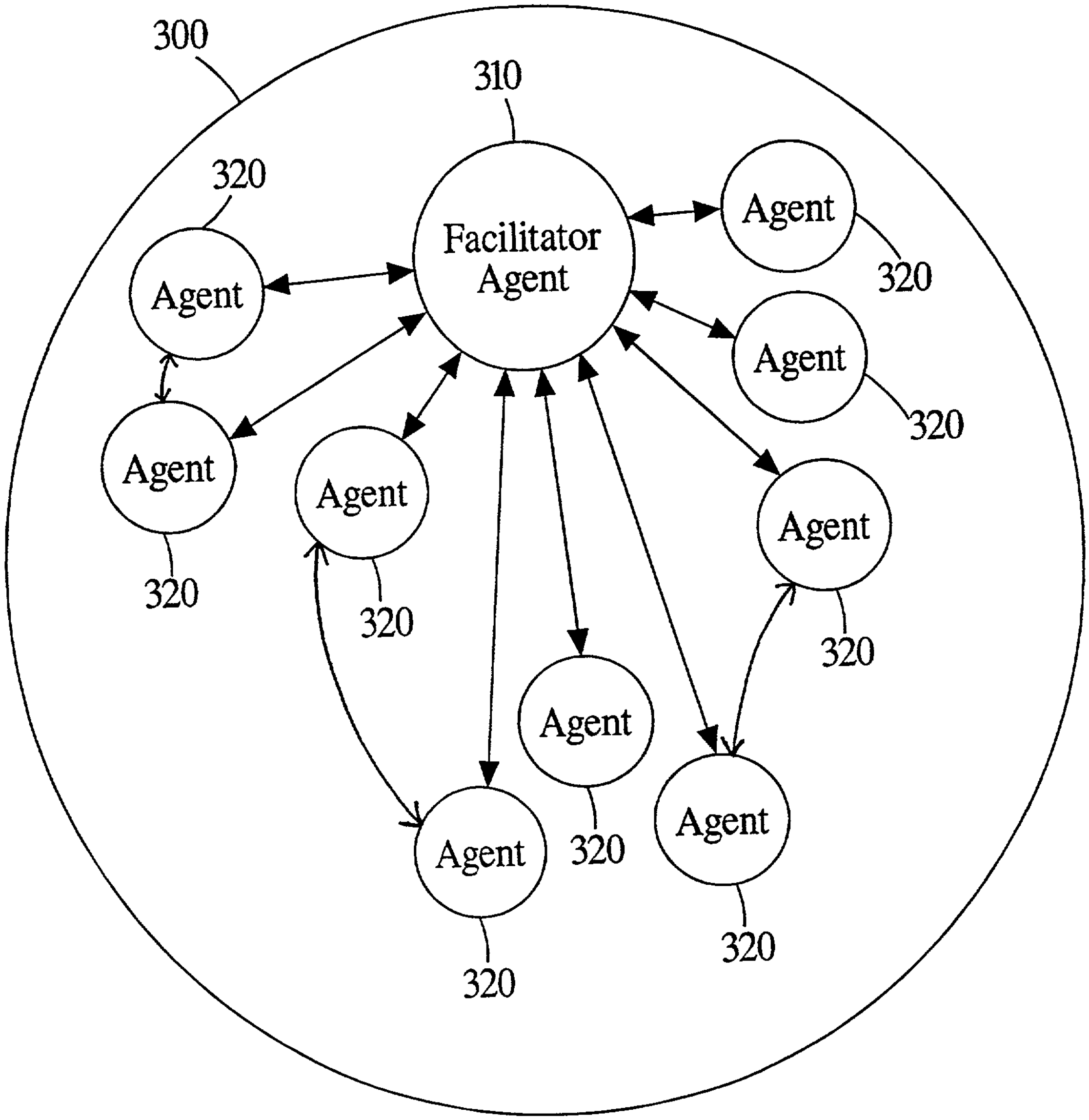

300
310
Facilitator Agent
Agent
Agent
Agent
Agent
Agent
Agent
Agent
Agent
Agent
Agent
320
320
320
320
320
320
320
320
320
320

Fig. 3

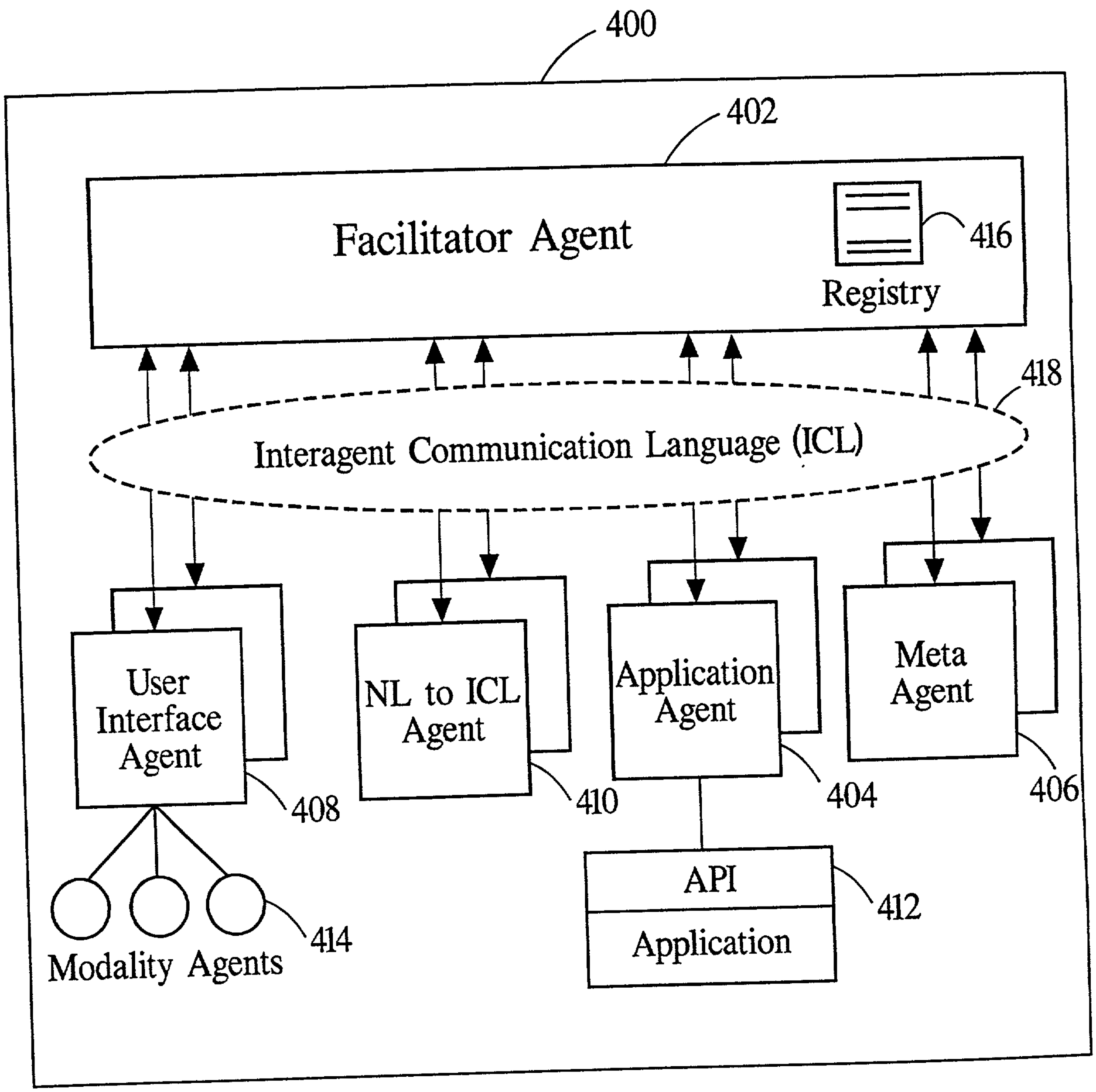

400
402
Facilitator Agent
416
Registry
418
Interagent Communication Language (ICL)
User Interface Agent
408
NL to ICL Agent
410
Application Agent
404
Meta Agent
406
API
Application
412
414
Modality Agents

Fig. 4

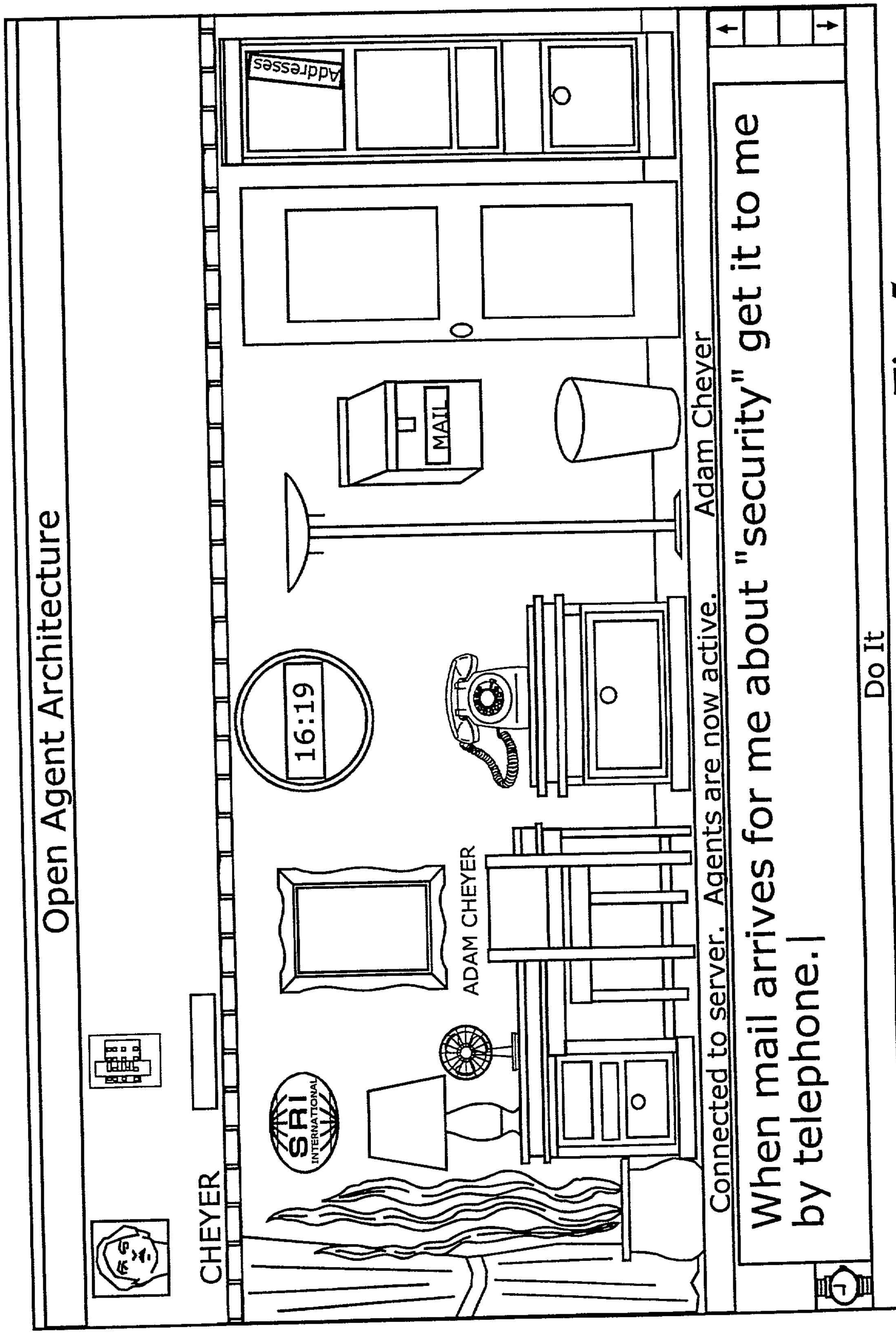

Open Agent Architecture
CHEYER
SRI
INTERNATIONAL
ADAM CHEYER
16:19
MAIL
Addresses
Adam Cheyer
Connected to server. Agents are now active.
When mail arrives for me about "security" get it to me by telephone.|
Do It

Fig. 5

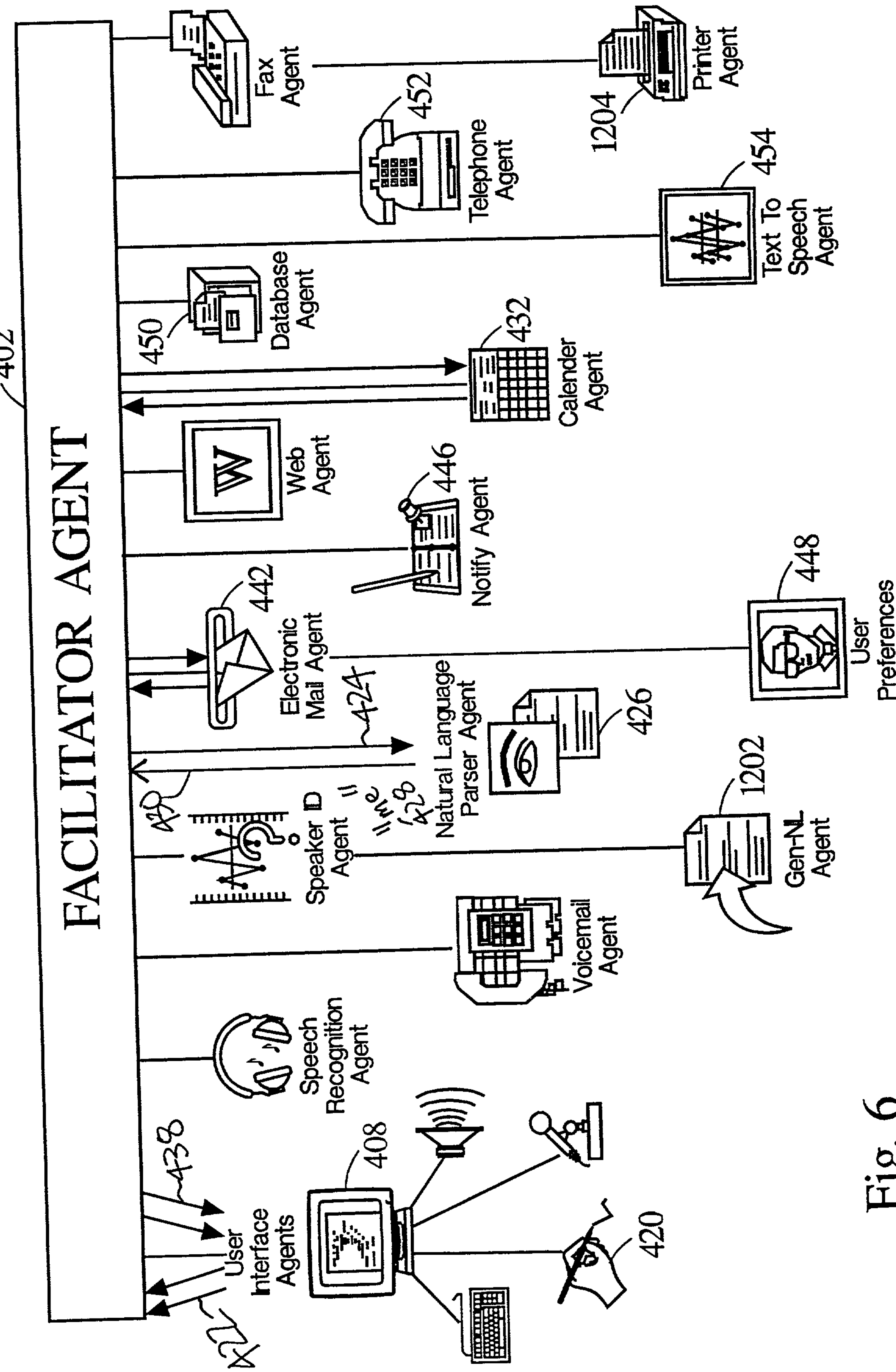

FACILITATOR AGENT
402
Fax Agent
Telephone Agent
452
Printer Agent
1204
Text To Speech Agent
454
Database Agent
450
Calender Agent
432
Web Agent
Notify Agent
446
Electronic Mail Agent
442
424
Natural Language Parser Agent
426
User Preferences
448
Speaker ID Agent
Gen-NL Agent
1202
Voicemail Agent
Speech Recognition Agent
User Interface Agents
408
438
420

Fig. 6

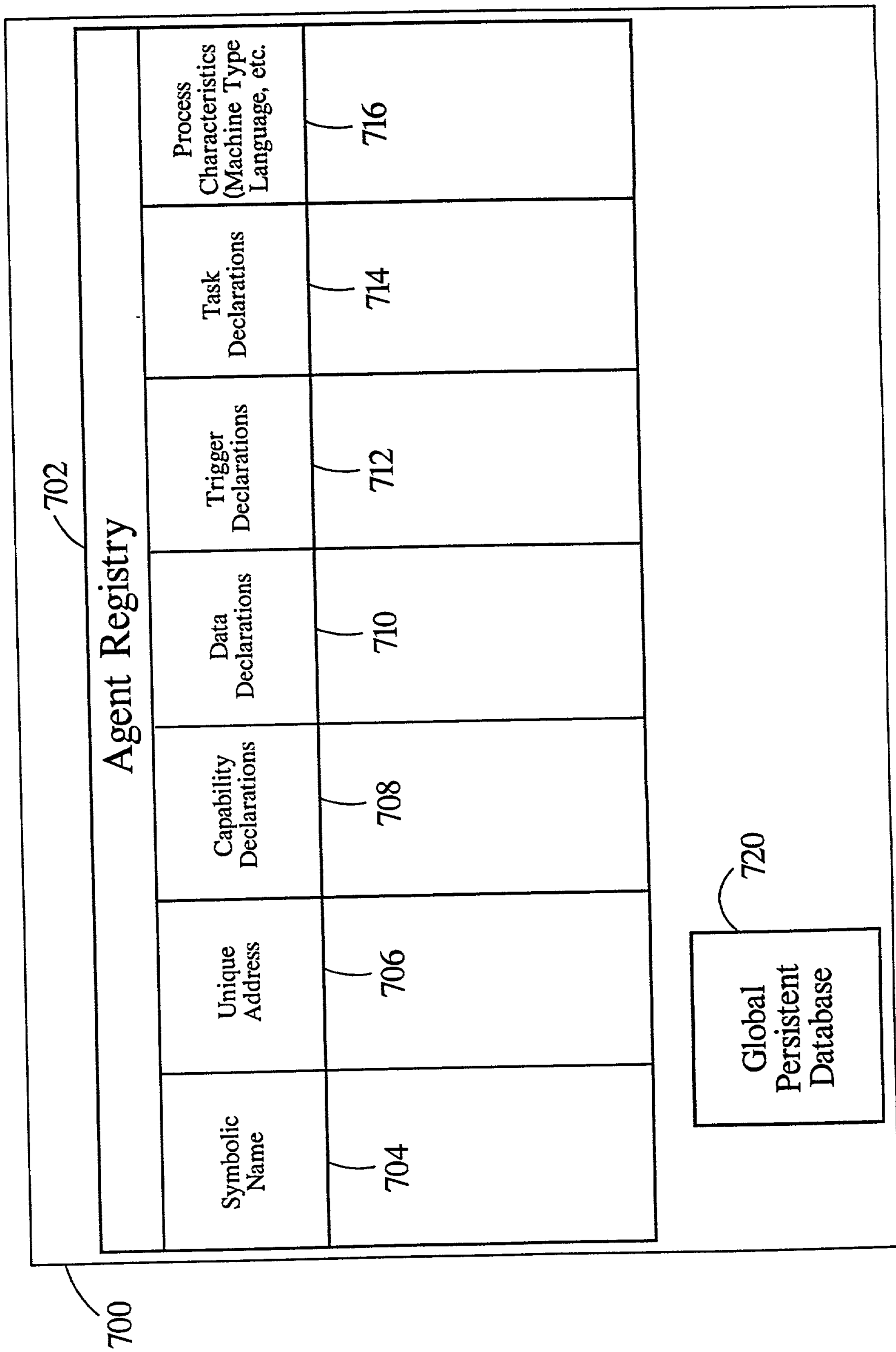

Agent Registry
702
Symbolic Name
704
Unique Address
706
Capability Declarations
708
Data Declarations
710
Trigger Declarations
712
Task Declarations
714
Process Characteristics (Machine Type Language, etc.
716
Global Persistent Database
720
700

Fig. 7

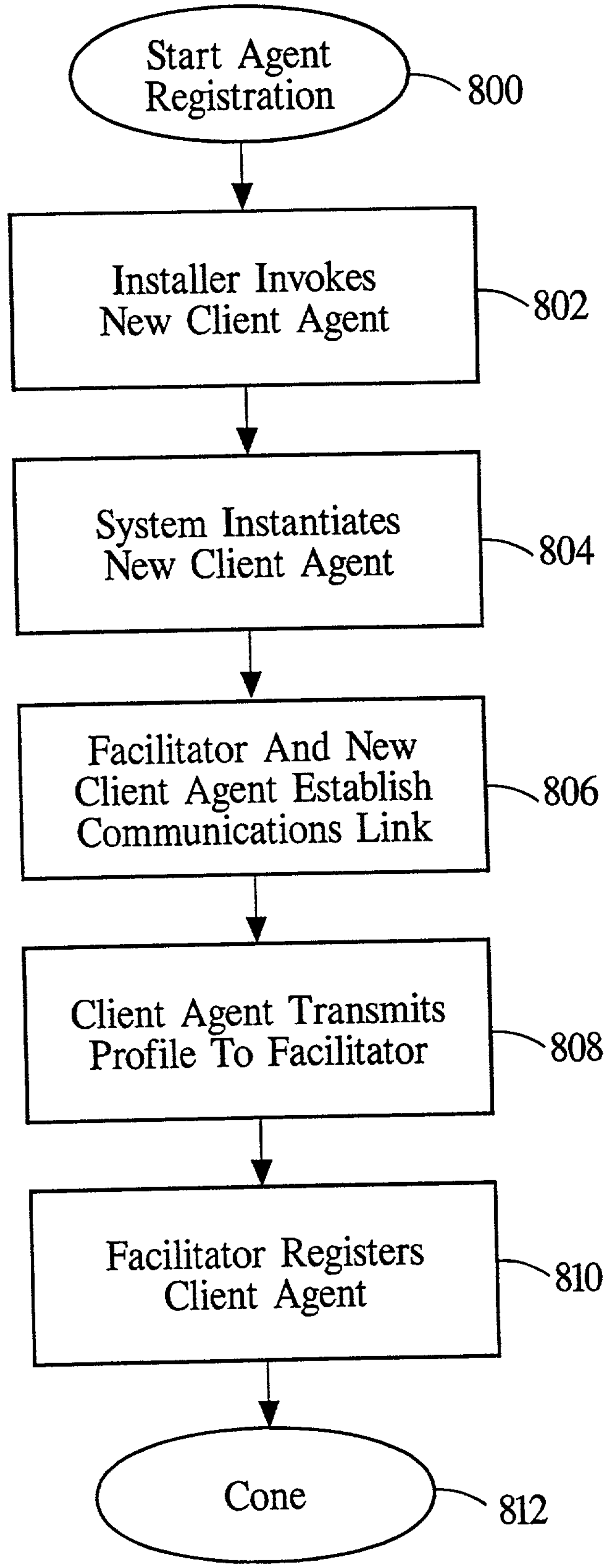

Start Agent Registration
800
Installer Invokes New Client Agent
802
System Instantiates New Client Agent
804
Facilitator And New Client Agent Establish Communications Link
806
Client Agent Transmits Profile To Facilitator
808
Facilitator Registers Client Agent
810
Cone
812

Fig. 8

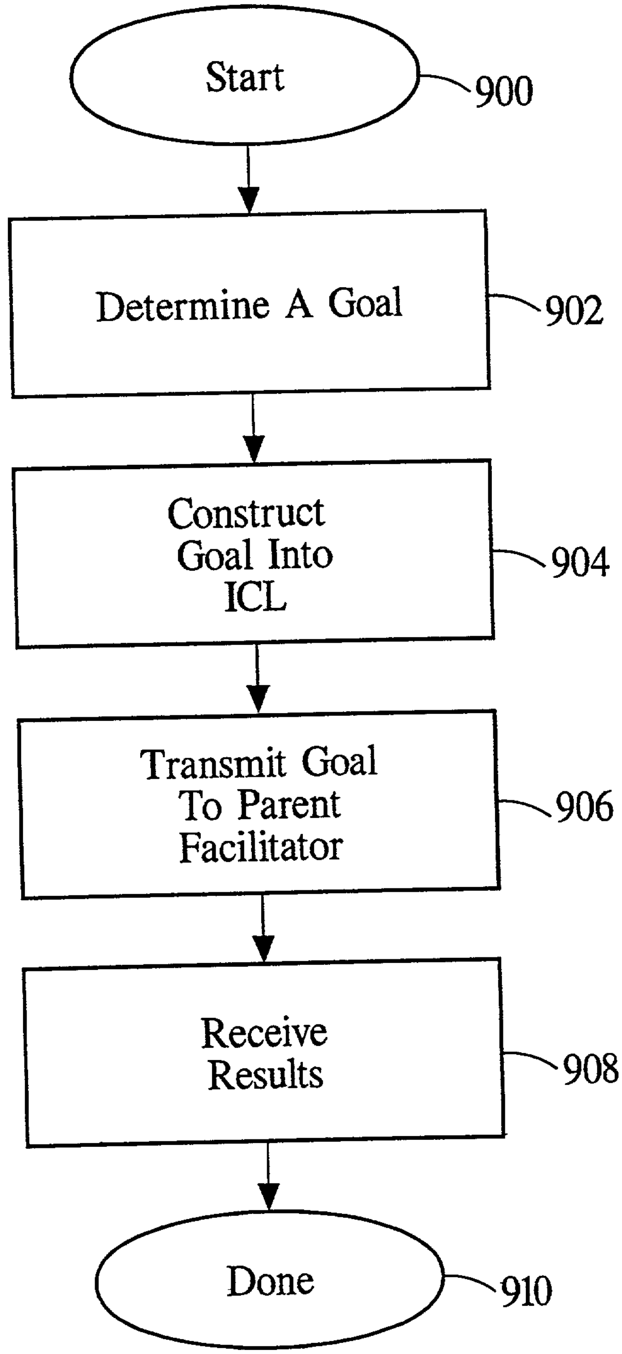

Start
900
Determine A Goal
902
Construct Goal Into ICL
904
Transmit Goal To Parent Facilitator
906
Receive Results
908
Done
910

Fig. 9

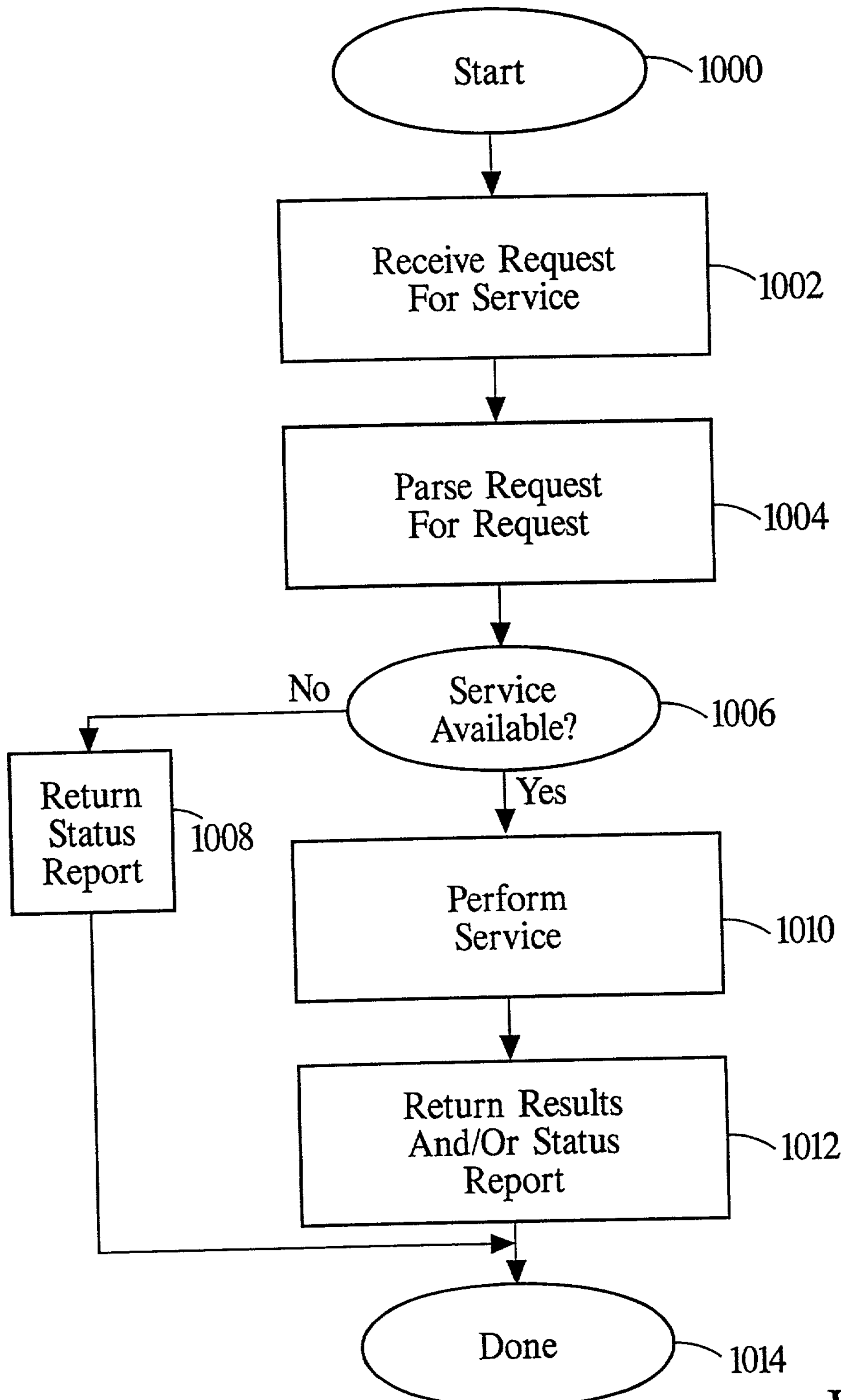

Start
1000
Receive Request For Service
1002
Parse Request For Request
1004
Service Available?
1006
No
Return Status Report
1008
Yes
Perform Service
1010
Return Results And/Or Status Report
1012
Done
1014

Fig. 10

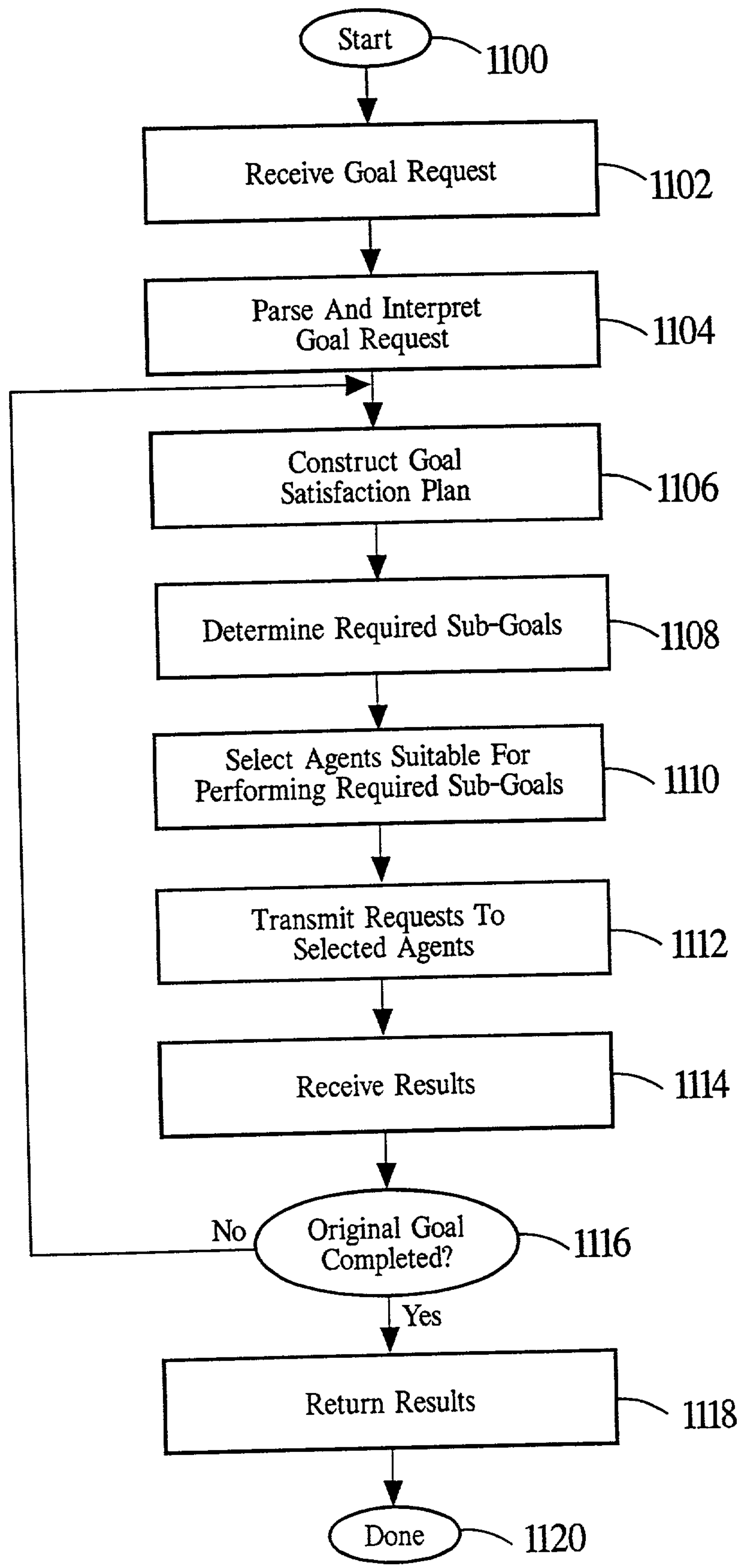

Start
1100
Receive Goal Request
1102
Parse And Interpret Goal Request
1104
Construct Goal Satisfaction Plan
1106
Determine Required Sub-Goals
1108
Select Agents Suitable For Performing Required Sub-Goals
1110
Transmit Requests To Selected Agents
1112
Receive Results
1114
Original Goal Completed?
No
Yes
1116
Return Results
1118
Done
1120

Fig. 11

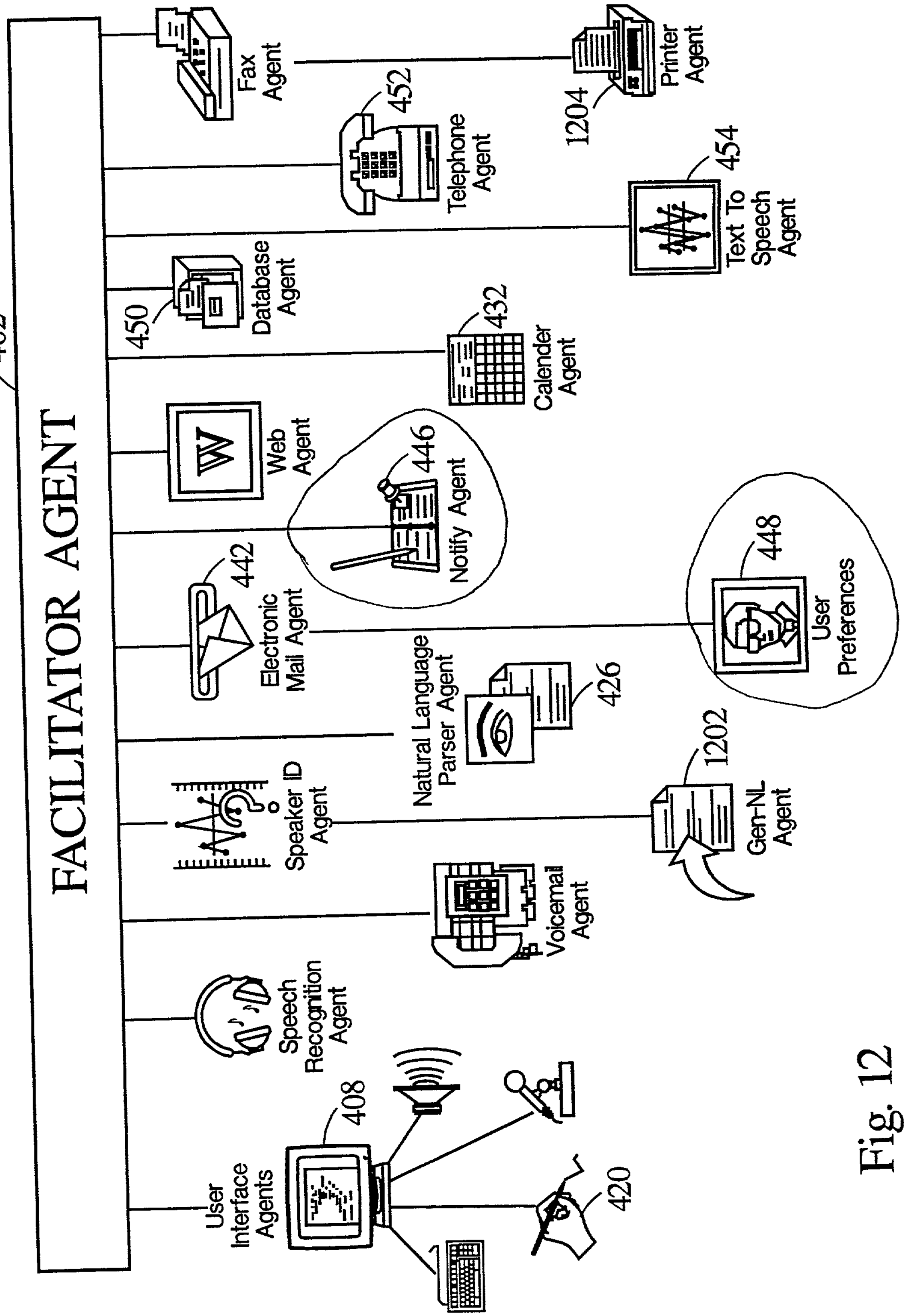

FACILITATOR AGENT
402
Fax Agent
Printer Agent
1204
Telephone Agent
452
Text To Speech Agent
454
Database Agent
450
Calender Agent
432
Web Agent
Notify Agent
446
Electronic Mail Agent
442
User Preferences
448
Natural Language Parser Agent
426
Speaker ID Agent
Gen-NL Agent
1202
Voicemail Agent
Speech Recognition Agent
User Interface Agents
408
420

Fig. 12

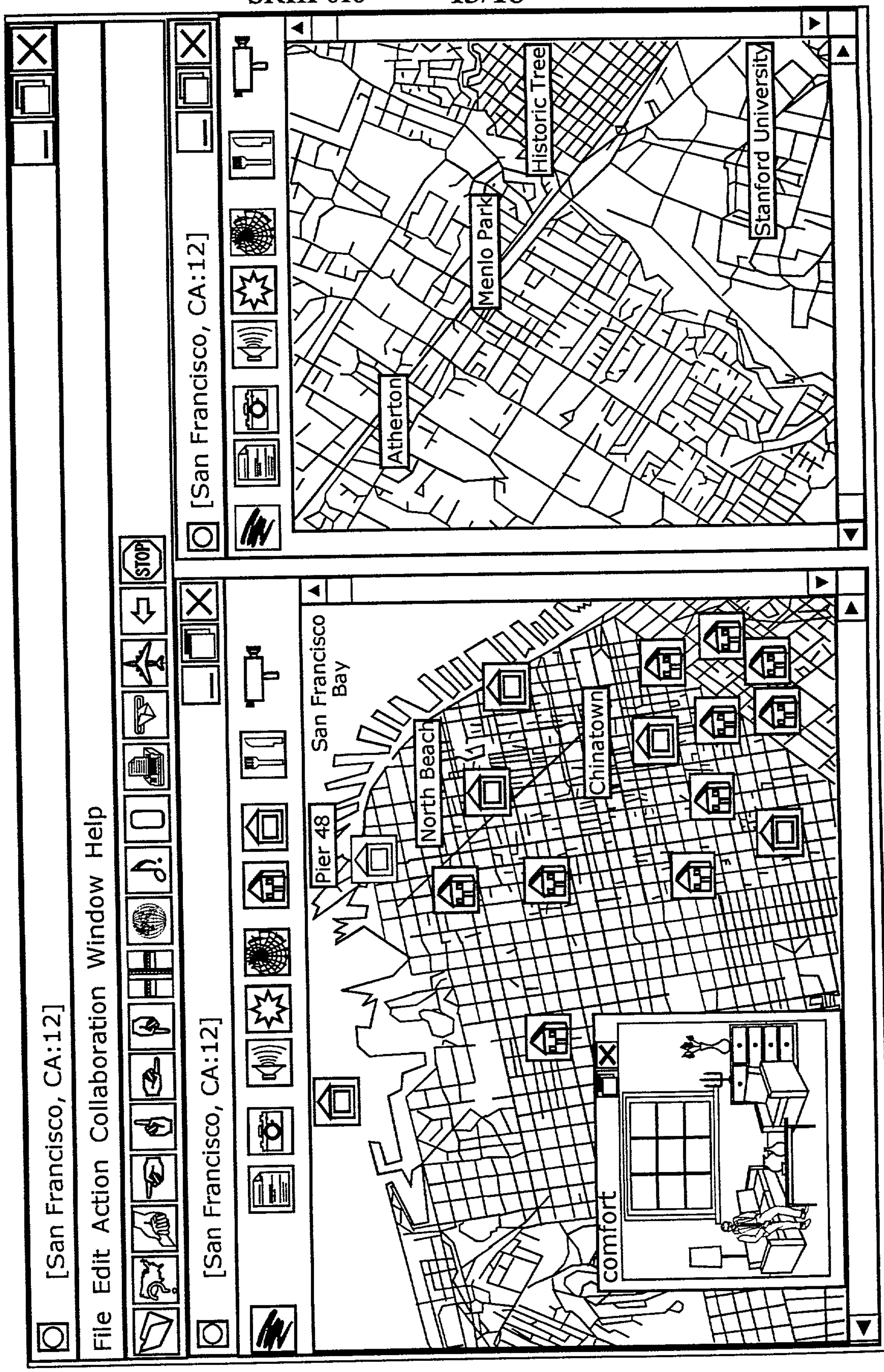

[San Francisco, CA:12]
File Edit Action Collaboration Window Help
STOP
[San Francisco, CA:12]
San Francisco Bay
Pier 48
North Beach
Chinatown
comfort
[San Francisco, CA:12]
Atherton
Menlo Park
Historic Tree
Stanford University

Fig. 13

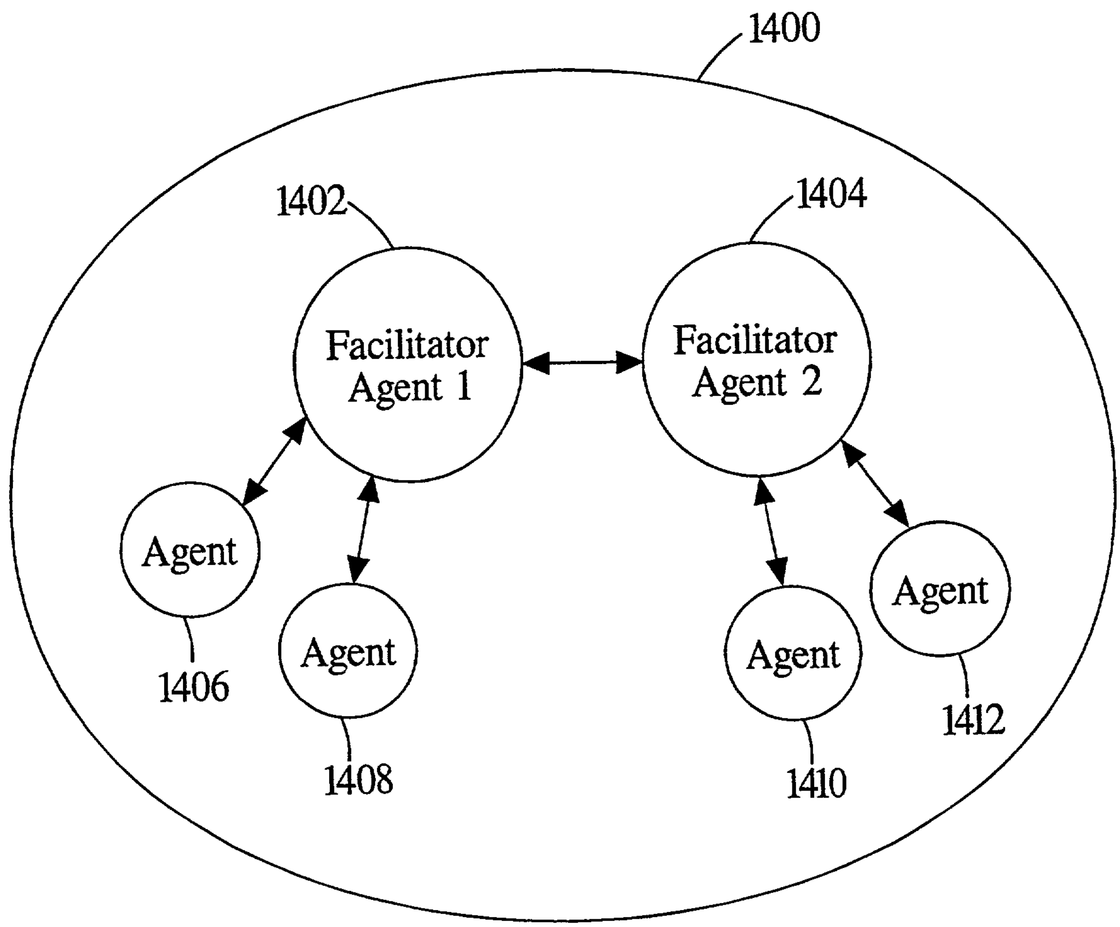

1400
1402
1404
Facilitator Agent 1
Facilitator Agent 2
Agent
Agent
Agent
Agent
1406
1408
1410
1412

Fig. 14

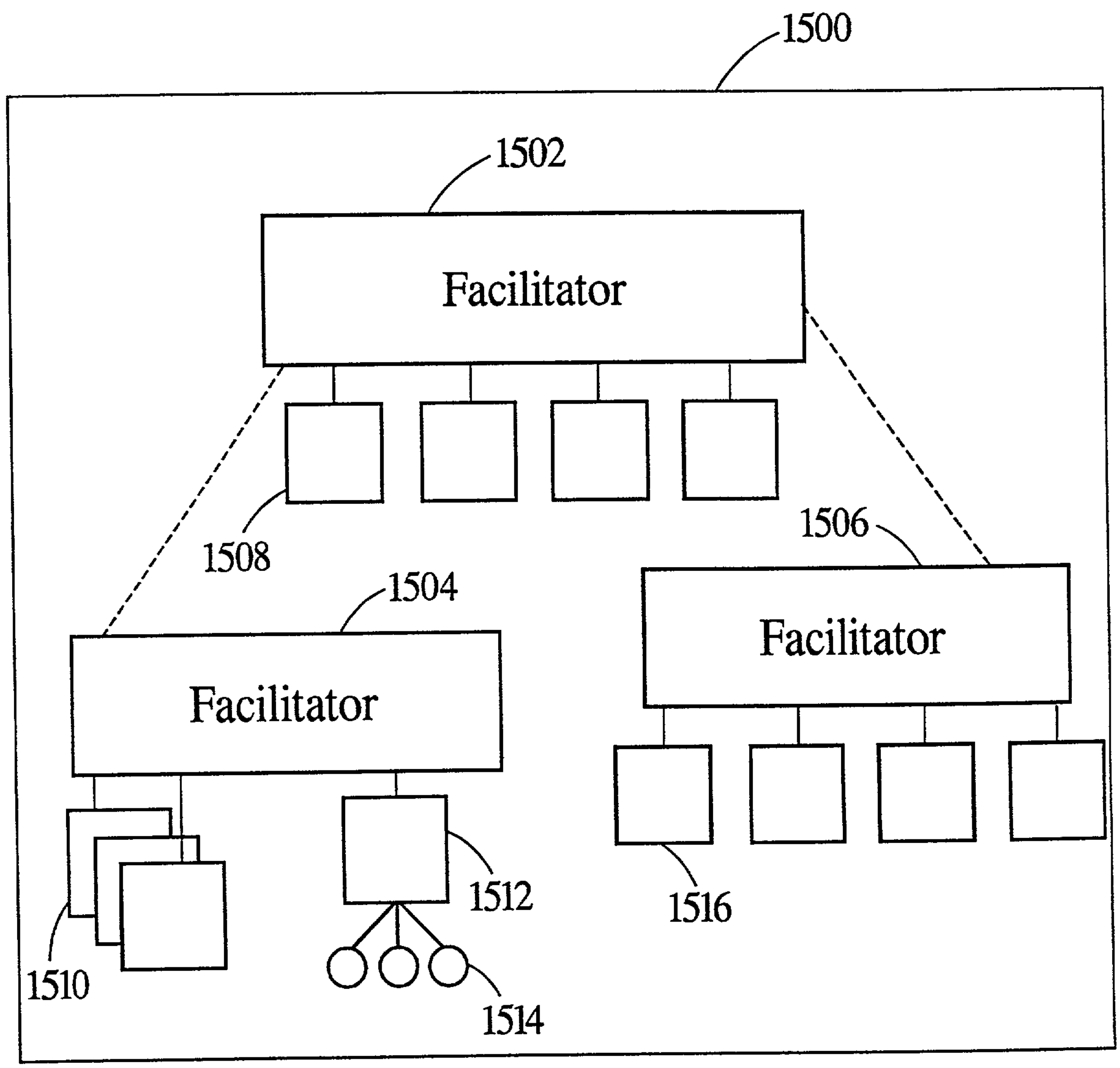

1500
1502
Facilitator
1508
1506
1504
Facilitator
Facilitator
1512
1516
1510
1514

Fig. 15

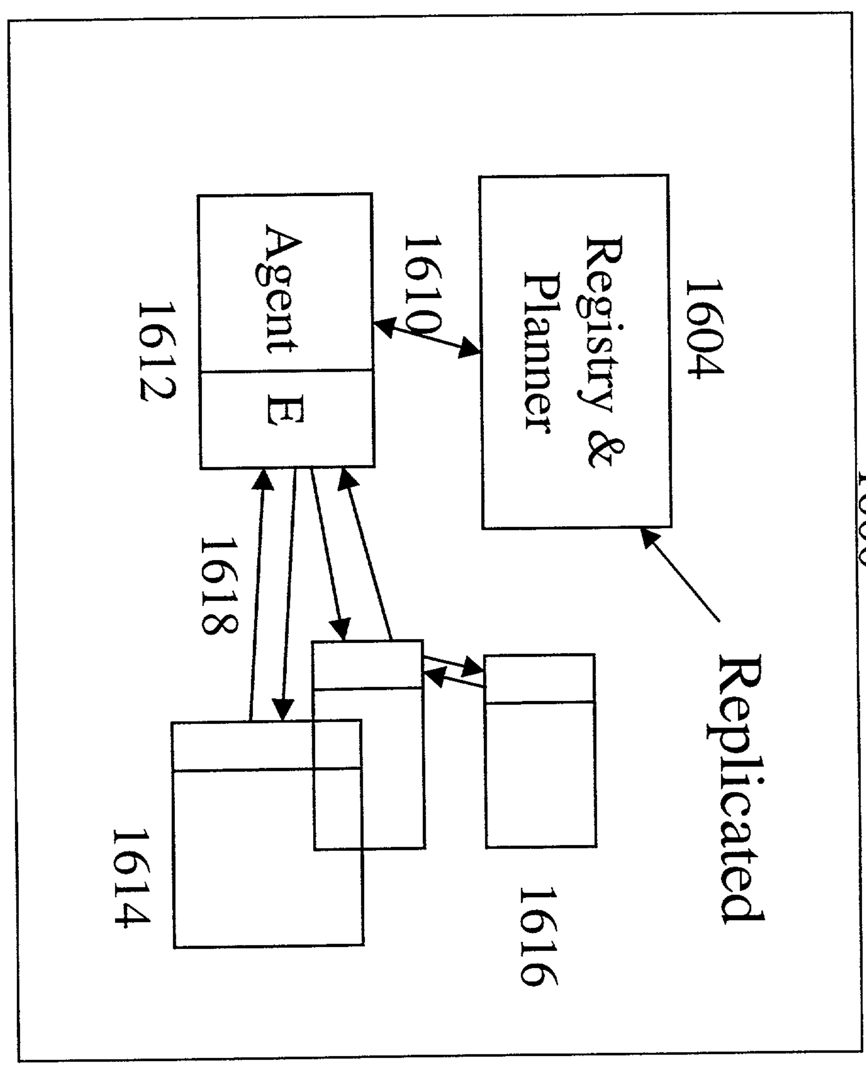

1600
Replicated
Registry & Planner
1604
1610
Agent
E
1612
1618
1614
1616

Figure 16

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 DLM Stephens & Coleman, LLP,

And I hereby appoint the law firm of Hickman ~~& Martine~~ 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.

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 CRONIN DR. Santa Clara, CA 95051

jc618 U.S. PTO
09/225198
01/05/99

Class	Subclass

ISSUE CLASSIFICATION

PATENT NUMBER

U.S. UTILITY PATENT APPLICATION

O.I.P.E.	PATENT DATE
TM SCANNED AGG Q.A. KB	

SECTOR	CLASS	SUBCLASS	ART UNIT	EXAMINER
	709	317	2151 2126	LCOCK

FILED WITH: ☐ DISK (CRF) ☐ FICHE
(Attached in pocket on right inside flap)

PREPARED AND APPROVED FOR ISSUE

ISSUING CLASSIFICATION

ORIGINAL		CROSS REFERENCE(S)						
CLASS	SUBCLASS	CLASS	SUBCLASS (ONE SUBCLASS PER BLOCK)					
INTERNATIONAL CLASSIFICATION								
/								
/								
/								
/								
/				☐ Continued on Issue Slip Inside File Jacket				

☐ TERMINAL DISCLAIMER	DRAWINGS			CLAIMS ALLOWED	
	Sheets Drwg.	Figs. Drwg.	Print Fig.	Total Claims	Print Claim for O.G.
☐ a) The term of this patent subsequent to ________ (date) has been disclaimed.	(Assistant Examiner) (Date)			NOTICE OF ALLOWANCE MAILED	
☐ b) The term of this patent shall not extend beyond the expiration date of U.S Patent. No. ________	(Primary Examiner) (Date)			ISSUE FEE: Amount Due	Date Paid
☐ c) The terminal ____months of this patent have been disclaimed.	(Legal Instruments Examiner) (Date)			ISSUE BATCH NUMBER	

WARNING:
The information disclosed herein may be restricted. Unauthorized disclosure may be prohibited by the United States Code Title 35, Sections 122, 181 and 368. Possession outside the U.S. Patent & Trademark Office is restricted to authorized employees and contractors only.

Form PTO-436A
(Rev. 6/98)

(LABEL AREA)

(FACE)

DISH, E

[redacted]

Best Available Copy

SEARCHED

Class	Sub.	Date	Exmr.
709	307 202	7/10/02	[illegible]
709	317 202	2/20/03	[illegible]
UPDATED		11/20/03	[illegible]

INTERFERENCE SEARCHED

Class	Sub.	Date	Exmr.

SEARCH NOTES
(INCLUDING SEARCH STRATEGY)

	Date	Exmr.
EAST WEST ACM IEEE INTERNET	7/10/02	[illegible]
UPDATED	2/20/03	[illegible]
UPDATED.	11/20/03	[illegible]

(RIGHT OUTSIDE)

ISSUE SLIP STAPLE AREA (for additional cross references)

POSITION	INITIALS	ID NO.	DATE
FEE DETERMINATION	LW	76534	01-19/99
O.I.P.E. CLASSIFIER		10	1/20
FORMALITY REVIEW	JC	71470	1/28/99

INDEX OF CLAIMS

✓ .... Rejected		N .... Non-elected	
= .... Allowed		I .... Interference	
— (Through numeral)... Canceled		A .... Appeal	
÷ .... Restricted		O .... Objected	

Claim: Final	Claim: Original	Date: 7/11/02	2/24/03	11/20/03							
	1	✓	✓	✓							
	2										
	3										
	4										
	5										
	6										
	7										
	8										
	9										
	10										
	11										
	12										
	13										
	14										
	15										
	16										
	17										
	18										
	19										
	20										
	21										
	22										
	23										
	24										
	25										
	26										
	27										
	28										
	29										
	30										
	31										
	32										
	33										
	34										
	35										
	36										
	37										
	38										
	39										
	40										
	41										
	42										
	43										
	44										
	45										
	46										
	47										
	48										
	49										
	50	✓	✓	✓							

Claim: Final	Claim: Original	Date: 7/11/02	2/24/03	11/20/03							
	51	✓	✓	✓							
	52										
	53										
	54										
	55										
	56										
	57										
	58										
	59										
	60										
	61										
	62										
	63										
	64										
	65										
	66										
	67										
	68										
	69										
	70										
	71										
	72										
	73										
	74										
	75										
	76										
	77										
	78										
	79										
	80										
	81										
	82										
	83										
	84										
	85										
	86										
	87										
	88										
	89	✓	✓	✓							
	90										
	91										
	92										
	93										
	94										
	95										
	96										
	97										
	98										
	99										
	100										

Claim: Final	Claim: Original	Date									
	101										
	102										
	103										
	104										
	105										
	106										
	107										
	108										
	109										
	110										
	111										
	112										
	113										
	114										
	115										
	116										
	117										
	118										
	119										
	120										
	121										
	122										
	123										
	124										
	125										
	126										
	127										
	128										
	129										
	130										
	131										
	132										
	133										
	134										
	135										
	136										
	137										
	138										
	139										
	140										
	141										
	142										
	143										
	144										
	145										
	146										
	147										
	148										
	149										
	150										

If more than 150 claims or 10 actions
staple additional sheet here

(LEFT INSIDE)

A

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF EXPRESS MAILING
I hereby certify that this paper and the documents and/or fees referred to as attached therein are being deposited with the United States Postal Service on January 05, 1999 in an envelope as "Express Mail Post Office to Addressee" service under 37 CFR §1.10, Mailing Label Number EL221766053US, addressed to the Assistant Commissioner for Patents, Washington, DC 20231.

Michael L. Gough

01/05/99 jc571 U.S. PTO

Attorney Docket No.: SRI1P016

First Named Inventor:

CHEYER, Adam J.

jc618 U.S. PTO 09/225198 01/05/99

UTILITY PATENT APPLICATION TRANSMITTAL (37 CFR § 1.53(b))

Assistant Commissioner for Patents
Box Patent Application
Washington, DC 20231

☐ Duplicate for fee processing

Sir: This is a request for filing a patent application under 37 CFR § 1.53(b) in the name of inventors: Adam J. Cheyer and David L. Martin

For: **SOFTWARE-BASED ARCHITECTURE FOR COMMUNICATION AND COOPERATION AMONG DISTRIBUTED ELECTRONIC AGENTS**

Application Elements:

☒ 59 Pages of Specification, Claims and Abstract
☒ 16 Sheets of Drawings
☒ 01 Pages Combined Declaration and Power of Attorney

Accompanying Application Parts:

☒ Assignment and Assignment Recordation Cover Sheet (recording fee not enclosed)
☒ Return Receipt Postcard

Fee Calculation (37 CFR § 1.16)

	(Col. 1) NO. FILED		(Col. 2) NO. EXTRA	SMALL ENTITY RATE	FEE	OR	LARGE ENTITY RATE	FEE
BASIC FEE				$395	$	OR	$760	$ 760.00
TOTAL CLAIMS	89	-20 =	69	x11 =	$	OR	x18 =	$1242.00
INDEP CLAIMS	06	-03 =	03	x41 =	$	OR	x78 =	$ 234.00
* If the difference in Col. 1 is less than zero, enter "0" in Col. 2.				Total	$	OR	Total	$2236.00

Including filing fees and the assignment recordation fee of $40.00, the Commissioner is authorized to charge all required fees to Deposit Account No. 50-0384 (Order No. SRI1P016).

☒ The Commissioner is authorized to charge any fees beyond the amount enclosed which may be required, or to credit any overpayment, to Deposit Account No. 50-0384 (Order No. SRI1P016).

General Authorization for Petition[illegible] Extension of Time (37 CFR §1.136)

☒ Applicants hereby make and generally authorize any Petitions for Extensions of Time as may be needed for any subsequent filings. The Commissioner is also authorized to charge any extension fees under 37 CFR §1.17 as may be needed to Deposit Account No. 50-0384.

☒ Please send correspondence to the following address:

Brian R. Coleman
HICKMAN STEPHENS & COLEMAN, LLP
P.O. Box 52037
Palo Alto, CA 94303-0746

Tel (650) 470-7430
Fax (650) 470-7440

Date: 1/5/99

Brian R. Coleman
Registration No. **39,145**

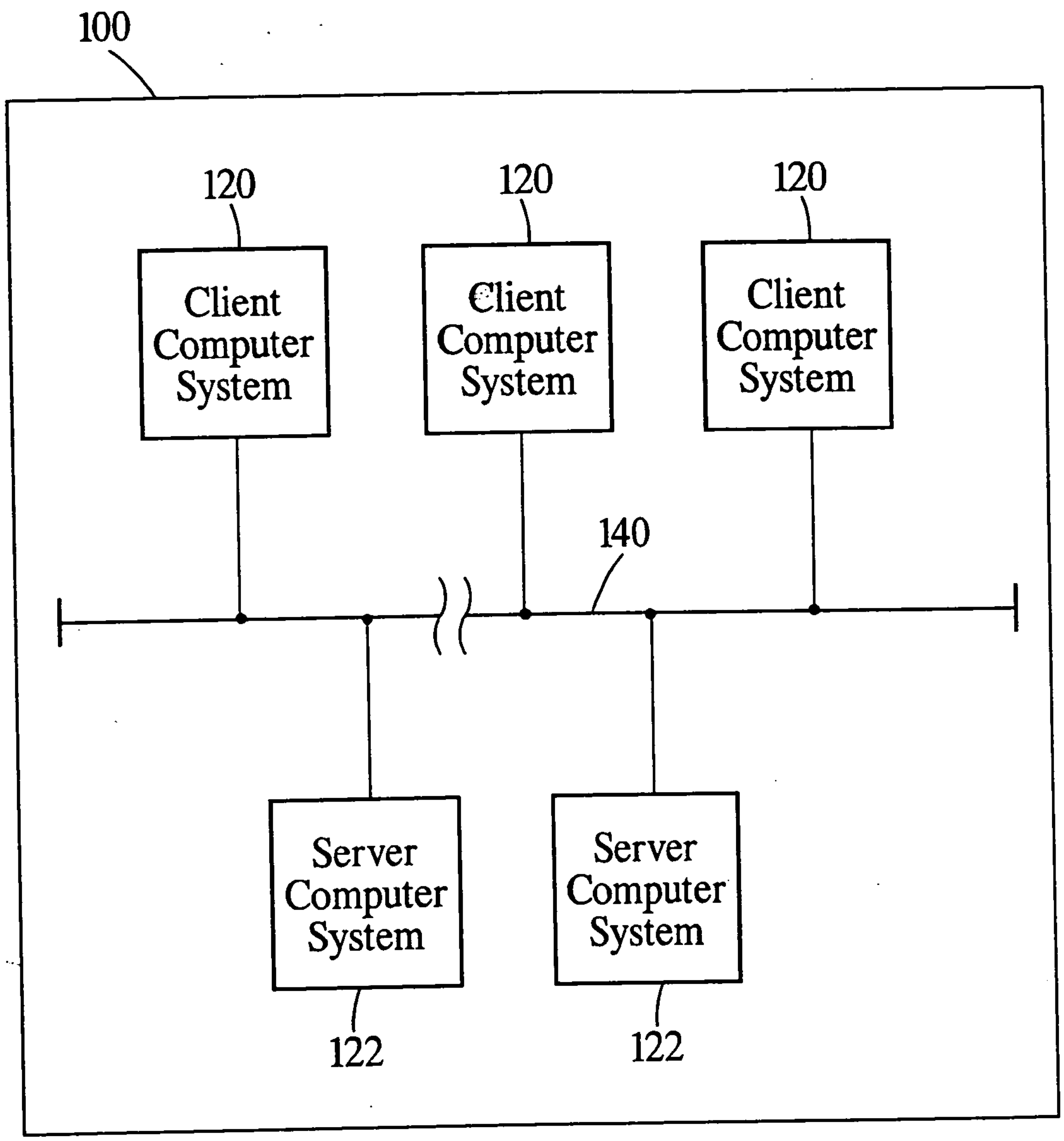


Fig. 1
(Prior Art)

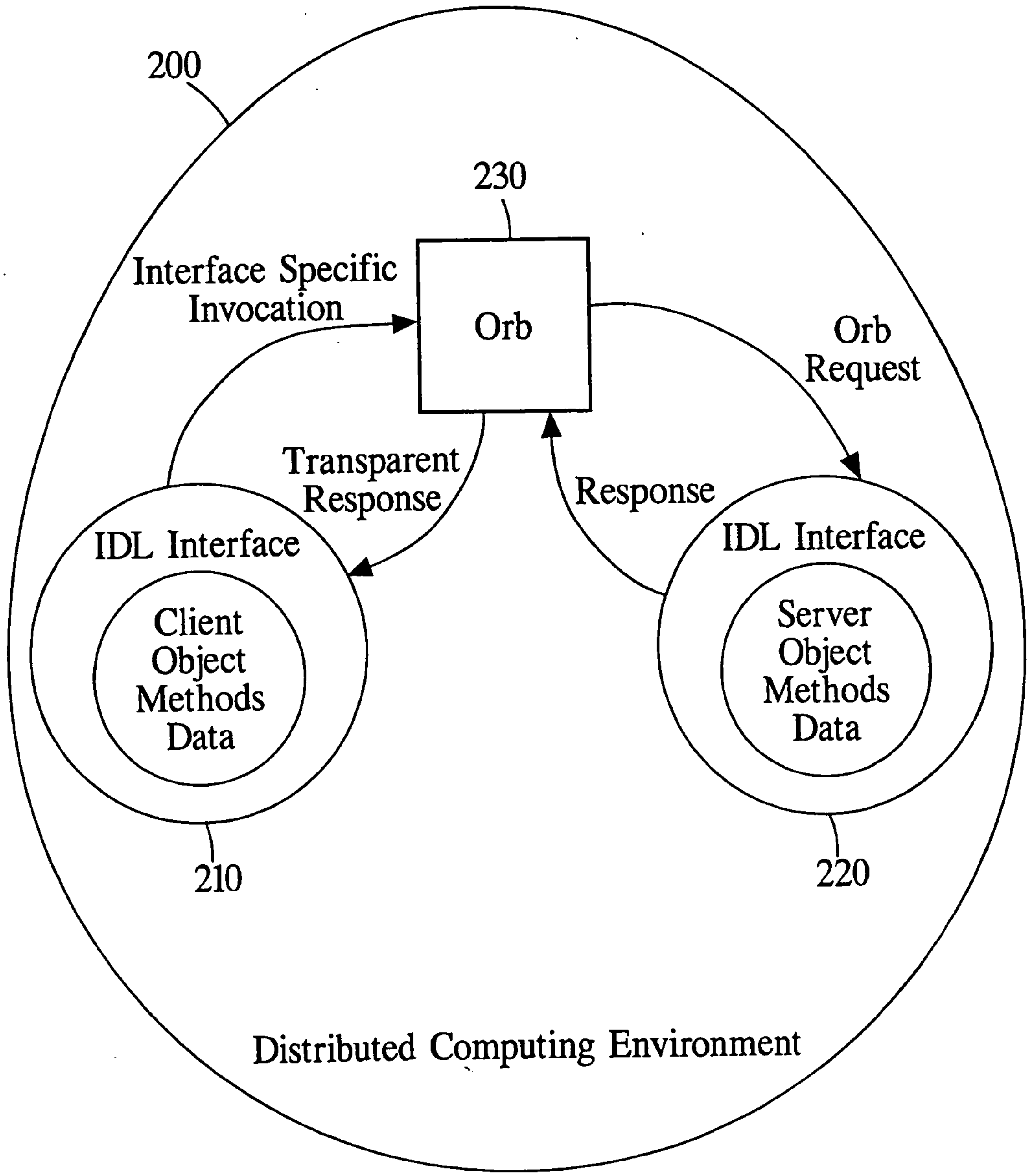


Fig. 2
(Prior Art)

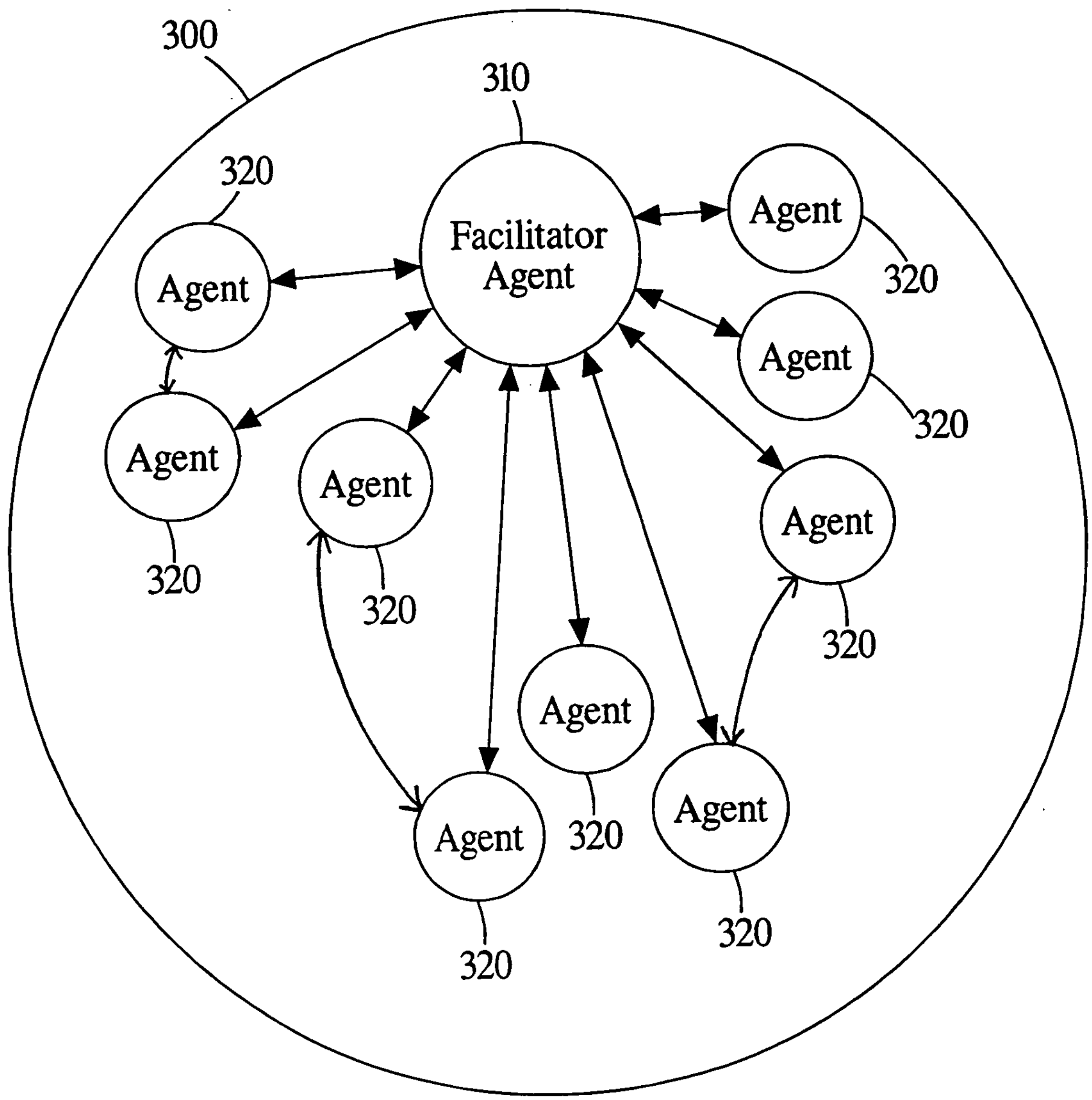


Fig. 3

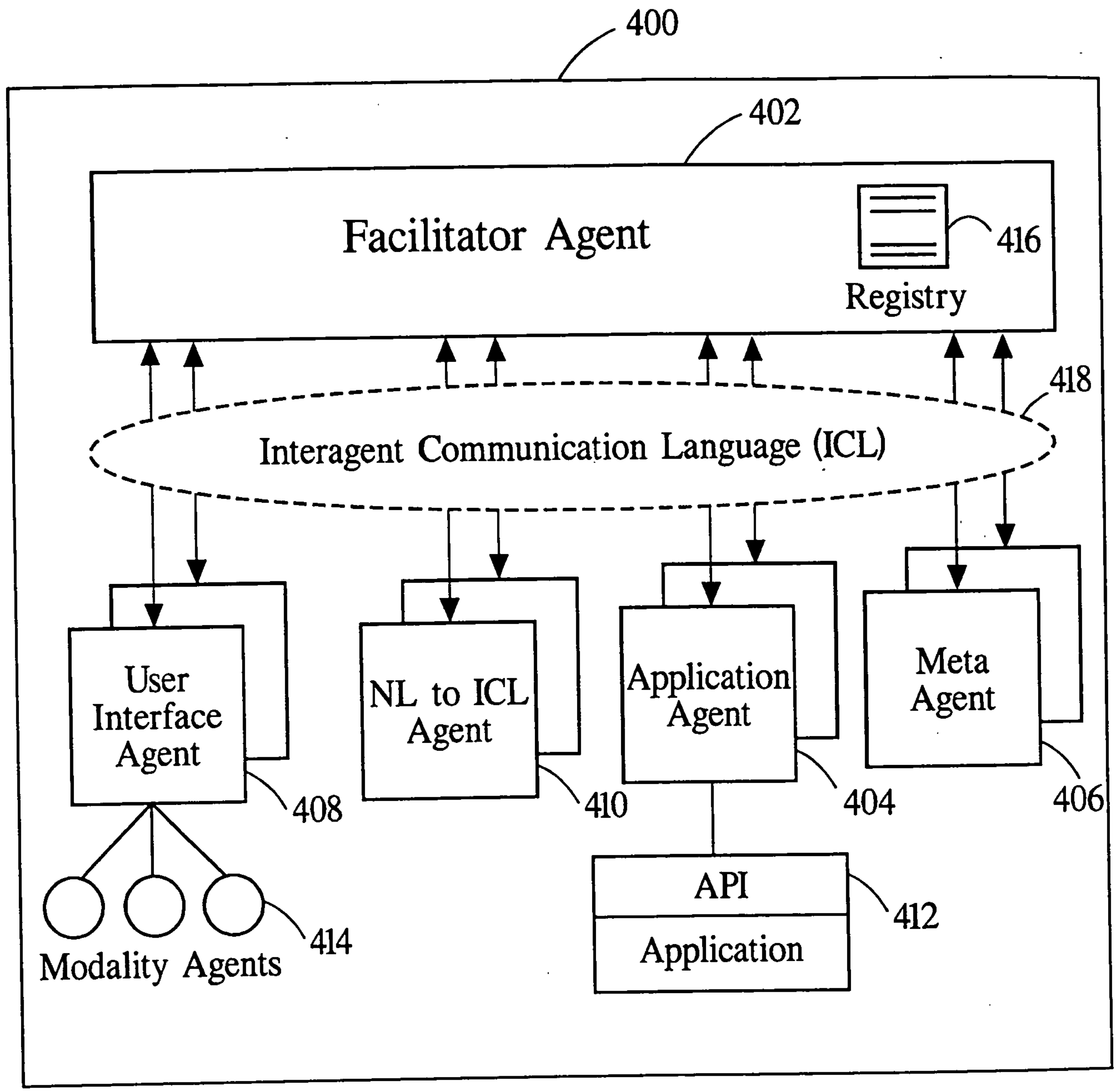

400
402
Facilitator Agent
416
Registry
418
Interagent Communication Language (ICL)
User Interface Agent
408
NL to ICL Agent
410
Application Agent
404
Meta Agent
406
Modality Agents
414
API
Application
412

Fig. 4

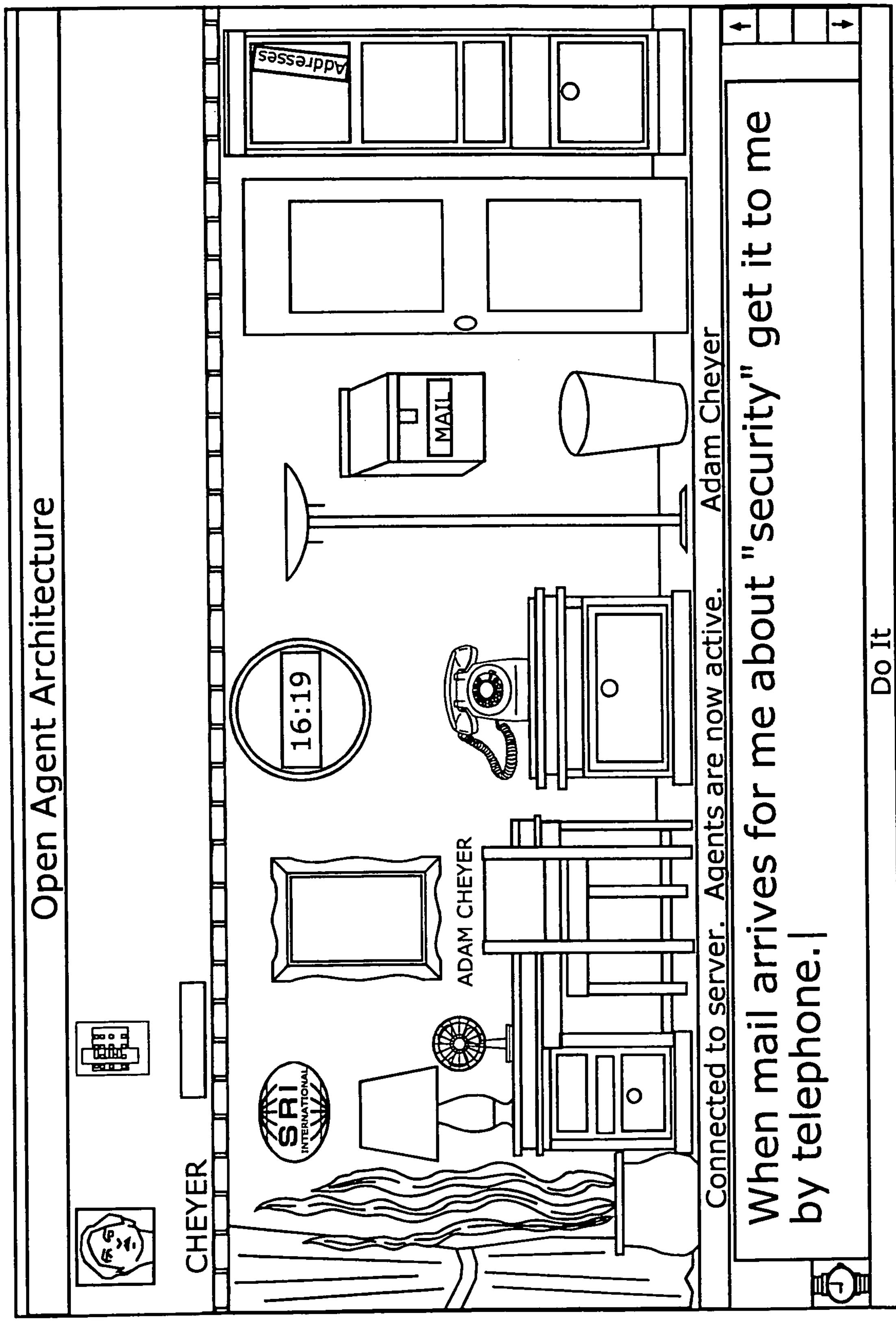

Open Agent Architecture
CHEYER
Addresses
MAIL
16:19
SRI INTERNATIONAL
ADAM CHEYER
Connected to server. Agents are now active.
Adam Cheyer
When mail arrives for me about "security" get it to me by telephone.
Do It

Fig. 5

APPROVED BY DRAFTSMAN	O.G. FIG.	
	CLASS	SUBCLASS

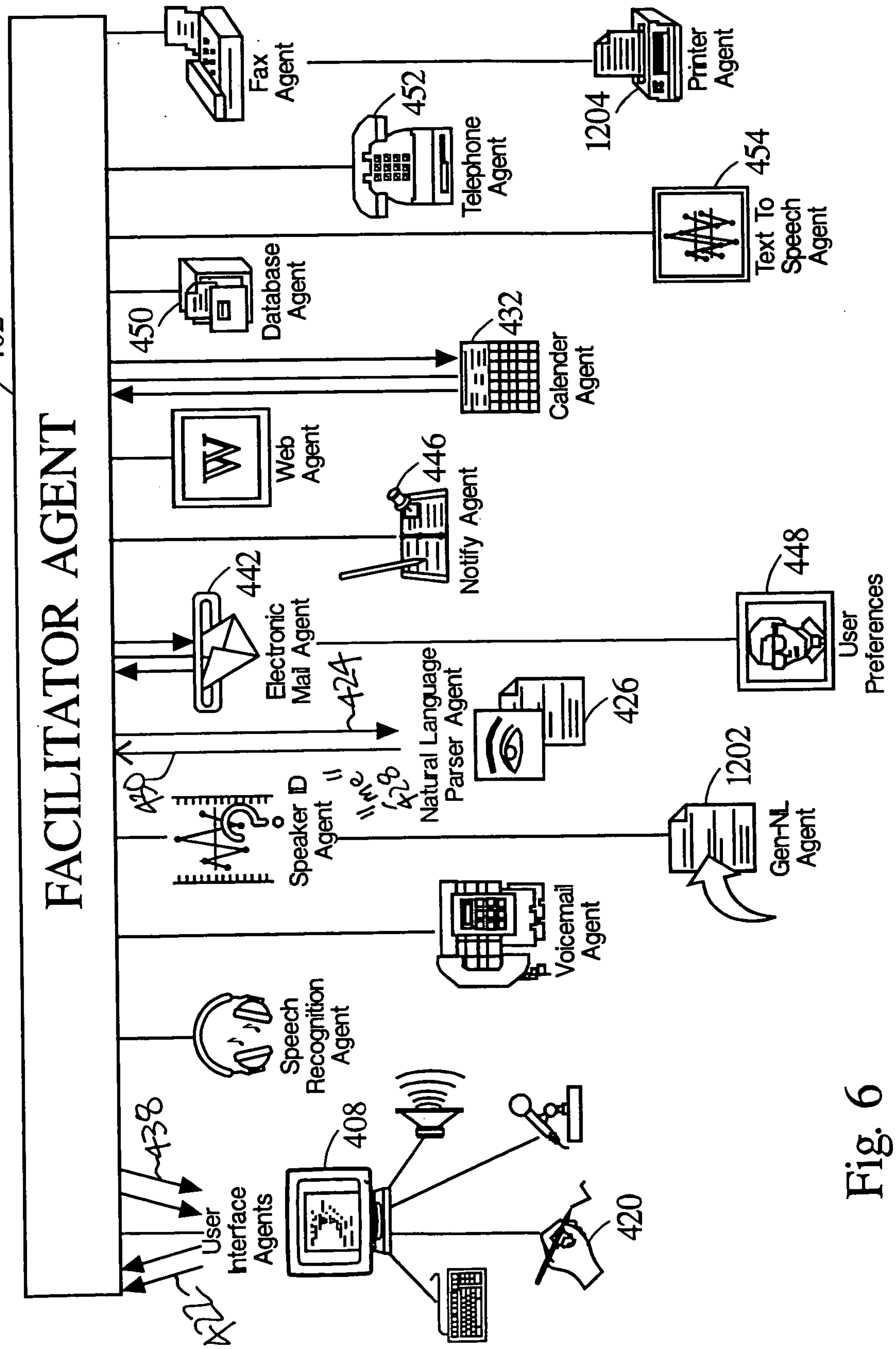


Fig. 6

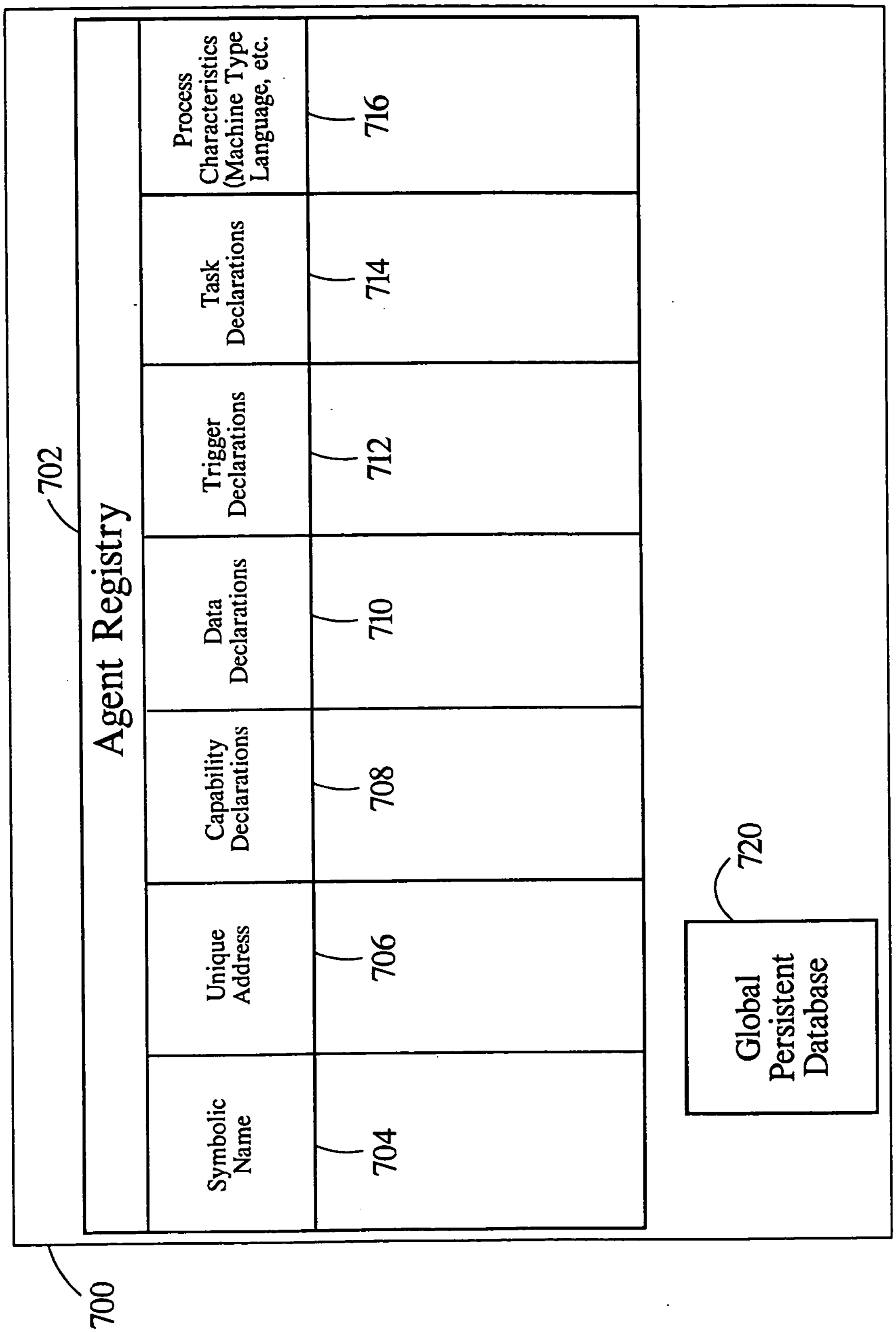

Agent Registry
702
Symbolic Name
704
Unique Address
706
Capability Declarations
708
Data Declarations
710
Trigger Declarations
712
Task Declarations
714
Process Characteristics (Machine Type Language, etc.
716
Global Persistent Database
720
700

Fig. 7

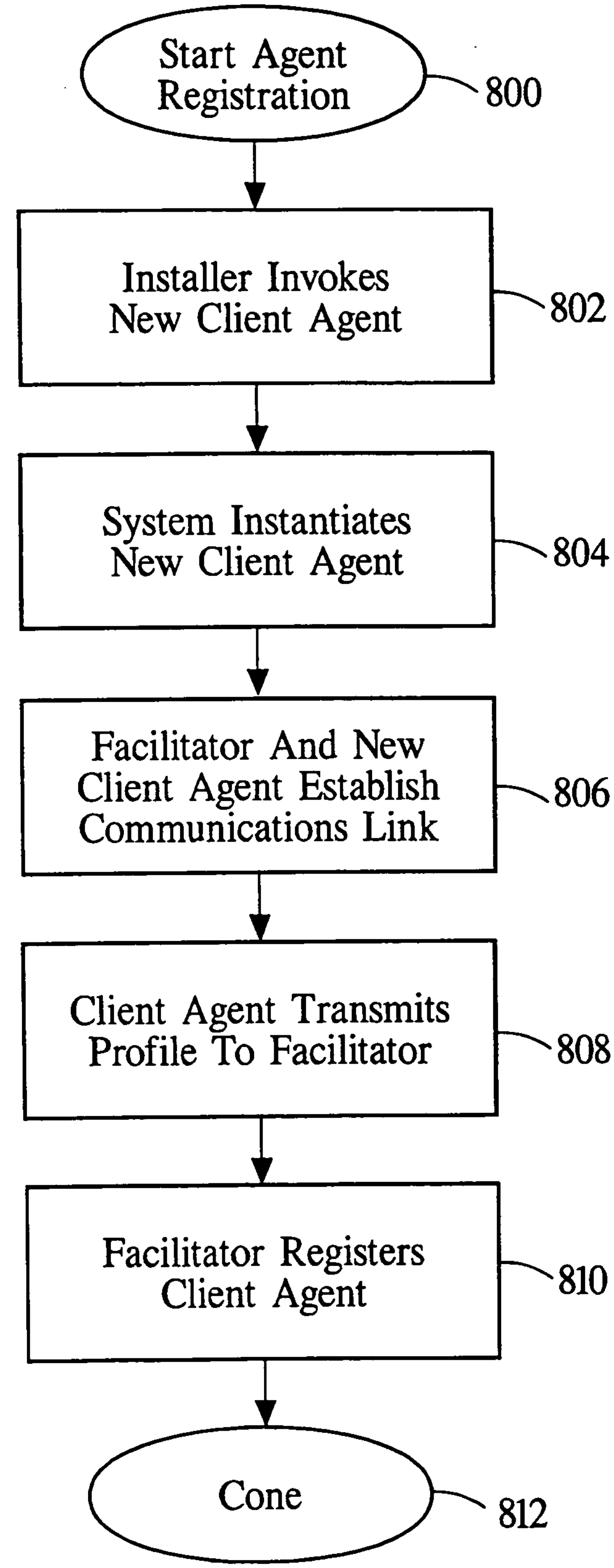

Start Agent Registration
800
Installer Invokes New Client Agent
802
System Instantiates New Client Agent
804
Facilitator And New Client Agent Establish Communications Link
806
Client Agent Transmits Profile To Facilitator
808
Facilitator Registers Client Agent
810
Cone
812

Fig. 8

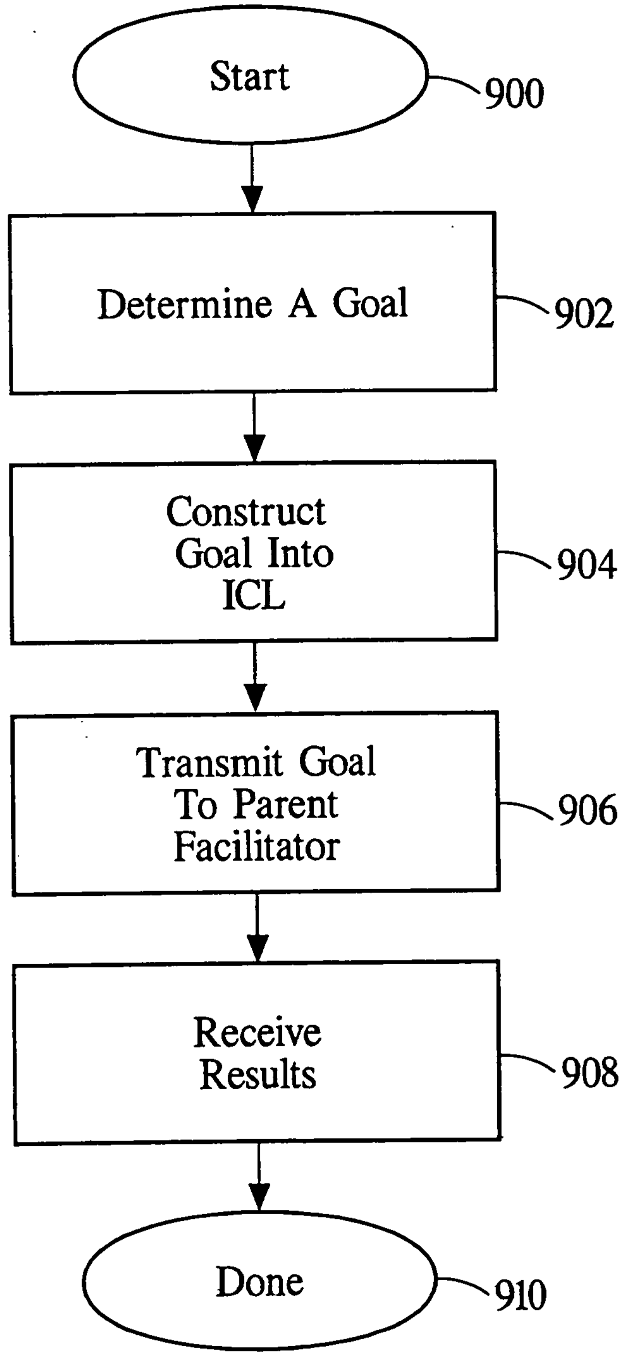

Start
900
Determine A Goal
902
Construct Goal Into ICL
904
Transmit Goal To Parent Facilitator
906
Receive Results
908
Done
910

Fig. 9

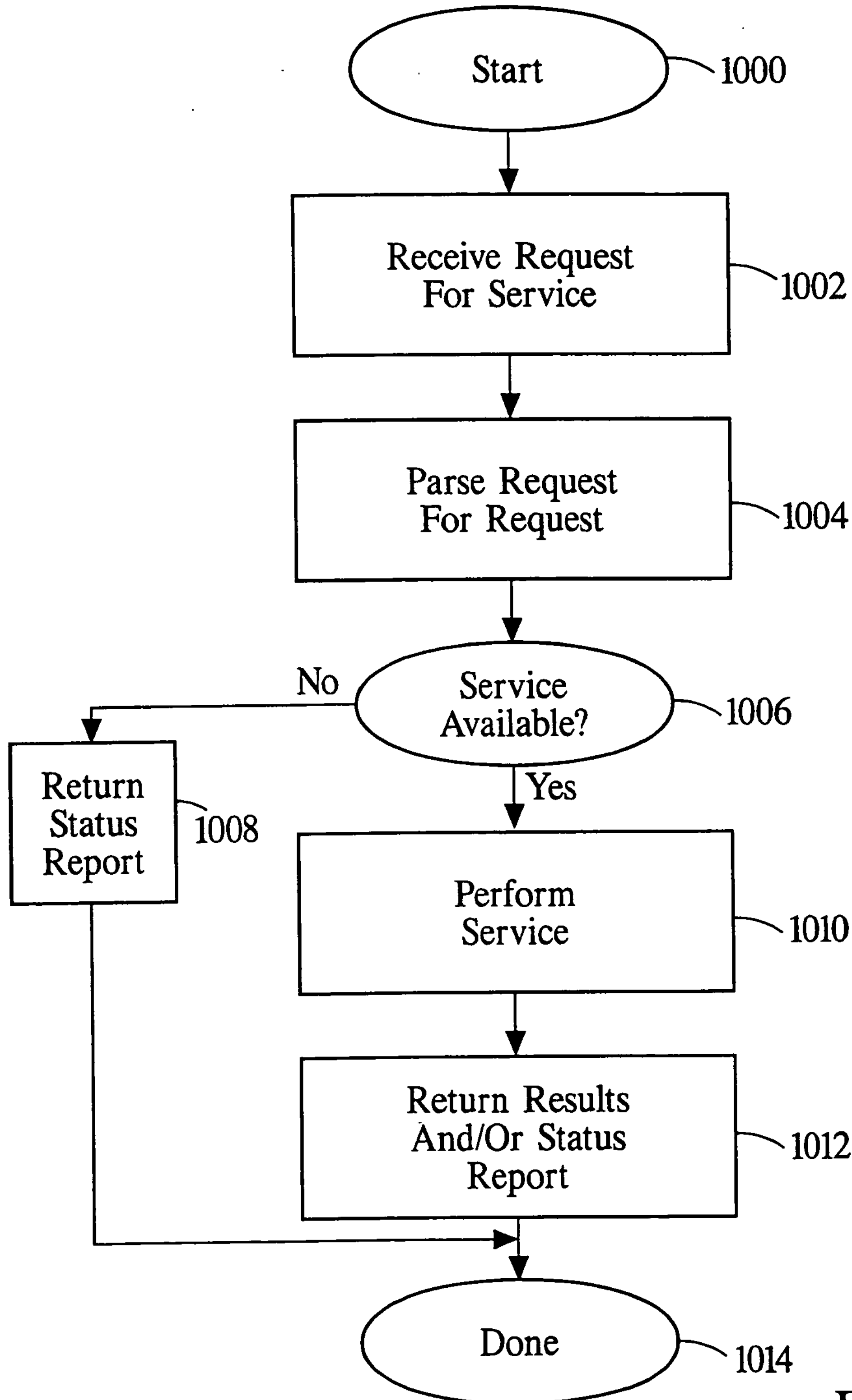

Start
1000
Receive Request For Service
1002
Parse Request For Request
1004
No
Service Available?
1006
Yes
Return Status Report
1008
Perform Service
1010
Return Results And/Or Status Report
1012
Done
1014

Fig. 10

APPROVED BY DRAFTSMAN | O.G. FIG. | CLASS | SUBCLASS

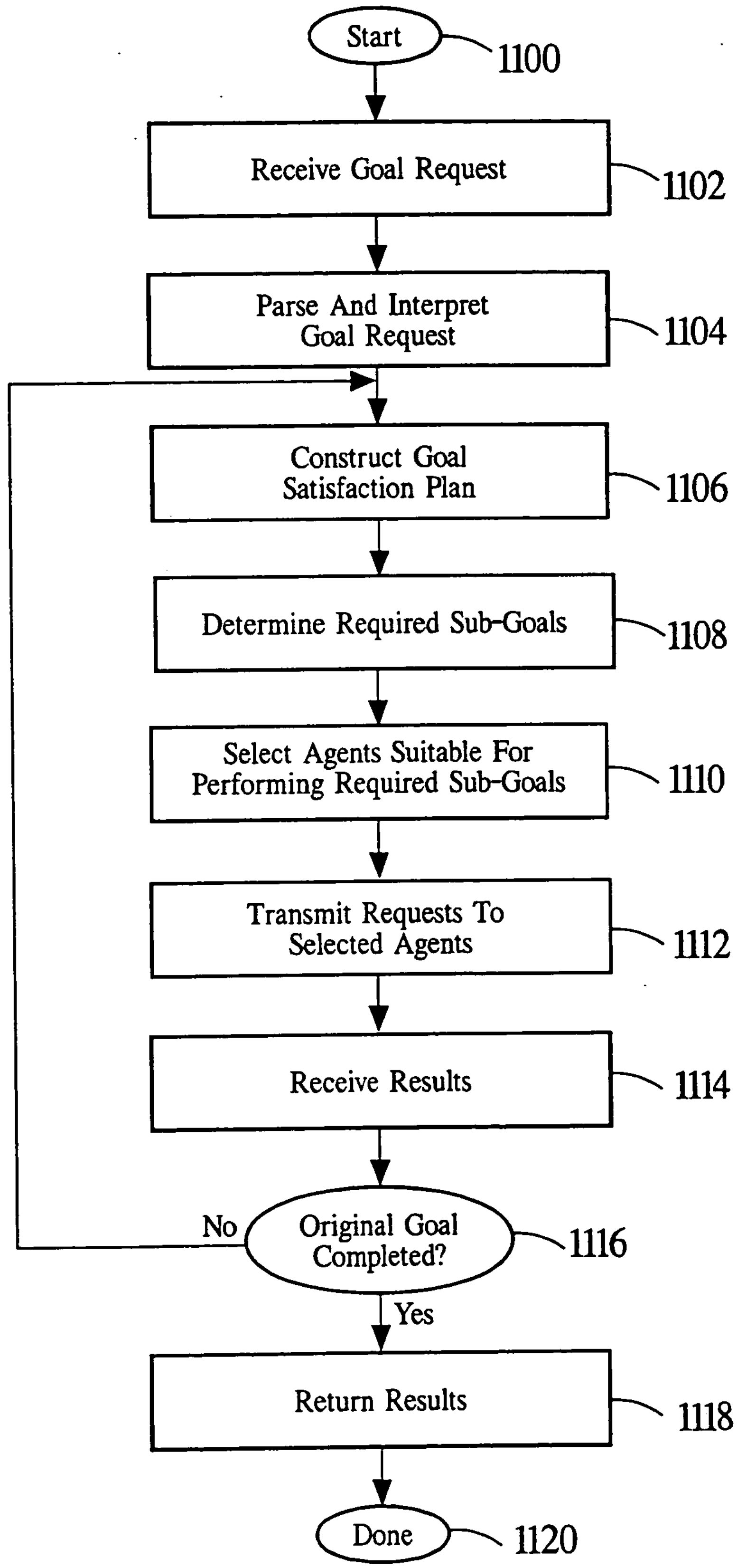


Fig. 11

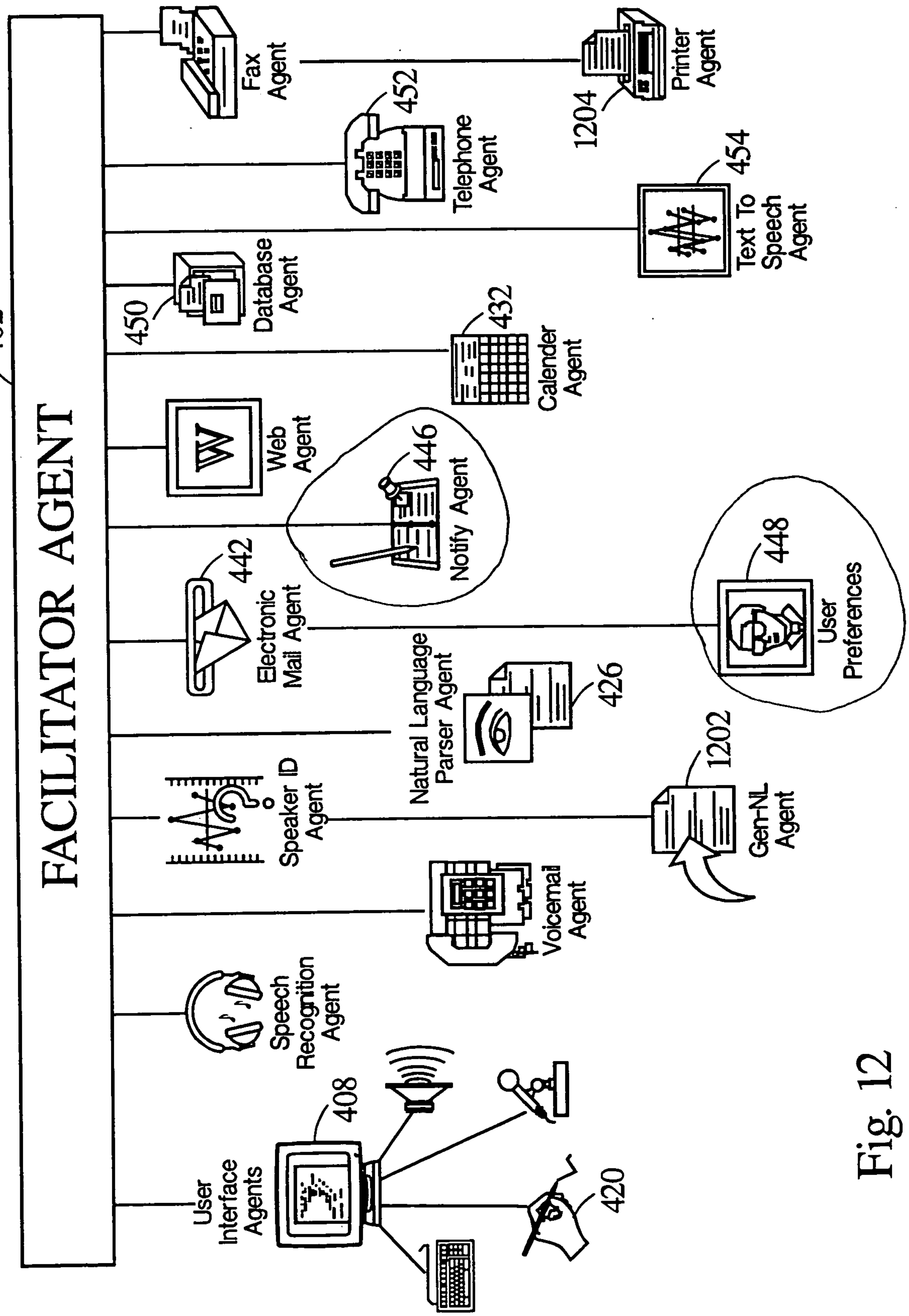

FACILITATOR AGENT
402
Fax Agent
Printer Agent
1204
Telephone Agent
452
Text To Speech Agent
454
Database Agent
450
Calender Agent
432
Web Agent
Notify Agent
446
Electronic Mail Agent
442
User Preferences
448
Natural Language Parser Agent
426
Speaker ID Agent
Gen-NL Agent
1202
Voicemail Agent
Speech Recognition Agent
User Interface Agents
408
420

Fig. 12

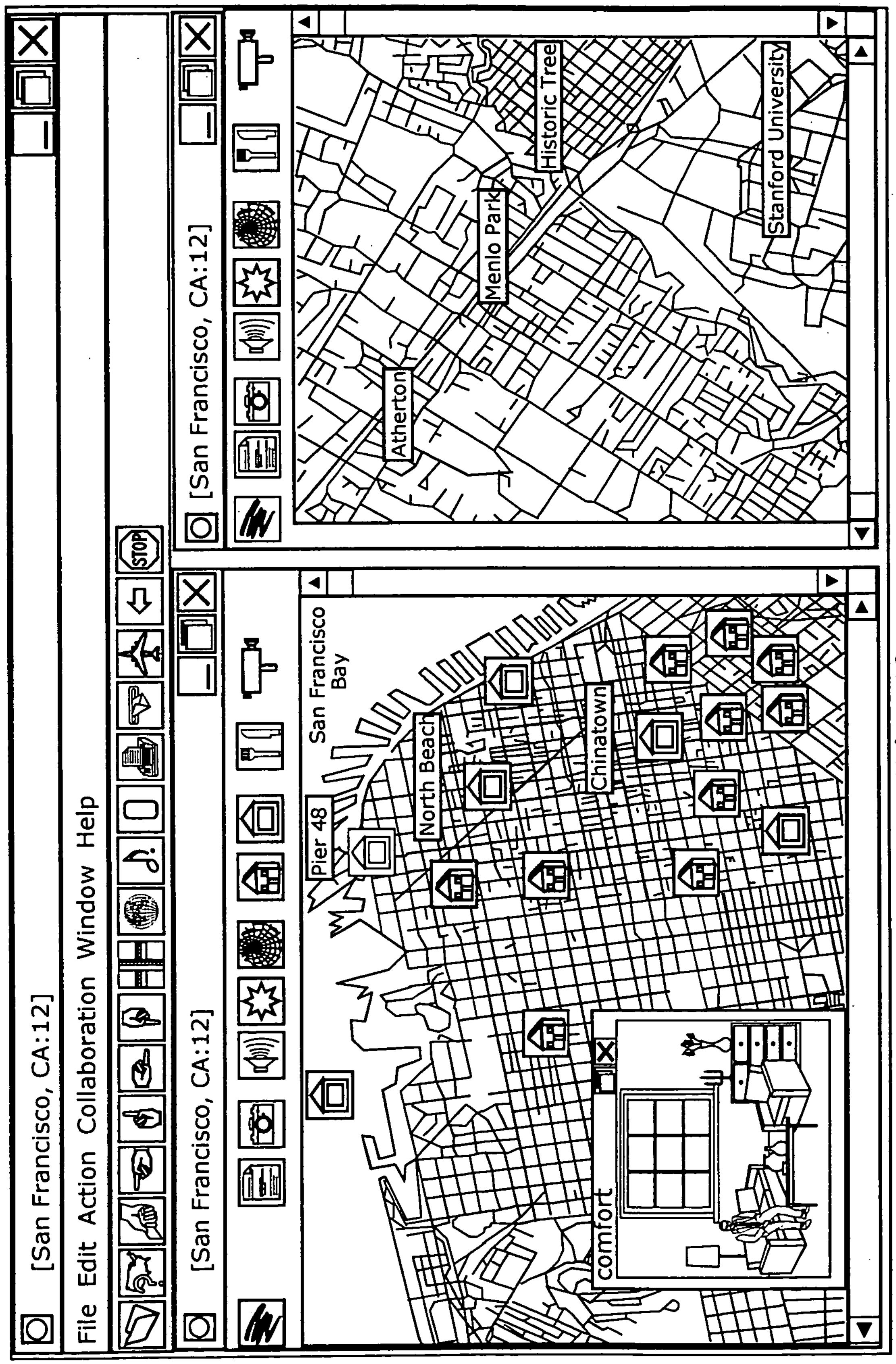

[San Francisco, CA:12]
File Edit Action Collaboration Window Help
STOP
[San Francisco, CA:12]
[San Francisco, CA:12]
San Francisco Bay
Pier 48
North Beach
Chinatown
comfort
Atherton
Menlo Park
Historic Tree
Stanford University

Fig. 13

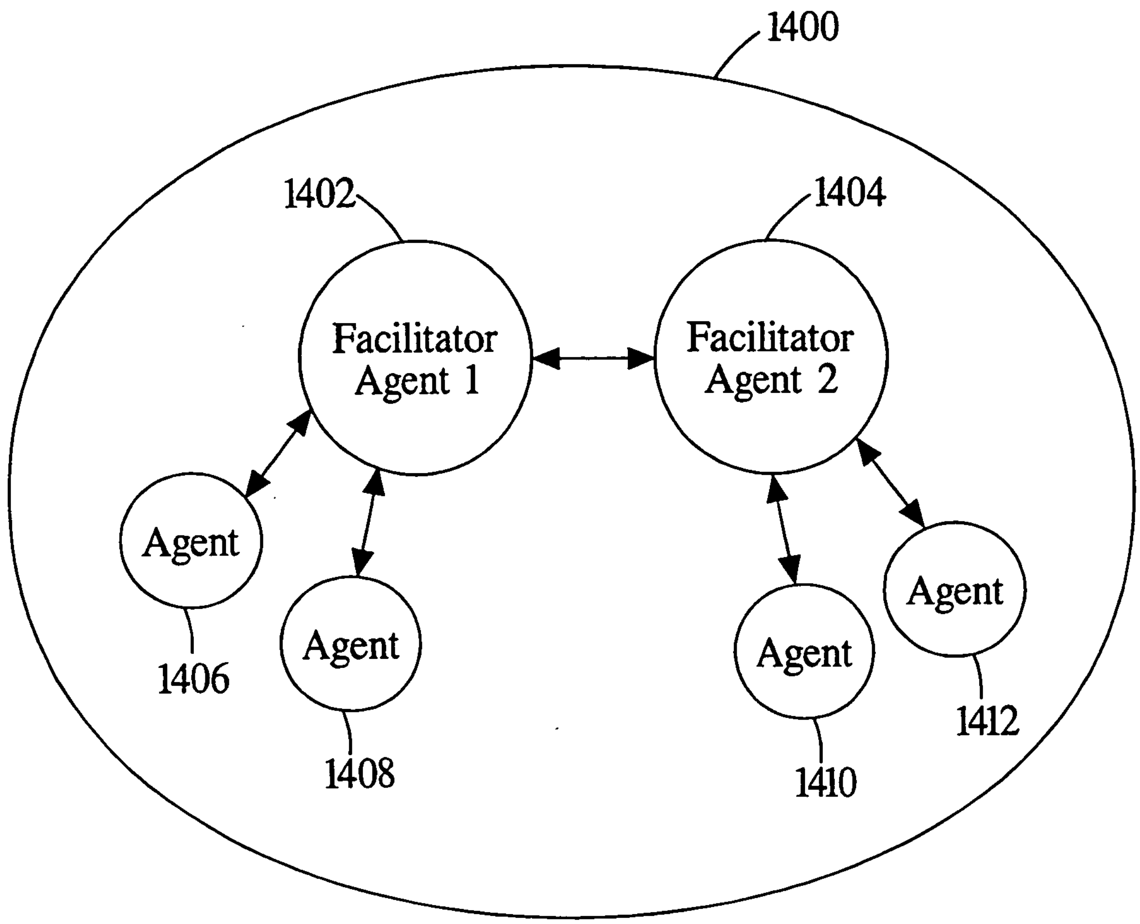

1400
1402
1404
Facilitator
Agent 1
Facilitator
Agent 2
Agent
Agent
Agent
Agent
1406
1408
1410
1412

Fig. 14

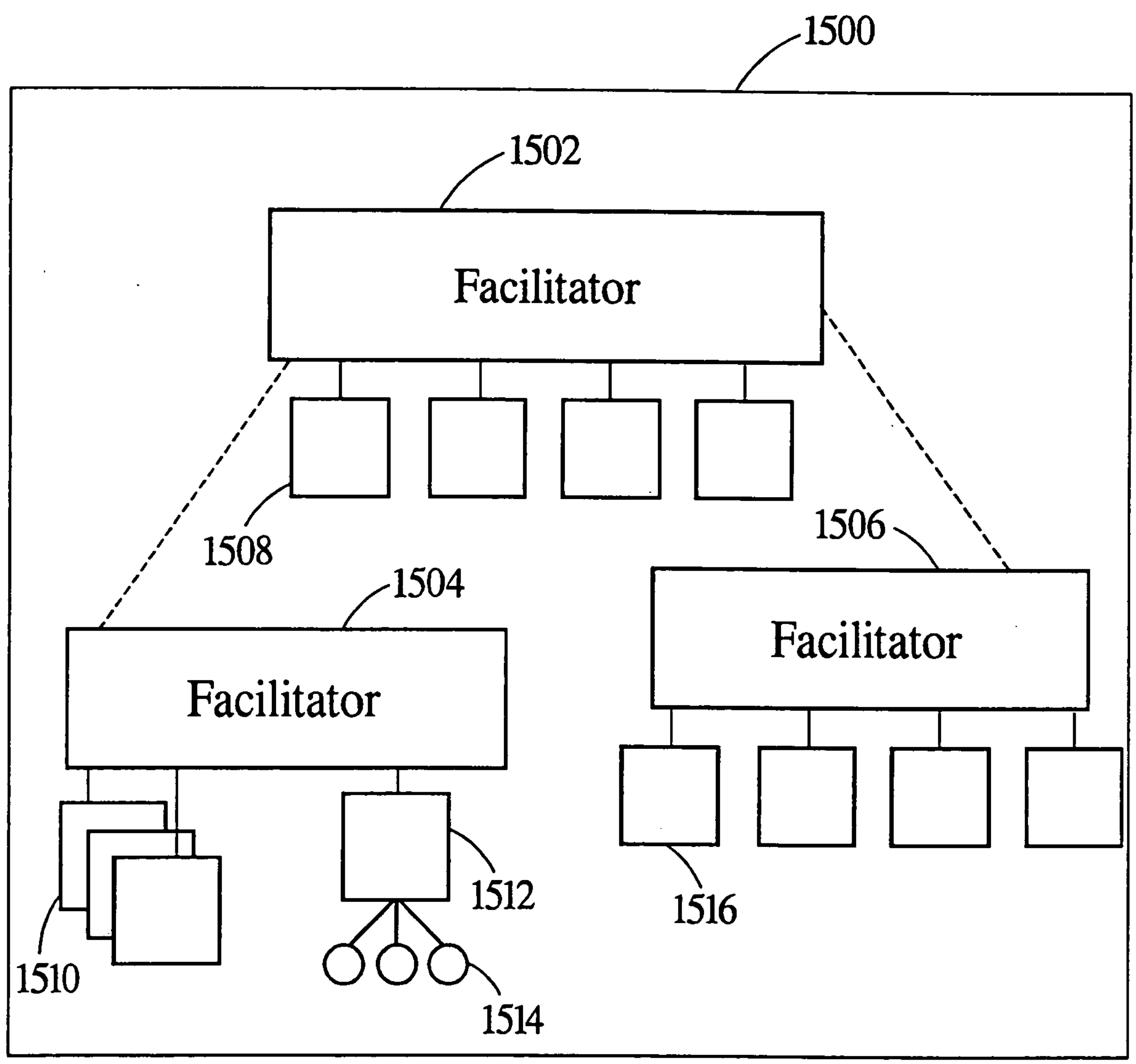

1500
1502
Facilitator
1508
1506
1504
Facilitator
Facilitator
1510
1512
1514
1516

Fig. 15

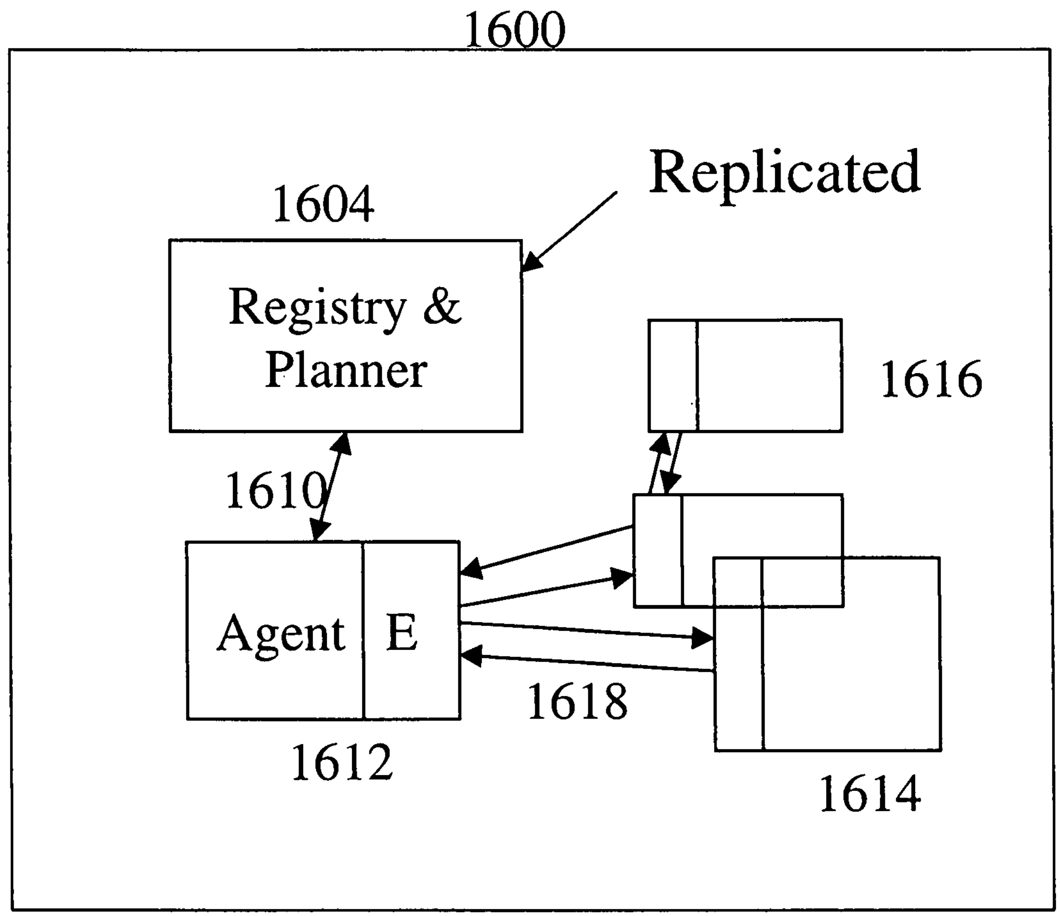


Figure 16

09225198 .010599

SRI1P016 16/61

Software-Based Architecture for Communication and Cooperation Among Distributed Electronic Agents

By:

Adam J. Cheyer and David L. Martin

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention is related to distributed computing environments and the completion of tasks within such environments. In particular, the present invention teaches a variety of software-based architectures for communication and cooperation among distributed electronic agents. Certain embodiments teach interagent communication languages enabling client agents to make requests in the form of arbitrarily complex goal expressions that are solved through facilitation by a facilitator agent.

Context and Motivation for Distributed Software Systems

The evolution of models for the design and construction of distributed software systems is being driven forward by several closely interrelated trends: the adoption of a *networked computing model*, rapidly rising expectations for *smarter, longer-lived, more autonomous software applications* and an ever increasing demand for *more accessible and intuitive user interfaces*.

Prior Art Figure 1 illustrates a *networked computing model* 100 having a plurality of client and server computer systems 120 and 122 coupled together over a physical transport mechanism 140. The adoption of the *networked computing model* 100 has lead to a greatly increased reliance on distributed sites for both data and processing resources. Systems such as the networked computing model 100 are based upon at least one physical transport mechanism 140 coupling the multiple computer systems 120 and 122 to support the transfer of information between these computers. Some of these computers basically support using the network and are known as *client* computers (*clients*). Some of these computers provide resources to other computers and are known as *server computers* (*servers*). The servers 122 can vary greatly in the resources they possess, access they provide and services made available to other computers across a network. Servers may service other servers as well as clients.

The Internet is a computing system based upon this network computing model. The Internet is continually growing, stimulating a paradigm shift for computing away from requiring all relevant data and programs to reside on the user's desktop machine. The data now routinely accessed from computers spread around the world has become increasingly rich in format, comprising multimedia documents, and audio and video streams. With the popularization of programming languages such as JAVA, data transported between local and remote machines may also include programs that can be downloaded and executed on the local machine. There is an ever increasing reliance on networked computing, necessitating software design approaches that allow for flexible composition of distributed processing elements in a dynamically changing and relatively unstable environment.

In an increasing variety of domains, application designers and users are coming to expect the deployment of *smarter, longer-lived, more autonomous, software applications*. Push technology, persistent monitoring of information sources, and the maintenance of user models, allowing for personalized responses and sharing of preferences, are examples of the simplest manifestations of this trend. Commercial enterprises are introducing significantly more advanced approaches, in many cases employing recent research results from artificial intelligence, data mining, machine learning, and other fields.

More than ever before, the increasing complexity of systems, the development of new technologies, and the availability of multimedia material and environments are creating a demand for *more accessible and intuitive user interfaces*. Autonomous, distributed, multi-component systems providing sophisticated services will no longer lend themselves to the familiar "direct manipulation" model of interaction, in which an individual user masters a fixed selection of commands provided by a single application. Ubiquitous computing, in networked environments, has brought about a situation in which the typical user of many software services is likely to be a non-expert, who may access a given service infrequently or only a few times.

Accommodating such usage patterns calls for new approaches. Fortunately, input modalities now becoming widely available, such as speech recognition and pen-based handwriting/gesture recognition, and the ability to manage the presentation of systems' responses by using multiple media provide an opportunity to fashion a style of human-computer interaction that draws much more heavily on our experience with human-human interactions.

PRIOR RELATED ART

Existing approaches and technologies for distributed computing include distributed objects, mobile objects, blackboard-style architectures, and agent-based software engineering.

The Distributed Object Approach

Object-oriented languages, such as C++ or JAVA, provide significant advances over standard procedural languages with respect to the reusability and modularity of code: *encapsulation*, *inheritance* and *polymorhpism*. Encapsulation encourages the creation of library interfaces that minimize dependencies on underlying algorithms or data structures. Changes to programming internals can be made at a later date with requiring modifications to the code that uses the library. Inheritance permits the extension and modification of a library of routines and data without requiring source code to the original library. Polymorphism allows one body of code to work on an arbitrary number of data types. For the sake of simplicity traditional objects may be seen to contain both methods and data. Methods provide the mechanisms by which the internal state of an object may be modified or by which communication may occur with another object or by which the instantiation or removal of objects may be directed.

With reference to Figure 2, a distributed object technology based around an Object Request Broker will now be described. Whereas "standard" object-oriented programming (OOP) languages can be used to build monolithic programs out of many object building blocks, distributed object technologies (DOOP) allow the creation of programs whose components may be spread across multiple machines. As shown in Figure 2, an object system 200 includes client objects 210 and server objects 220. To implement a client-server relationship between objects, the distributed object system

200 uses a registry mechanism (CORBA's registry is called an Object Request Broker, or ORB) 230 to store the interface descriptions of available objects. Through the services of the ORB 230, a client can transparently invoke a method on a remote server object. The ORB 230 is then responsible for finding the object 220 that can implement the request, passing it the parameters, invoking its method, and returning the results. In the most sophisticated systems, the client 210 does not have to be aware of where the object is located, its programming language, its operating system, or any other system aspects that are not part of the server object's interface.

Although distributed objects offer a powerful paradigm for creating networked applications, certain aspects of the approach are not perfectly tailored to the constantly changing environment of the Internet. A major restriction of the DOOP approach is that the interactions among objects are fixed through explicitly coded instructions by the application developer. It is often difficult to reuse an object in a new application without bringing along all its inherent dependencies on other objects (embedded interface definitions and explicit method calls). Another restriction of the DOOP approach is the result of its reliance on a remote procedure call (RPC) style of communication. Although easy to debug, this single thread of execution model does not facilitate programming to exploit the potential for parallel computation that one would expect in a distributed environment. In addition, RPC uses a blocking (synchronous) scheme that does not scale well for high-volume transactions.

Mobile Objects

Mobile objects, sometimes called mobile agents, are bits of code that can move to another execution site (presumably on a different machine) under their own programmatic control, where they can then interact with the local environment. For certain types of problems, the mobile object paradigm offers advantages over more traditional distributed object approaches. These advantages include network bandwidth and parallelism. Network bandwidth advantages exist for some database queries or electronic commerce applications, where it is more efficient to perform tests on data by bringing the tests to the data than by bringing large amounts of data to the testing program. Parallelism advantages include situations in which mobile agents can be spawned in parallel to accomplish many tasks at once.

Some of the disadvantages and inconveniences of the mobile agent approach include the programmatic specificity of the agent interactions, lack of coordination support between participant agents and execution environment irregularities regarding specific programming languages supported by host processors upon which agents reside. In a fashion similar to that of DOOP programming, an agent developer must programmatically specify where to go and how to interact with the target environment. There is generally little coordination support to encourage interactions among multiple (mobile) participants. Agents must be written in the programming language supported by the execution environment, whereas many other distributed technologies support heterogeneous communities of components, written in diverse programming languages.

Blackboard Architectures

Blackboard architectures typically allow multiple processes to communicate by reading and writing tuples from a global data store. Each process can watch for items of interest, perform computations based on the state of the blackboard, and then add partial results or queries that other processes can consider. Blackboard architectures provide a flexible framework for problem solving by a dynamic community of distributed processes. A blackboard architecture provides one solution to eliminating the tightly bound interaction links that some of the other distributed technologies require during interprocess communication. This advantage can also be a disadvantage: although a programmer does not need to refer to a specific process during computation, the framework does not provide programmatic control for doing so in cases where this would be practical.

Agent-based Software Engineering

Several research communities have approached distributed computing by casting it as a problem of modeling communication and cooperation among autonomous entities, or agents. Effective communication among independent agents requires four components: (1) a transport mechanism carrying messages in an asynchronous fashion, (2) an interaction protocol defining various types of communication interchange and their social implications (for instance, a response is expected of a question), (3) a content language permitting the expression and interpretation of utterances, and (4) an agreed-upon set of shared vocabulary and meaning for concepts (often called an *ontology*). Such mechanisms permit a much richer style of interaction among participants than can be expressed using a distributed object's RPC model or a blackboard architecture's centralized exchange approach.

Agent-based systems have shown much promise for flexible, fault-tolerant, distributed problem solving. Several agent-based projects have helped to evolve the notion of facilitation. However, existing agent-based technologies and architectures are typically very limited in the extent to which agents can specify complex goals or influence the strategies used by the facilitator. Further, such prior systems are not sufficiently attuned to the importance of integrating human agents (i.e., users) through natural language and other human-oriented user interface technologies.

The initial version of SRI International's Open Agent Architecture™ ("*OAA*®") technology provided only a very limited mechanism for dealing with compound goals. Fixed formats were available for specifying a flat list of either conjoined (AND) sub-goals or disjoined (OR) sub-goals; in both cases, parallel goal solving was hard-wired in, and only a single set of parameters for the entire list could be specified. More complex goal expressions involving (for example) combinations of different boolean connectors, nested expressions, or conditionally interdependent ("IF .. THEN") goals were not supported. Further, system scalability was not adequately addressed in this prior work.

SUMMARY OF INVENTION

A first embodiment of the present invention discloses a highly flexible, software-based architecture 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 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 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 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. Specific embodiments providing enhanced scalability are also described.

BRIEF DESCRIPTION OF THE DRAWINGS

Prior Art

Prior Art FIGURE 1 depicts a networked computing model;

Prior Art FIGURE 2 depicts a distributed object technology based around an Object Resource Broker;

Examples of the Invention

FIGURE 3 depicts a distributed agent system based around a facilitator agent;

FIGURE 4 presents a structure typical of one small system of the present invention;

FIGURE 5 depicts an Automated Office system implemented in accordance with an example embodiment of the present invention supporting a mobile user with a laptop computer and a telephone;

FIGURE 6 schematically depicts an Automated Office system implemented as a network of agents in accordance with a preferred embodiment of the present invention;

FIGURE 7 schematically shows data structures internal to a facilitator in accordance with a preferred embodiment of the present invention;

FIGURE 8 depicts operations involved in instantiating a client agent with its parent facilitator in accordance with a preferred embodiment of the present invention;

FIGURE 9 depicts operations involved in a client agent initiating a service request and receiving the response to that service request in accordance with a certain preferred embodiment of the present invention;

FIGURE 10 depicts operations involved in a client agent responding to a service request in accordance with another preferable embodiment of the present invention;

FIGURE 11 depicts operations involved in a facilitator agent response to a service request in accordance with a preferred embodiment of the present invention;

FIGURE 12 depicts an Open Agent Architecture™ based system of agents implementing a unified messaging application in accordance with a preferred embodiment of the present invention;

FIGURE 13 depicts a map oriented graphical user interface display as might be displayed by a multi-modal map application in accordance with a preferred embodiment of the present invention;

FIGURE 14 depicts a peer to peer multiple facilitator based agent system supporting distributed agents in accordance with a preferred embodiment of the present invention;

FIGURE 15 depicts a multiple facilitator agent system supporting at least a limited form of a hierarchy of facilitators in accordance with a preferred embodiment of the present invention; and

FIGURE 16 depicts a replicated facilitator architecture in accordance with one embodiment of the present invention.

BRIEF DESCRIPTION OF THE APPENDICES

The Appendices provide source code for an embodiment of the present invention written in the PROLOG programming language.

APPENDIX A: Source code file named compound.pl.

APPENDIX B: Source code file named fac.pl.

APPENDIX C: Source code file named libcom_tcp.pl.

APPENDIX D: Source code file named liboaa.pl.

APPENDIX E: Source code file named translations.pl.

DETAILED DESCRIPTION OF THE INVENTION

Figure 3 illustrates a distributed agent system 300 in accordance with one embodiment of the present invention. The agent system 300 includes a facilitator agent 310 and a plurality of agents 320. The illustration of Figure 3 provides a high level view of one simple system structure contemplated by the present invention. The facilitator agent 310 is in essence the "parent" facilitator for its "children" agents 320. The agents 320 forward service requests to the facilitator agent 310. The facilitator agent 310 interprets these requests, organizing a set of goals which are then delegated to appropriate agents for task completion.

The system 300 of Figure 3 can be expanded upon and modified in a variety of ways consistent with the present invention. For example, the agent system 300 can be distributed across a computer network such as that illustrated in Figure 1. The facilitator agent 310 may itself have its functionality distributed across several different computing platforms. The agents 320 may engage in interagent communication (also called peer to peer communications). Several different systems 300 may be coupled together for enhanced performance. These and a variety of other structural configurations are described below in greater detail.

Figure 4 presents the structure typical of a small system 400 in one embodiment of the present invention, showing user interface agents 408, several application agents 404 and meta-agents 406, the system 400 organized as a community of peers by their common relationship to a facilitator agent 402. As will be appreciated, Figure 4 places more structure upon the system 400 than shown in Figure 3, but both are valid representations of structures of the present invention. The facilitator 402 is a specialized server agent that is responsible for coordinating agent communications and cooperative problem-solving. The facilitator 402 may also provide a global data store for its client agents, allowing them to adopt a blackboard style of interaction. Note that certain advantages are found in utilizing two or more facilitator agents within the system 400. For example, larger systems can be assembled from multiple facilitator/client groups, each having the sort of structure shown in Figure 4. All agents that are not facilitators are referred to herein generically as *client* agents -- so called because each acts (in some respects) as a client of some facilitator, which provides communication and other essential services for the client.

The variety of possible client agents is essentially unlimited. Some typical categories of client agents would include application agents 404, meta-agents 406, and user interface agents 408, as depicted in Figure 4. Application agents 404 denote specialists that provide a collection of services of a particular sort. These services could be domain-independent technologies (such as speech recognition, natural language processing 410, email, and some forms of data retrieval and data mining) or user-specific or domain-specific (such as a travel planning and reservations agent). Application agents may be based on legacy applications or libraries, in which case the agent may be little more than a wrapper that calls a pre-existing API 412, for example. Meta-agents 406 are agents whose role is to assist the facilitator agent 402 in coordinating the activities of other agents. While the facilitator 402 possesses domain-independent coordination strategies, meta-agents 406 can augment these by using domain- and application-specific knowledge or reasoning (including but not limited to rules, learning algorithms and planning).

With further reference to Figure 4, user interface agents 408 can play an extremely important and interesting role in certain embodiments of the present invention. By way of explanation, in some systems, a user interface agent can be implemented as a collection of "micro-agents", each monitoring a different input modality (point-and-click, handwriting, pen gestures, speech), and collaborating to produce the best interpretation of the current inputs. These micro-agents are depicted in Figure 4, for example, as Modality Agents 414. While describing such subcategories of client agents is useful for purposes of illustration and understanding, they need not be formally distinguished within the system in preferred implementations of the present invention.

The operation of one preferred embodiment of the present invention will be discussed in greater detail below, but may be briefly outlined as follows. When invoked, a client agent makes a connection to a facilitator, which is known as its *parent facilitator*. These connections are depicted as a double headed arrow between the client agent and the facilitator agent in Figure 3 and 4, for example. Upon connection, an agent registers with its parent facilitator a specification of the capabilities and services it can provide. For example, a natural language agent may register the characteristics of its available natural language vocabulary. (For more details regarding client agent connections, see the discussion of Figure 8 below.) Later during task completion, when a facilitator determines that the registered services 416 of one of its client agents will help satisfy a goal, the facilitator sends that client a request expressed in the Interagent Communication Language (*ICL*) 418. (See Figure 11 below for a more detailed discussion of the facilitator operations involved.) The agent parses this request, processes it, and returns answers or status reports to the facilitator. In processing a request, the client agent can make use of a variety of infrastructure capabilities provided in the preferred embodiment. For example, the client agent can use *ICL* 418 to request services of other agents, set triggers, and read or write shared data on the facilitator or other client agents that maintain shared data. (See the discussion of Figures 9-11 below for a more detailed discussion of request processing.)

The functionality of each client agent are made available to the agent community through registration of the client agent's capabilities with a facilitator 402. A software "wrapper" essentially surrounds the underlying application program performing the services offered by each client. The common infrastructure for constructing agents is preferably supplied by an *agent library*. The agent library is preferably accessible in the runtime environment of several different programming languages. The agent library preferably minimizes the effort required to construct a new system and maximizes the ease with which legacy systems can be “wrapped” and made compatible with the agent-based architecture of the present invention.

By way of further illustration, a representative application is now briefly presented with reference to Figures 5 and 6. In the Automated Office system depicted in Figure 5, a mobile user with a telephone and a laptop computer can access and task commercial applications such as calendars, databases, and email systems running back at the office. A user interface (UI) agent 408, shown in Figure 6, runs on the user's local laptop and is responsible for accepting user input, sending requests to the facilitator 402 for delegation to appropriate agents, and displaying the results of the distributed computation. The user may interact directly with a specific remote application by clicking on active areas in the interface, calling up a form or window for that application, and making queries with standard interface dialog mechanisms. Conversely, a user may express a task to be executed by using typed, handwritten, or spoken (over the telephone) English sentences, without explicitly specifying which agent or agents should perform the task.

For instance, if the question "What is my schedule?" is written 420 in the user interface 408, this request will be sent 422 by the UI 408 to the facilitator 402, which in turn will ask 424 a natural language (NL) agent 426 to translate the query into *ICL* 18. To accomplish this task, the NL agent 426 may itself need to make requests of the agent community to resolve unknown words such as "me" 428 (the UI agent 408 can respond 430 with the name of the current user) or "schedule" 432 (the calendar agent 434 defines this word 436). The resulting *ICL* expression is then routed by the facilitator 402 to appropriate agents (in this case, the calendar agent 434) to execute the request. Results are sent back 438 to the UI agent 408 for display.

The spoken request "When mail arrives for me about security, notify me immediately." produces a slightly more complex example involving communication among all agents in the system. After translation into *ICL* as described above, the facilitator installs a trigger 440 on the mail agent 442 to look for new messages about security. When one such message does arrive in its mail spool, the trigger fires, and the facilitator matches the action part of the trigger to capabilities published by the notification agent 446. The notification agent 446 is a meta-agent, as it makes use of rules concerning the optimal use of different output modalities (email, fax, speech generation over the telephone) plus information about an individual user's preferences 448 to determine the best way of relaying a message through available media transfer application agents. After some competitive parallelism to locate the user (the calendar agent 434 and database agent 450 may have different guesses as to where to find the user) and some cooperative parallelism to produce required information (telephone number of location, user password, and an audio file containing a text-to-speech representation of the email message), a telephone agent 452 calls the user, verifying its identity through touchtones, and then play the message.

The above example illustrates a number of inventive features. As new agents connect to the facilitator, registering capability specifications and natural language vocabulary, what the user can say and do dynamically changes; in other words, the ICL is dynamically ***expandable***. For example, adding a calendar agent to the system in the previous example and registering its capabilities enables users to ask natural language questions about their "schedule" without any need to revise code for the facilitator, the natural language agents, or any other client agents. In addition, the interpretation and execution of a task is a distributed process, with no single agent defining the set of possible inputs to the system. Further, a single request can produce cooperation and flexible communication among many agents, written in different programming languages and spread across multiple machines.

Design Philosophy and Considerations

One preferred embodiment provides an integration mechanism for heterogeneous applications in a distributed infrastructure, incorporating some of the dynamism and extensibility of blackboard approaches, the efficiency associated with mobile objects, plus the rich and complex interactions of communicating agents. Design goals for preferred embodiments of the present invention may be categorized under the general headings of *interoperation and cooperation, user interfaces*, and *software engineering*. These design goals are not absolute requirements, nor will they necessarily be satisfied by all embodiments of the present invention, but rather simply reflect the inventor's currently preferred design philosophy.

Versatile mechanisms of interoperation and cooperation

Interoperation refers to the ability of distributed software components - agents - to communicate meaningfully. While every system-building framework must provide mechanisms of interoperation at some level of granularity, agent-based frameworks face important new challenges in this area. This is true primarily because autonomy, the hallmark of *individual* agents, necessitates greater flexibility in interactions within *communities* of agents. *Coordination* refers to the mechanisms by which a community of agents is able to work together productively on some task. In these areas, the goals for our framework are to *provide flexibility in assembling* *communities of autonomous service providers, provide flexibility in structuring cooperative interactions, impose the right amount of structure*, as well as *include legacy and "owned-elsewhere" applications.*

Provide flexibility in assembling communities of autonomous service providers -- both at development time and at runtime. Agents that conform to the linguistic and ontological requirements for effective communication should be able to participate in an agent community, in various combinations, with minimal or near minimal prerequisite knowledge of the characteristics of the other players. Agents with duplicate and overlapping capabilities should be able to coexist within the same community, with the system making optimal or near optimal use of the redundancy.

Provide flexibility in structuring cooperative interactions among the members of a community of agents. A framework preferably provides an economical mechanism for setting up a variety of interaction patterns among agents, without requiring an inordinate amount of complexity or infrastructure within the individual agents. The provision of a service should be independent or minimally dependent upon a particular configuration of agents.

Impose the right amount of structure on individual agents. Different approaches to the construction of multi-agent systems impose different requirements on the individual agents. For example, because KQML is neutral as to the content of messages, it imposes minimal structural requirements on individual agents. On the other hand, the BDI paradigm tends to impose much more demanding requirements, by making assumptions about the nature of the programming elements that are meaningful to individual agents. Preferred embodiments of the present invention should fall somewhere between the two, providing a rich set of interoperation and coordination capabilities, without precluding any of the software engineering goals defined below.

Include legacy and "owned-elsewhere" applications. Whereas *legacy* usually implies reuse of an established system fully controlled by the agent-based system developer, *owned-elsewhere* refers to applications to which the developer has partial access, but no control. Examples of owned-elsewhere applications include data sources and services available on the World Wide Web, via simple form-based interfaces, and applications used cooperatively within a virtual enterprise, which remain the properties of separate corporate entities. Both classes of application must preferably be able to interoperate, more or less as full-fledged members of the agent community, without requiring an overwhelming integration effort.

Human-oriented user interfaces

Systems composed of multiple distributed components, and possibly dynamic configurations of components, require the crafting of intuitive user interfaces to *provide conceptually natural interaction mechanisms, treat users as privileged members of the agent community* and *support collaboration.*

Provide conceptually natural interaction mechanisms with multiple distributed components. When there are numerous disparate agents, and/or complex tasks implemented by the system, the user should be able to express requests without having detailed knowledge of the individual agents. With speech recognition, handwriting recognition, and natural language technologies becoming more mature, agent architectures should preferably support these forms of input playing increased roles in the tasking of agent communities.

Preferably treat *users as privileged members* of the agent community by providing an appropriate level of task specification within *software* agents, and reusable translation mechanisms between this level and the level of *human* requests, supporting constructs that seamlessly incorporate interactions between both human-interface and software types of agents.

Preferably support *collaboration* (simultaneous work over shared data and processing resources) between users and agents.

Realistic software engineering requirements

System-building frameworks should preferably address the practical concerns of real-world applications by the specification of requirements which preferably include: *Minimize the effort* required to create new agents, and to wrap existing applications. *Encourage reuse,* both of domain-independent and domain-specific components. The concept of *agent orientation*, like that of object orientation, provides a natural conceptual framework for reuse, so long as mechanisms for encapsulation and interaction are structured appropriately. *Support lightweight, mobile platforms.* Such platforms should be able to serve as hosts for agents, without requiring the installation of a massive environment. It should also be possible to construct individual agents that are relatively small and modest in their processing requirements. *Minimize platform and language barriers.* Creation of new agents, as well as wrapping of existing applications, should not require the adoption of a new language or environment.

Mechanisms of Cooperation

Cooperation among agents in accordance with the present invention is preferably achieved via messages expressed in a common language, *ICL*. Cooperation among agent is further preferably structured around a three-part approach: providers of services register capabilities specifications with a facilitator, requesters of services construct goals and relay them to a facilitator, and facilitators coordinate the efforts of the appropriate service providers in satisfying these goals.

The Interagent Communication Language (ICL)

Interagent Communication Language (*"ICL"*) 418 refers to an interface, communication, and task coordination language preferably shared by all agents, regardless of what platform they run on or what computer language they are programmed in. *ICL* may be used by an agent to task itself or some subset of the agent community. Preferably, *ICL* allows agents to specify explicit control parameters while simultaneously supporting expression of goals in an underspecified, loosely constrained manner. In a further preferred embodiment, agents employ *ICL* to perform queries, execute actions, exchange information, set triggers, and manipulate data in the agent community.

In a further preferred embodiment, a program element expressed in *ICL* is the *event*. The activities of every agent, as well as communications between agents, are preferably structured around the transmission and handling of events. In communications, events preferably serve as messages between agents; in regulating the activities of individual agents, they may preferably be thought of as goals to be satisfied. Each event preferably has a type, a set of parameters, and content. For example, the agent library procedure *oaa_Solve* can be used by an agent to request services of other agents. A call to *oaa_Solve*, within the code of agent *A*, results in an event having the form

ev_post_solve(Goal, Params)

going from *A* to the facilitator, where *ev_post_solve* is the type, *Goal* is the content, and *Params* is a list of parameters. The allowable content and parameters preferably vary according to the type of the event.

The *ICL* preferably includes a layer of conversational protocol and a content layer. The conversational layer of *ICL* is defined by the event types, together with the parameter lists associated with certain of these event types. The content layer consists of the specific goals, triggers, and data elements that may be embedded within various events.

The *ICL* conversational protocol is preferably specified using an orthogonal, parameterized approach, where the conversational aspects of each element of an interagent conversation are represented by a selection of an event type and a selection of values from at least one orthogonal set of parameters. This approach offers greater expressiveness than an approach based solely on a fixed selection of *speech acts*, such as embodied in KQML. For example, in KQML, a request to satisfy a query can employ either of the performatives *ask_all* or *ask_one*. In *ICL*, on the other hand, this type of request preferably is expressed by the event type *ev_post_solve*, together with the *solution_limit(N)* parameter - where *N* can be any positive integer. (A request for all solutions is indicated by the omission of the *solution_limit* parameter.) The request can also be accompanied by other parameters, which combine to further refine its semantics. In KQML, then, this example forces one to choose between two possible conversational options, neither of which may be precisely what is desired. In either case, the performative chosen is a single value that must capture the entire conversational characterization of the communication. This requirement raises a difficult challenge for the language designer, to select a set of performatives that provides the desired functionality without becoming unmanageably large. Consequently, the debate over the right set of performatives has consumed much discussion within the KQML community.

The content layer of the *ICL* preferably supports unification and other features found in logic programming language environments such as PROLOG. In some embodiments, the content layer of the *ICL* is simply an extension of at least one programming language. For example, the Applicants have found that PROLOG is suitable for implementing and extending into the content layer of the *ICL*. The agent libraries preferably provide support for constructing, parsing, and manipulating *ICL* expressions. It is possible to embed content expressed in other languages within an *ICL* event. However, expressing content in *ICL* simplifies the facilitator's access to the content, as well as the conversational layer, in delegating requests. This gives the facilitator more information about the nature of a request and helps the facilitator decompose compound requests and delegate the sub-requests.

Further, *ICL* expressions preferably include, in addition to events, at least one of the following: capabilities declarations, requests for services, responses to requests, trigger specifications, and shared data elements. A further preferred embodiment of the present invention incorporates *ICL* expressions including at least all of the following: events, capabilities declarations, requests for services, responses to requests, trigger specifications, and shared data elements.

Providing Services: Specifying "Solvables"

In a preferred embodiment of the present invention, every participating agent defines and publishes a set of capability declarations, expressed in *ICL*, describing the services that it provides. These declarations establish a high-level interface to the agent. This interface is used by a facilitator in communicating with the agent, and, most important, in delegating service requests (or parts of requests) to the agent. Partly due to the use of PROLOG as a preferred basis for *ICL*, these capability declarations are referred as *solvables*. The agent library preferably provides a set of procedures allowing an agent to add, remove, and modify its solvables, which it may preferably do at any time after connecting to its facilitator.

There are preferably at least two major types of solvables: *procedure* solvables and *data* solvables. Intuitively, a procedure solvable performs a test or action, whereas a data solvable provides access to a collection of data. For example, in creating an agent for a mail system, procedure solvables might be defined for sending a message to a person, testing whether a message about a particular subject has arrived in the mail queue, or displaying a particular message onscreen. For a database wrapper agent, one might define a distinct data solvable corresponding to each of the relations present in the database. Often, a data solvable is used to provide a *shared* data store, which may be not only queried, but also updated, by various agents having the required permissions.

There are several primary technical differences between these two types of solvables. First, each procedure solvable must have a handler declared and defined for it, whereas this is preferably not necessary for a data solvable. The handling of requests for a data solvable is preferably provided transparently by the agent library. Second, data solvables are preferably associated with a dynamic collection of facts (or clauses), which may be further preferably modified at runtime, both by the agent providing the solvable, and by other agents (provided they have the required permissions). Third, special features, available for use with data solvables, preferably facilitate maintaining the associated facts. In spite of these differences, it should be noted that the mechanism of *use* by which an agent requests a service is the same for the two types of solvables.

In one embodiment, a request for one of an agent's services normally arrives in the form of an event from the agent's facilitator. The appropriate handler then deals with this event. The handler may be coded in whatever fashion is most appropriate, depending on the nature of the task, and the availability of task-specific libraries or legacy code, if any. The only hard requirement is that the handler return an appropriate response to the request, expressed in *ICL*. Depending on the nature of the request, this response could be an indication of success or failure, or a list of solutions (when the request is a data query).

A solvable preferably has three parts: a *goal*, a list of *parameters*, and a list of *permissions*, which are declared using the format:

solvable(Goal, Parameters, Permissions)

The goal of a solvable, which syntactically takes the preferable form of an *ICL* structure, is a logical representation of the service provided by the solvable. (An *ICL* structure consists of a *functor* with 0 or more arguments. For example, in the structure a(b,c), `a' is the functor, and `b' and `c' the arguments.) As with a PROLOG structure, the goal's arguments themselves may preferably be structures.

Various options can be included in the parameter list, to refine the semantics associated with the solvable. The *type* parameter is preferably used to say whether the solvable is *data* or *procedure*. When the type is *procedure*, another parameter may be used to indicate the handler to be associated with the solvable. Some of the parameters appropriate for a *data* solvable are mentioned elsewhere in this application. In either case (procedure or data solvable), the *private* parameter may be preferably used to restrict the use of a solvable to the declaring agent when the agent intends the solvable to be solely for its internal use but wishes to take advantage of the mechanisms in accordance with the present invention to access it, or when the agent wants the solvable to be available to outside agents only at selected times. In support of the latter case, it is preferable for the agent to change the status of a solvable from private to non-private at any time.

The permissions of a solvable provide mechanisms by which an agent may preferably control access to its services allowing the agent to restrict calling and writing of a solvable to itself and/or other selected agents. (*Calling* means requesting the service encapsulated by a solvable, whereas *writing* means modifying the collection of facts associated with a data solvable.) The default permission for every solvable in a further preferred embodiment of the present invention is to be callable by anyone, and for data solvables to be writable by anyone. A solvable's permissions can preferably be changed at any time, by the agent providing the solvable.

For example, the solvables of a simple email agent might include:

```
solvable(send_message(email, +ToPerson, +Params),
        [type(procedure), callback(send_mail)],
           [])
solvable(last_message(email, -MessageId),
           [type(data), single_value(true)],
           [write(true)]),
        solvable(get_message(email, +MessageId, -
Msg),
              [type(procedure), callback(get_mail)],
        [])
```

The symbols `+' and `-', indicating input and output arguments, are at present used only for purposes of documentation. Most parameters and permissions have default values, and specifications of default values may be omitted from the parameters and permissions lists.

Defining an agent's capabilities in terms of solvable declarations effectively creates a vocabulary with which other agents can communicate with the new agent. Ensuring that agents will speak the same language and share a common, unambiguous semantics of the vocabulary involves *ontology*. Agent development tools and services (automatic translations of solvables by the facilitator) help address this issue; additionally, a preferred embodiment of the present invention will typically rely on vocabulary from either formally engineered ontologies for specific domains or from ontologies constructed during the incremental development of a body of agents for several applications or from both specific domain ontologies and incrementally developed ontologies. Several example tools and services are described in Cheyer et al.'s paper entitled "Development Tools for the Open Agent Architecture," as presented at the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 96), London, April 1996.

Although the present invention imposes no hard restrictions on the form of solvable declarations, two common usage conventions illustrate some of the utility associated with solvables.

Classes of services are often preferably tagged by a particular type. For instance, in the example above, the "last_message" and "get_message" solvables are specialized for email, not by modifying the *names* of the services, but rather by the use of the `email' parameter, which serves during the execution of an *ICL* request to select (or not) a specific type of message.

Actions are generally written using an imperative verb as the functor of the solvable in a preferred embodiment of the present invention, the direct object (or item class) as the first argument of the predicate, required arguments following, and then an extensible parameter list as the last argument. The parameter list can hold optional information usable by the function. The *ICL* expression generated by a natural language parser often makes use of this parameter list to store prepositional phrases and adjectives.

As an illustration of the above two points, "Send mail to Bob about lunch" will be translated into an *ICL* request send_message(email, `Bob Jones', [subject(lunch)]), whereas "Remind Bob about lunch" would leave the transport unspecified (send_message(KIND, 'Bob Jones', [subject(lunch)])), enabling all available message transfer agents (e.g., fax, phone, mail, pager) to compete for the opportunity to carry out the request.

Requesting Services

An agent preferably requests services of the community of agent by delegating tasks or goals to its facilitator. Each request preferably contains calls to one or more agent solvables, and optionally specifies parameters containing advice to help the facilitator determine how to execute the task. Calling a solvable preferably does *not* require that the agent specify (or even know of) a particular agent or agents to handle the call. While it is possible to specify one or more agents using an address parameter (and there are situations in which this is desirable), in general it is advantageous to leave this delegation to the facilitator. This greatly reduces the hard-coded component dependencies often found in other distributed frameworks. The agent libraries of a preferred embodiment of the present invention provide an agent with a single, unified point of entry for requesting services of other agents: the library procedure *oaa_Solve*. In the style of logic programming, *oaa_Solve* may preferably be used both to retrieve data and to initiate actions, so that calling a *data* solvable looks the same as calling a *procedure* solvable.

Complex Goal Expressions

A powerful feature provided by preferred embodiments of the present invention is the ability of a client agent (or a user) to submit compound goals of an arbitrarily complex nature to a facilitator. A compound goal is a single goal expression that specifies multiple sub-goals to be performed. In speaking of a *"complex goal expression"* we mean that a single goal expression that expresses multiple sub-goals can potentially include more than one type of logical connector (e.g., AND, OR, NOT), and/or more than one level of logical nesting (e.g., use of parentheses), or the substantive equivalent. By way of further clarification, we note that when speaking of an *"arbitrarily complex goal expression"* we mean that goals are expressed in a language or syntax that allows expression of such complex goals when appropriate or when desired, not that every goal is itself necessarily complex.

It is contemplated that this ability is provided through an interagent communication language having the necessary syntax and semantics. In one example, the goals may take the form of compound goal expressions composed using operators similar to those employed by PROLOG, that is, the comma for conjunction, the semicolon for disjunction, the arrow for conditional execution, etc. The present invention also contemplates significant extensions to PROLOG syntax and semantics. For example, one embodiment incorporates a "parallel disjunction" operator indicating that the disjuncts are to be executed by different agents concurrently. A further embodiment supports the specification of whether a given sub-goal is to be executed breadth-first or depth-first.

A further embodiment supports each sub-goal of a compound goal optionally having an address and/or a set of parameters attached to it. Thus, each sub-goal takes the form

Address:Goal::Parameters

where both *Address* and *Parameters* are optional.

An address, if present, preferably specifies one or more agents to handle the given goal, and may employ several different types of referring expression: unique names, symbolic names, and shorthand names. Every agent has preferably a unique name, assigned by its facilitator, which relies upon network addressing schemes to ensure its global uniqueness. Preferably, agents also have self-selected symbolic names (for example, "mail"), which are not guaranteed to be unique. When an address includes a symbolic name, the facilitator preferably takes this to mean that all agents having that name should be called upon. Shorthand names include `self' and `parent' (which refers to the agent's facilitator). The address associated with a goal or sub-goal is preferably always optional. When an address is not present, it is the facilitator's job to supply an appropriate address.

The distributed execution of compound goals becomes particularly powerful when used in conjunction with natural language or speech-enabled interfaces, as the query itself may specify how functionality from distinct agents will be combined. As a simple example, the spoken utterance "Fax it to Bill Smith's manager." can be translated into the following compound *ICL* request:

oaa_Solve((manager('Bill Smith', M), fax(it,M,[])), [strategy(action)])

Note that in this ICL request there are two sub-goals, "manager('Bill Smith',M)" and "fax(it,M,[])," and a single global parameter "strategy(action)." According to the present invention, the facilitator is capable of mapping global parameters in order to apply the constraints or advice across the separate sub-goals in a meaningful way. In this instance, the global parameter strategy(action) implies a parallel constraint upon the first sub-goal; i.e., when there are multiple agents that can respond to the manager sub-goal, each agent should receive a request for service. In contrast, for the second sub-goal, parallelism should not be inferred from the global parameter strategy(action) because such an inference would possibly result in the transmission of duplicate facsimiles.

Refining Service Requests

In a preferred embodiment of the present invention, parameters associated with a goal (or sub-goal) can draw on useful features to refine the request's meaning. For example, it is frequently preferred to be able to specify whether or not solutions are to be returned synchronously; this is done using the *reply* parameter, which can take any of the values *synchronous, asynchronous*, or *none*. As another example, when the goal is a non-compound query of a data solvable, the *cache* parameter may preferably be used to request local caching of the facts associated with that solvable. Many of the remaining parameters fall into two categories: feedback and advice.

Feedback parameters allow a service requester to receive information from the facilitator about how a goal was handled. This feedback can include such things as the identities of the agents involved in satisfying the goal, and the amount of time expended in the satisfaction of the goal.

Advice parameters preferably give constraints or guidance to the facilitator in completing and interpreting the goal. For example, a *solution_limit* parameter preferably allows the requester to say how many solutions it is interested in; the facilitator and/or service providers are free to use this information in optimizing their efforts. Similarly, a *time_limit* is preferably used to say how long the requester is willing to wait for solutions to its request, and, in a multiple facilitator system, a *level_limit* may preferably be used to say how remote the facilitators may be that are consulted in the search for solutions. A *priority* parameter is preferably used to indicate that a request is more urgent than previous requests that have not yet been satisfied. Other preferred advice parameters include but are not limited to parameters used to tell the facilitator whether parallel satisfaction of the parts of a goal is appropriate, how to combine and filter results arriving from multiple solver agents, and whether the requester itself may be considered a candidate solver of the sub-goals of a request.

Advice parameters preferably provide an extensible set of low-level, orthogonal parameters capable of combining with the *ICL* goal language to fully express how information should flow among participants. In certain preferred embodiments of the present invention, multiple parameters can be grouped together and given a group name. The resulting *high-level advice parameters* can preferably be used to express concepts analogous to KQML's performatives, as well as define classifications of problem types. For instance, KQML's "ask_all" and "ask_one" performatives would be represented as combinations of values given to the parameters *reply*, *parallel_ok*, and *solution_limit*. As an example of a higher-level problem type, the strategy "math_problem" might preferably send the query to all appropriate math solvers in parallel, collect their responses, and signal a conflict if different answers are returned. The strategy "essay_question" might preferably send the request to all appropriate participants, and signal a problem (i.e., cheating) if any of the returned answers are identical.

Facilitation

In a preferred embodiment of the present invention, when a facilitator receives a compound goal, its job is to construct a goal satisfaction plan and oversee its satisfaction in an optimal or near optimal manner that is consistent with the specified advice. The facilitator of the present invention maintains a knowledge base that records the capabilities of a collection of agents, and uses that knowledge to assist requesters and providers of services in making contact.

Figure 7 schematically shows data structures 700 internal to a facilitator in accordance with one embodiment of the present invention. Consider the function of a Agent Registry 702 in the present invention. Each registered agent may be seen as associated with a collection of fields found within its parent facilitator such as shown in the figure. Each registered agent may optionally possess a Symbolic Name which would be entered into field 704. As mentioned elsewhere, Symbolic Names need not be unique to each instance of an agent. Note that an agent may in certain preferred embodiments of the present invention possess more than one Symbolic Name. Such Symbolic Names would each be found through their associations in the Agent Registry entries. Each agent, when registered, must possess a Unique Address, which is entered into the Unique Address field 706.

With further reference to Figure 7, each registered agent may be optionally associated with one or more capabilities, which have associated Capability Declaration fields 708 in the parent facilitator Agent Registry 702. These capabilities may define not just functionality, but may further provide a utility parameter indicating, in some manner (e.g., speed, accuracy, etc), how effective the agent is at providing the declared capability. Each registered agent may be optionally associated with one or more data components, which have associated Data Declaration fields 710 in the parent facilitator Agent Registry 702. Each registered agent may be optionally associated with one or more triggers, which preferably could be referenced through their associated Trigger Declaration fields 712 in the parent facilitator Agent Registry 702. Each registered agent may be optionally associated with one or more tasks, which preferably could be referenced through their associated Task Declaration fields 714 in the parent facilitator Agent Registry 702. Each registered agent may be optionally associated with one or more Process Characteristics, which preferably could be referenced through their associated Process Characteristics Declaration fields 716 in the parent facilitator Agent Registry 702. Note that these characteristics in certain preferred embodiments of the present invention may include one or more of the following: Machine Type (specifying what type of computer may run the agent), Language (both computer and human interface).

A facilitator agent in certain preferred embodiments of the present invention further includes a Global Persistent Database 720. The database 720 is composed of data elements which do not rely upon the invocation or instantiation of client agents for those data elements to persist. Examples of data elements which might be present in such a database include but are not limited to the network address of the facilitator agent's server, facilitator agent's server accessible network port list, firewalls, user lists, and security options regarding the access of server resources accessible to the facilitator agent.

A simplified walk through of operations involved in creating a client agent, a client agent initiating a service request, a client agent responding to a service request and a facilitator agent responding to a service request are including hereafter by way of illustrating the use of such a system. These figures and their accompanying discussion are provided by way of illustration of one preferred embodiment of the present invention and are not intended to limit the scope of the present invention.

Figure 8 depicts operations involved in instantiating a client agent with its parent facilitator in accordance with a preferred embodiment of the present invention. The operations begin with starting the Agent Registration in a step 800. In a next step 802, the Installer, such as a client or facilitator agent, invokes a new client agent. It will be appreciated that any computer entity is capable of invoking a new agent. The system then instantiates the new client agent in a step 804. This operation may involve resource allocations somewhere in the network on a local computer system for the client agent, which will often include memory as well as placement of references to the newly instantiated client agent in internal system lists of agents within that local computing system. Once instantiated, the new client and its parent facilitator establish a communications link in a step 806. In certain preferred embodiments, this communications link involves selection of one or more physical transport mechanisms for this communication. Once established, the client agent transmits it profile to the parent facilitator in a step 808. When received, the parent facilitator registers the client agent in a step 810. Then, at a step 812, a client agent has been instantiated in accordance with one preferred embodiment of the present invention.

Figure 9 depicts operations involved in a client agent initiating a service request and receiving the response to that service request in accordance with a preferred embodiment of the present invention. The method of Figure 9 begins in a step 900, wherein any initialization or other such procedures may be performed. Then, in a step 902, the client agent determines a goal to be achieved (or solved). This goal is then translated in a step 904 into *ICL*, if it is not already formulated in it. The goal, now stated in *ICL*, is then transmitted to the client agent's parent facilitator 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, 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.

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 delegation is possible because agents' capabilities (solvables) 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.

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 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 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 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 *interpretation* of a goal involves the coordination of requests to the satisfying agents, and assembling their responses into a coherent whole, for return to the requester.

A further preferred embodiment of present invention extends facilitation so the facilitator can employ strategies and advice given by the requesting agent, resulting in a variety of interaction patterns that may be instantiated in the satisfaction of a request.

A further preferred embodiment of present invention handles the distribution of both data update requests and requests for installation of triggers, preferably using some of the same strategies that are employed in the delegation of service requests.

Note that the reliance on facilitation is not absolute; that is, there is no hard requirement that requests and services be matched up by the facilitator, or that interagent communications go through the facilitator. There is preferably support in the agent library for explicit addressing of requests. However, a preferred embodiment of the present invention encourages employment the paradigm of agent communities, minimizing their development effort, by taking advantage of the facilitator's provision of transparent delegation and handling of compound goals.

A facilitator is preferably viewed as a *coordinator*, not a controller, of cooperative task completion. A facilitator preferably never initiates an activity. A facilitator preferably responds to requests to manage the satisfaction of some goal, the update of some data repository, or the installation of a trigger by the appropriate agent or agents. All agents can preferably take advantage of the facilitator's expertise in delegation, and its up-to-date knowledge about the current membership of a dynamic community. The facilitator's coordination services often allows the developer to lessen the complexity of individual agents, resulting in a more manageable software development process, and enabling the creation of lightweight agents.

Maintaining Data Repositories

The agent library supports the creation, maintenance, and use of databases, in the form of data solvables. Creation of a data solvable requires only that it be declared. Querying a data solvable, as with access to any solvable, is done using *oaa_Solve*.

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 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 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 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 (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 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 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: "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 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 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 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."

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 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 *intra*agent and *inter*agent 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 *intra*agent with *inter*agent 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.

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.

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 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. 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.

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.

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:
       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 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 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:
       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.

```
1.4  FIND ALL NEW MAIL MESSAGES.
     There are 2 messages available.
     Message 1, from Mark Tierny, entitled "OAA meeting."
1.5  NEXT MESSAGE
     Message 2, from Jennifer Schwefler, entitled "Presentation Summary."
1.6  PLAY IT.
     This message is a multipart MIME-encoded message. There are two parts.
     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
     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.".

```
print(Object, Parameters) {
   ' If Object is reference to "it", find an appropriate
document
   if (Object = "ref(it)")
      oaa_Solve(resolve_reference(the, document, Params,
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,
[], 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))
      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
     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
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 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 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 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 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, 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.
       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.
       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 hotel.
       Photo appears.
2.7    [Points to another hotel]
       Photo appears.
2.8    [Speaking] Price of the other hotel?
       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]
       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.
```

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.

```
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 walk there from
```

```
        this hot .
        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.
        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.*

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 MVIEWS 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 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 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.

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, NewIParams),
    new_goal_id(NewId),
    assert(
        continuation(NewId, NewRequestees, NewSolvers, Bindings,
          continue_local_cut(Id, RKS, NewRequestees, NewSolvers, G2, false,
NewIParams,
                         Vars, Bindings))),
    execute(NewId, RKS, Requestees, Solvers, G1, NewIParams, Vars).

    % Disjunction:
execute(Id, RKS, Requestees, Solvers, ICLGoal, InheritedParams, Vars) :-
    icl_GoalComponents(ICLGoal, _, (G1; G2), Params),
    !,
    append(Params, InheritedParams, NewIParams),
    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, NewIParams, Vars),
    execute(Id2, RKS, Requestees, Solvers, G2, NewIParams, 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, NewIParams),
    execute(Id, RKS, Requestees, Solvers, Goal, NewIParams, 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(NumKSs, Ids, KSs, RKS, _Requestees, _Solvers,
             Goal, Params, Vars) :-
    retractall( ks_num(_) ),
    assert( ks_num(1) ),
    repeat,
    ks_num(Num),
    ( Num > NumKSs ->
       !
    | 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 deconnects */
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:oaa_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, 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, 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_solvables(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_solvables(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_solvables(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_solvables(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_solvables/1.
ks_ready(SomeKS, ListOfGoals) :-
    oaa_Id(SomeKS),
    oaa:oaa_solvables(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:oaa_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, 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_trigger_event(Id, RKS, Mode, Type, Condition, Action, Params,
SomeKS),
    fail.
dispatch_trigger_request(_RKS, _Mode, _Type, _Condition, _Action, _Params,
                   _KSList).

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_solvables(Solvables),
    com_ListenAt(incoming, CInfo),
    format('Listening at ~p~n~n', [CInfo]),
    oaa_RegisterCallback(app_do_event, user:oaa_AppDoEvent),
    oaa_Register(incoming, 'root', Solvables),
    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, commtype, 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, '[y)es, n)o, h)elp]']),
             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/oaa.pl,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_solvables/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_solvables fact.  Note that application
% code should NOT include a declaration or clause for oaa_solvables/1.
oaa_solvables([]).



%*****************************************************************************
% 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 | Addrs], [SAddr | SAddrs]) :-
    icl_standardize_addressee(Addr, SAddr),
    !,
    icl_standardize_address(Addrs, SAddrs).
icl_standardize_address([_Addr | Addrs], SAddrs) :-
    icl_standardize_address(Addrs, 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_solvables(+StandardSolvables,
    %                                  -ShorthandSolvables).
    % This is provided for use in "pretty-printing" solvables, in trace
    % messages, etc.
icl_readable_solvables([], []).
icl_readable_solvables([Standard | RestStan], [Shorthand | RestSh]) :-
    icl_readable_solvable(Standard, Shorthand),
    icl_readable_solvables(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_solvables(+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_solvables(Goal, Solvables, RealGoal, RealMatched) :-
    oaa_built_in_solvables(BuiltIns),
    append(BuiltIns, Solvables, AllSolvables),
    oaa_goal_in_solvables(Goal, AllSolvables, Matched),
    Matched = solvable(_, Params, _),
```

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


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_goal_in_solvables(+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_solvables.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_goal_in_solvables(Goal, [solvable(G1,Params,Perms) | _Rest],
                  solvable(G1,Params,Perms)) :-
      would_unify(Goal, G1),
        icl_GetParamValue(synonym(Goal, _RealGoal), Params),
      !.
oaa_goal_in_solvables(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_solvables(Goal, [_|Rest], Matched) :-
      oaa_goal_in_solvables(Goal, Rest, Matched).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_data_matches_solvables(+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_solvables(Clause, Solvables, Perm, RealClause, RealMatched) :-
    oaa_built_in_solvables(BuiltIns),
    append(BuiltIns, Solvables, AllSolvables),
    oaa_data_in_solvables(Clause, AllSolvables, Perm, Matched),
    Matched = solvable(_, Params, _),
    ( Clause = (Head :- Body) ->
      true
    | otherwise ->
      Head = Clause
    ),
    % See if Clause is a synonym predicate
    ( icl_GetParamValue(synonym(Head, SynHead), Params) ->
        ( Clause = (Head :- Body) ->
          SynClause = (SynHead :- Body)
      | otherwise ->
          SynClause = SynHead
      ),
        oaa_data_matches_solvables(SynClause, Solvables, Perm,
                                   RealClause, RealMatched)
    | otherwise ->
        RealClause = Clause,
      RealMatched = Matched
    ),
    !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_data_in_solvables(+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_solvables.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_data_in_solvables(Clause, [solvable(G1,Params,Perms) | _Rest], _Perm,
                  solvable(G1,Params,Perms) ) :-
        ( Clause = (Head :- _Body) ->
            true
      | otherwise ->
          Head = Clause
      ),
      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_solvables(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_solvables(Clause, [_|Rest], Perm, Matched) :-
      oaa_data_in_solvables(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(+ICLExpression, +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(fail, _) :- !, 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_solvables(Solvables) -> true | otherwise -> Solvables = []),
      ( (oaa_goal_matches_solvables(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, becuase 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,KSList) :-
    oaa_Solve(can_solve(Goal, KSList), [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(Solvs, Params, DeclaredSolvs) :-
    % It's only necessary to specify the non-default perms and params.
    CommonPerms = [write(true)],
    CommonParams = [type(data)],
    oaa_Declare(Solvs, 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_solvables_data([Solvable])
    | true
    ),
    % Assert the new solvables list:
    retractall(oaa_solvables(_)),
    replace_element(Solvable, Current, NewSolvable, New),
    assert(oaa_solvables(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).
```

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


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name: solvables_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.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
solvables_to_be_removed([], _Current, []).
solvables_to_be_removed([Solvable | Solvables], Current,
                  [Solvable | OKSolvables]) :-
    memberchk(Solvable, Current),
    !,
    solvables_to_be_removed(Solvables, Current, OKSolvables).
solvables_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]),
    solvables_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(Clause1, Clause2, Params) :-
    oaa_update(replace, Clause1, [with(Clause2) | 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_solvables(Solvables) -> true | otherwise -> Solvables = []),
    oaa_data_matches_solvables(Clause1, Solvables, 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_solvables(Solvables) -> true | otherwise -> Solvables = []),
    oaa_data_matches_solvables(Clause1, Solvables, 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(Clause1In, Params) :-
    memberchk(with(Clause2In), Params),
    ( oaa_solvables(Solvables) -> true | otherwise -> Solvables = []),
    oaa_data_matches_solvables(Clause1In, 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(Clause1, Clause2, Owner, Callback)
        | otherwise ->
            oaa_retract(Clause1, _OldOwner, Callback),
            (icl_GetParamValue(at_beginning(true), AllParams) ->
               oaa_asserta(Clause2, Owner, Callback)
          |  oaa_assertz(Clause2, Owner, Callback)
          )
      )
    | otherwise ->
        ( icl_GetParamValue(do_all(true), Params) ->
            replace_all(Clause1, Clause2)
      | otherwise ->
            retract(Clause1),
            (icl_GetParamValue(at_beginning(true), AllParams) ->
               asserta(Clause2)
          |  assertz(Clause2)
          )
      )
    ),
    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(Clause1) :-
   retract(Clause1),
```

```
   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(Clause)])
    ),
    fail.
oaa_retractall(_Clause, _Owner, _Callback).

oaa_replace_all(Clause1, Clause2, Owner, Callback) :-
   oaa_retract(Clause1, _OldOwner, Callback),
   oaa_assertz(Clause2, Owner, Callback),
   % This would be redundant:
   % oaa_call_callback(app_on_data_change, Callback, [replace(Clause1,
Clause2)]),
   fail.
oaa_replace_all(_Clause1, _Clause2, _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_solvables(Solvables) -> true | otherwise -> Solvables = []),
    ( icl_compound_goal(Goal1) ;
      icl_BuiltIn(Goal1) ;
      oaa_goal_matches_solvables(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_solvables_data(Solvables).
% 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_solvables_data([]).
oaa_remove_solvables_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_solvables_data(Solvables).
oaa_remove_solvables_data([_Solvable | Solvables]) :-
    oaa_remove_solvables_data(Solvables).

oaa_remove_data_owned_by(Id) :-
    ( oaa_solvables(Solvables) -> true | otherwise -> Solvables = []),
    oaa_built_in_solvables(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, [Y|Rest], New, [Y|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:oaa_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:oaa_solve_local(agent_data(Id, _, _, Name), []),
    oaa:oaa_add_data_local(agent_host(Id, Name, Host),
                           [from(Id) | EvParams]).
oaa_AppDoEvent(write_bb(Item, Data), EvParams) :-
    !,
    memberchk(from(Id), EvParams),
    oaa:oaa_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:oaa_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:oaa_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:oaa_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:

See G1

1. A computer-implemented method for communication and cooperative task completion among a plurality of distributed electronic agents, comprising the acts of:

registering a description of each active client agent's functional capabilities, using an expandable, platform-independent, inter-agent language;

receiving a request for service as a base goal in the inter-agent language, in the form of an arbitrarily complex goal expression; and

dynamically interpreting the goal expression, said act of interpreting further 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, based on a match between the sub-goal being dispatched and the registered functional capabilities of the selected client agent.

2. A computer-implemented method as recited in claim 1, further including the following acts of:

receiving a new request for service as a base goal using the inter-agent language, in the form of another arbitrarily complex goal expression, from at least one of the selected client agents in response to the sub-goal dispatched to said agent; and

recursively applying the last step of claim 1 in order to perform the new request for service.

3. A computer implemented method as recited in claim 2 wherein the act of registering a specific agent further includes:

invoking the specific agent in order to activate the specific agent;

instantiating an instance of the specific agent; and

transmitting the new agent profile from the specific agent to the facilitator agent in response to the instantiation of the specific agent.

Sub B1

4. A computer implemented method as recited in claim 1 further including the act of deactivating a specific client agent no longer available to provide services by deleting the registration of the specific client agent.

5. A computer implemented method as recited in claim 1 further comprising the act of providing an agent registry data structure.

6. A computer implemented method as recited in claim 5 wherein the agent registry data structure includes at least one symbolic name for each active agent.

7. A computer implemented method as recited in claim 5 wherein the agent registry data structure includes at least one data declaration for each active agent.

8. A computer implemented method as recited in claim 5 wherein the agent registry data structure includes at least one trigger declaration for one active agent.

9. A computer implemented method as recited in claim 5 wherein the agent registry data structure includes at least one task declaration, and process characteristics for each active agent.

10. A computer implemented method as recited in claim 5 wherein the agent registry data structure includes at least one process characteristic for each active agent.

11. A computer implemented method as recited in claim 1 further comprising the act of establishing communication between the plurality of distributed agents.

12. A computer implemented method as recited in claim 1 further comprising the acts of:

receiving a request for service in a second language differing from the inter-agent language;

selecting a registered agent capable of converting the second language into the inter-agent language; and

forwarding the request for service in a second language to the registered agent capable of converting the second language into the inter-agent language, implicitly requesting that such a conversion be performed and the results returned.

13. A computer implemented method as recited in claim 12 wherein the request includes a natural language query, and the registered agent capable of converting the second language into the inter-agent language service is a natural language agent.

14. A computer implemented method as recited in claim 13 wherein the natural language query was generated by a user interface agent.

15. A computer implemented method as recited in claim 1, wherein the base goal requires setting a trigger having conditional functionality and consequential functionality.

16. A computer implemented method as recited in claim 15 wherein the trigger is an outgoing communications trigger, the computer implemented method further including the acts of:

monitoring all outgoing communication events in order to determine whether a specific outgoing communication event has occurred; and

in response to the occurrence of the specific outgoing communication event, performing the particular action defined by the trigger.

17. A computer implemented method as recited in claim 15 wherein the trigger is an incoming communications trigger, the computer implemented method further including the acts of:

monitoring all incoming communication events in order to determine whether a specific incoming communication event has occurred; and

in response to the occurrence of a specific incoming communication event satisfying the trigger conditional functionality, performing the particular consequential functionality defined by the trigger.

18. A computer implemented method as recited in claim 15 wherein the trigger is a data trigger, the computer implemented method further including the acts of:

monitoring a state of a data repository; and

in response to a particular state event satisfying the trigger conditional functionality, performing the particular consequential functionality defined by the trigger.

19. A computer implemented method as recited in claim 15 wherein the trigger is a time trigger, the computer implemented method further including the acts of:

monitoring for the occurrence of a particular time condition; and

in response to the occurrence of a particular time condition satisfying the trigger conditional functionality, performing the particular consequential functionality defined by the trigger.

20. A computer implemented method as recited in claim 15 wherein the trigger is installed and executed within the facilitator agent.

21. A computer implemented method as recited in claim 15 wherein the trigger is installed and executed within a first service-providing agent.

22. A computer implemented method as recited in claim 15 wherein the conditional functionality of the trigger is installed on a facilitator agent.

23. A computer implemented method as recited in claim 22 wherein the consequential functionality is installed on a specific service-providing agent other than a facilitator agent.

24. A computer implemented method as recited in claim 15 wherein the conditional functionality of the trigger is installed on a specific service-providing agent other than a facilitator agent.

25. A computer implemented method as recited in claim 15 wherein the consequential functionality of the trigger is installed on a facilitator agent.

26. A computer implemented method as recited in claim 1 wherein the base goal is a compound goal having sub-goals separated by operators.

27. A computer implemented method as recited in claim 26 wherein the type of available operators includes a conjunction operator, a disjunction operator, and a conditional execution operator.

28. A computer implemented method as recited in claim 27 wherein the type of available operators further includes a parallel disjunction operator that indicates that disjunct goals are to be performed by different agents.

29. A computer program stored on a computer readable medium, the computer program executable to facilitate cooperative task completion within a distributed computing environment, the distributed computing environment including a plurality of autonomous electronic agents, the distributed computing environment supporting an Interagent Communication Language, the computer program comprising computer executable instructions for:

providing an agent registry that declares capabilities of service-providing electronic agents currently active within the distributed computing environment;

interpreting a service request in order to determine a base goal that may be a compound, arbitrarily complex base goal, the service request adhering to an Interagent Communication Language (ICL), the act of interpreting including the sub-acts of:

determining any task completion advice provided by the base goal, and

determining any task completion constraints provided by the base goal;

constructing a base goal satisfaction plan including the sub-acts of:

determining whether the requested service is available,

determining sub-goals required in completing the base goal,

selecting service-providing electronic agents from the agent registry suitable for performing the determined sub-goals, and

ordering a delegation of sub-goal requests to best complete the requested service; and

implementing the base goal satisfaction plan.

30. A computer program as recited in claim 29 wherein the computer executable instruction for providing an agent registry includes the following computer executable instructions for registering a specific service-providing electronic agent into the agent registry:

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, the new agent profile defining publicly available capabilities of the specific agent; and

registering the specific agent together with the new agent profile within the agent registry, thereby making available to the facilitator agent the capabilities of the specific agent.

31. A computer program as recited in claim 30 wherein the computer executable instruction for registering a specific agent further includes:

invoking the specific agent in order to activate the specific agent;

instantiating an instance of the specific agent; and

transmitting the new agent profile from the specific agent to the facilitator agent in response to the instantiation of the specific agent.

32. A computer program as recited in claim 29 wherein the computer executable instruction for providing an agent registry includes a computer executable instruction for removing a specific service-providing electronic agent from the registry upon determining that the specific agent is no longer available to provide services.

33. A computer program as recited in claim 29 wherein the provided agent registry includes a symbolic name, a unique address, data declarations, trigger declarations, task declarations, and process characteristics for each active agent.

34. A computer program as recited in claim 29 further including computer executable instructions for receiving the service request via a communications link established with a client.

35. A computer program as recited in claim 29 wherein the computer executable instruction for providing a service request includes instructions for:

receiving a non-ICL format service request;

selecting an active agent capable of converting the non-ICL formal service request into an ICL format service request;

forwarding the non-ICL format service request to the active agent capable of converting the non-ICL format service request, together with a request that such conversion be performed; and

receiving an ICL format service request corresponding to the non-ICL format service request.

36. A computer program as recited in claim 35 wherein the non-ICL format service request includes a natural language query, and the active agent capable of converting the non-ICL formal service request into an ICL format service request is a natural language agent.

37. A computer program as recited in claim 36 wherein the natural language query is generated by a user interface agent.

38. A computer program as recited in claim 29, the computer program further including computer executable instructions for implementing a base goal that requires setting a trigger having conditional and consequential functionality.

39. A computer program as recited in claim 38 wherein the trigger is an outgoing communications trigger, the computer program further including computer executable instructions for:

monitoring all outgoing communication events in order to determine whether a specific outgoing communication event has occurred; and

in response to the occurrence of the specific outgoing communication event, performing the particular action defined by the trigger.

40. A computer program as recited in claim 38 wherein the trigger is an incoming communications trigger, the computer program further including computer executable instructions for:

monitoring all incoming communication events in order to determine whether a specific incoming communication event has occurred; and

in response to the occurrence of the specific incoming communication event, performing the particular action defined by the trigger.

41. A computer program as recited in claim 38 wherein the trigger is a data trigger, the computer program further including computer executable instructions for:

monitoring a state of a data repository; and

in response to a particular state event, performing the particular action defined by the trigger.

42. A computer program as recited in claim 38 wherein the trigger is a time trigger, the computer program further including computer executable instructions for:

monitoring for the occurrence of a particular time condition; and

in response to the occurrence of the particular time condition, performing the particular action defined by the trigger.

43. A computer program as recited in claim 38 further including computer executable instructions for installing and executing the trigger within the facilitator agent.

44. A computer program as recited in claim 38 further including computer executable instructions for installing and executing the trigger within a first service-providing agent.

45. A computer program as recited in claim 29 further including computer executable instructions for interpreting compound goals having sub-goals separated by operators.

46. A computer program as recited in claim 45 wherein the type of available operators includes a conjunction operator, a disjunction operator, and a conditional execution operator.

47. A computer program as recited in claim 46 wherein the type of available operators further includes a parallel disjunction operator that indicates that disjunct goals are to be performed by different agents.

Sub A2

~~48.~~ An Interagent Communication Language (ICL) providing a basis for facilitated cooperative task completion within a distributed computing environment having a facilitator agent and a plurality of autonomous service-providing electronic agents, the ICL enabling agents to perform queries of other agents, exchange information with other agents, set triggers within other agents, 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 disjunctive operator parallel disjunctive operator that indicates that disjunct goals are to be performed by different agents.

49. An ICL as recited in claim 48, wherein the ICL is computer platform independent.

50. An ICL as recited in claim 48 wherein the ICL is independent of computer programming languages which the plurality of agents are programmed in.

51. An ICL as recited in claim 48 wherein the ICL syntax supports explicit task completion constraints within goal expressions.

52. An ICL as recited in claim 51 wherein possible types of task completion constraints include use of specific agent constraints and response time constraints.

53. An ICL as recited in claim 51 wherein the ICL syntax supports explicit task completion advisory suggestions within goal expressions.

54. An ICL as recited in claim 48 wherein the ICL syntax supports explicit task completion advisory suggestions within goal expressions.

55. An ICL as recited in claim 48 wherein each autonomous service-providing electronic agent defines and publishes a set of capability declarations or solvables, expressed in ICL, that describes services provided by such electronic agent.

56. An ICL as recited in claim 55 wherein an electronic agent's solvables define an interface for the electronic agent.

57. An ICL as recited in claim 56 wherein the facilitator agent maintains an agent registry making available a plurality of electronic agent interfaces.

58. An ICL as recited in claim 57 wherein the possible types of solvables includes procedure solvables, a procedure solvable operable to implement a procedure such as a test or an action.

59. An ICL as recited in claim 58 wherein the possible types of solvables further includes data solvables, a data solvable operable to provide access to a collection of data.

60. An ICL as recited in claim 58 wherein the possible types of solvables includes data solvables, a data solvable operable to provide access to a collection of data.

61. A facilitator agent arranged to coordinate cooperative task completion within a distributed computing environment having a plurality of autonomous service-providing electronic agents, the facilitator agent comprising:

an agent registry that declares capabilities of service-providing electronic agents currently active within the distributed computing environment; and

a facilitating engine operable to parse a service request in order to interpret a compound goal set forth therein, the compound goal including both local and global constraints and control parameters, the service request formed according to an Interagent Communication Language (ICL), the facilitating engine further operable to construct a goal satisfaction plan specifying the coordination of a suitable delegation of sub-goal requests to complete the requested service satisfying both the local and global constraints and control parameters.

62. A facilitator agent as recited in claim 61, wherein the facilitating engine is capable of modifying the goal satisfaction plan during execution, the modifying initiated by events such as new agent declarations within the agent registry, decisions made by remote agents, and information provided to the facilitating engine by remote agents.

63. A facilitator agent as recited in claim 61 wherein the agent registry includes a symbolic name, a unique address, data declarations, trigger declarations, task declarations, and process characteristics for each active agent.

64. A facilitator agent as recited in claim 61 wherein the facilitating engine is operable to install a trigger mechanism requesting that a certain action be taken when a certain set of conditions are met.

65. A facilitator agent as recited in claim 64 wherein the trigger mechanism is a communication trigger that monitors communication events and performs the certain action when a certain communication event occurs.

66. A facilitator agent as recited in claim 64 wherein the trigger mechanism is a data trigger that monitors a state of a data repository and performs the certain action when a certain data state is obtained.

67. A facilitator agent as recited in claim 66 wherein the data repository is local to the facilitator agent.

68. A facilitator agent as recited in claim 66 wherein the data repository is remote from the facilitator agent.

69. A facilitator agent as recited in claim 64 wherein the trigger mechanism is a task trigger having a set of conditions.

70. A facilitator agent as recited in claim 61, the facilitator agent further including a global database accessible to at least one of the service-providing electronic agents.

71. A software-based, flexible computer architecture for communication and cooperation among distributed electronic agents, the architecture contemplating a distributed computing system comprising:

a plurality of service-providing electronic agents; and

a facilitator agent in bi-directional communications with the plurality of service-providing electronic agents, the facilitator agent including:

an agent registry that declares capabilities of service-providing electronic agents currently active within the distributed computing environment;

a facilitating engine operable to parse a service request in order to interpret an arbitrarily complex goal set forth therein, the facilitating engine further operable to construct a goal satisfaction plan including the coordination of a suitable delegation of sub-goal requests to best complete the requested service.

72. A computer architecture as recited in claim 71, wherein the basis for the computer architect is an Interagent Communication Language (ICL) enabling agents to perform queries of other agents, exchange information with other agents, and set triggers within other agents, the ICL further defined by 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 disjunctive operator parallel disjunctive operator that indicates that disjunct goals are to be performed by different agents.

73. A computer architecture as recited in claim 72, wherein the ICL is computer platform independent.

74. A computer architecture as recited in claim 73 wherein the ICL is independent of computer programming languages in which the plurality of agents are programmed.

75. A computer architecture as recited in claim 73 wherein the ICL syntax supports explicit task completion constraints within goal expressions.

76. A computer architecture as recited in claim 75 wherein possible types of task completion constraints include use of specific agent constraints and response time constraints.

77. A computer architecture as recited in claim 75 wherein the ICL syntax supports explicit task completion advisory suggestions within goal expressions.

78. A computer architecture as recited in claim 73 wherein the ICL syntax supports explicit task completion advisory suggestions within goal expressions.

79. A computer architecture as recited in claim 73 wherein each autonomous service-providing electronic agent defines and publishes a set of capability declarations or solvables, expressed in ICL, that describes services provided by such electronic agent.

80. A computer architecture as recited in claim 79 wherein an electronic agent's solvables define an interface for the electronic agent.

81. A computer architecture as recited in claim 80 wherein the possible types of solvables includes procedure solvables, a procedure solvable operable to implement a procedure such as a test or an action.

82. A computer architecture as recited in claim 81 wherein the possible types of solvables further includes data solvables, a data solvable operable to provide access to a collection of data.

83. A computer architecture as recited in claim 82 wherein the possible types of solvables includes a data solvable operable to provide access to modify a collection of data.

84. A computer architecture as recited in claim 71 wherein the planning component of the facilitating engine are distributed across at least two computer processes.

85. A computer architecture as recited in claim 71 wherein the execution component of the facilitating engine is distributed across at least two computer processes.

~~86.~~ A data wave carrier providing a transport mechanism for information communication in a distributed computing environment having at least one facilitator agent and at least one active client agent, the data wave carrier comprising a signal representation of an inter-agent language description of an active client agent's functional capabilities.

87. A data wave carrier as recited in claim 85, the data wave carrier further comprising a signal representation of a request for service in the inter-agent language from a first agent to a second agent.

88. A data wave carrier as recited in claim 85, the data wave carrier further comprising a signal representation of a goal dispatched to an agent for performance from a facilitator agent.

89. A data wave carrier as recited in claim 88 wherein a later state of the data wave carrier comprises 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.

Software-Based Architecture for Communication and Cooperation Among Distributed Electronic Agents

ABSTRACT

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 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 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 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.

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 DLM Stephens & Coleman, LLP,

And I hereby appoint the law firm of Hickman ~~& Martine~~, 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.

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 CRONIN DR. Santa Clara, CA 95051

16

PATENT APPLICATION FEE DETERMINATION RECORD

Effective November 10, 1998

Application or Docket Number: 09/225,198

CLAIMS AS FILED - PART I

FOR	NUMBER FILED (Column 1)	NUMBER EXTRA (Column 2)	SMALL ENTITY RATE	SMALL ENTITY FEE	OR	OTHER THAN SMALL ENTITY RATE	OTHER THAN SMALL ENTITY FEE
BASIC FEE				380.00	OR		760.00
TOTAL CLAIMS	89 minus 20=	* 69	X$ 9=		OR	X$18=	1242
INDEPENDENT CLAIMS	6 minus 3 =	* 3	X39=		OR	X78=	234
MULTIPLE DEPENDENT CLAIM PRESENT			+130=		OR	+260=	
			TOTAL		OR	TOTAL	2236

SMALL ENTITY TYPE ☐

* If the difference in column 1 is less than zero, enter "0" in column 2

CLAIMS AS AMENDED - PART II

AMENDMENT A

	CLAIMS REMAINING AFTER AMENDMENT (Column 1)		HIGHEST NUMBER PREVIOUSLY PAID FOR (Column 2)	PRESENT EXTRA (Column 3)	SMALL ENTITY RATE	ADDITIONAL FEE	OR	OTHER THAN SMALL ENTITY RATE	ADDITIONAL FEE
Total	* 89	Minus	** 20	= 69	X$ 9=		OR 69	X$18=	1242
Independent	* 6	Minus	*** 3	= 3	X39=		OR 3	X78=	234.00
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM					+130=		OR	+260=	
					TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT B

	CLAIMS REMAINING AFTER AMENDMENT (Column 1)		HIGHEST NUMBER PREVIOUSLY PAID FOR (Column 2)	PRESENT EXTRA (Column 3)	SMALL ENTITY RATE	ADDITIONAL FEE	OR	OTHER THAN SMALL ENTITY RATE	ADDITIONAL FEE
Total	*	Minus	**	=	X$ 9=		OR	X$18=	
Independent	*	Minus	***	=	X39=		OR	X78=	
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM					+130=		OR	+260=	
					TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT C

	CLAIMS REMAINING AFTER AMENDMENT (Column 1)		HIGHEST NUMBER PREVIOUSLY PAID FOR (Column 2)	PRESENT EXTRA (Column 3)	SMALL ENTITY RATE	ADDITIONAL FEE	OR	OTHER THAN SMALL ENTITY RATE	ADDITIONAL FEE
Total	*	Minus	**	=	X$ 9=		OR	X$18=	
Independent	*	Minus	***	=	X39=		OR	X78=	
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM					+130=		OR	+260=	
					TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If th "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20."
*** If th "Highest Number Previously Paid F r" IN THIS SPACE is less than 3, enter "3."
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in th appropriate box in column 1.

Best Available Copy

PATENT APPLICATION SERIAL NO. ____________________

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE
FEE RECORD SHEET

01/19/1999 NVILLARI 00000027 500384 09225198

01 FC:101 760.00 CH
02 FC:102 234.00 CH
03 FC:103 1242.00 CH

PTO-1556
(5/87)

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.

- [x] CD(s) containing:
  - [x] computer program listing
    Doc Code: Computer — Artifact Type Code: P
  - [ ] pages of specification and/or sequence listing and/or table
    Doc Code: Artifact — Artifact Type Code: S
  - [ ] content unspecified or combined
    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

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

09225198PB

Indicate quantity of a single type of artifact received but not scanned. Create individual artifact folder/box and artifact number for each Artifact Type.

- [x] CD(s) containing:
  - [x] computer program listing
    Doc Code: Computer — Artifact Type Code: P
  - [ ] pages of specification and/or sequence listing and/or table
    Doc Code: Artifact — Artifact Type Code: S
  - [ ] content unspecified or combined
    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

G-8 2755

PATENT

#RS 2

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re the application of:	)	Group: 2755
Cheyer et al.	)	Examiner: Unassigned
Application No.: 09/225,198	)	Atty. Docket No.: SRI1P016
Filed: January 5, 1999	)	Date: May 11, 1999
For: SOFTWARE-BASED ARCHITECTURE FOR COMMUNICATION AND COOPERATION AMONG DISTRIBUTED ELECTRONIC AGENTS	)	

O I P E JC48 MAY 14 1999 PATENT & TRADEMARK OFFICE

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.

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	

DATE MAILED: 07/17/2002

Please find below and/or attached an Office communication concerning this application or proceeding.

PTO-90C (Rev. 07-01)

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. conjuction 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 facilator 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 negatived 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.

ST. JOHN COURTENAY III
PRIMARY EXAMINER

***
July 11, 2002

Form PTO 948 (Rev. 8-98) U.S. DEPARTMENT OF COMMERCE - Patent and Trademark Office Application No. 09/225,198

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 sumitted according to the instructions on the back of this notice.

1. DRAWINGS. 37 CFR 1.84(a): Acceptable categories of drawings: Black ink. Color.
   ____ Color drawings are not acceptable until petiton is granted. Fig(s) ____
   ____ Pencil and non black ink not permitted. Fig(s) ____
2. PHOTOGRAPHS. 37 CFR 1.84 (b)
   ____ 1 full-tone set is required. Fig(s) ____
   ____ Photographs not properly mounted (must use brystol board or photographic double-weight paper). Fig(s) ____
   ____ Foor quality (half-tone). Fig(s) ____
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) ____
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) ____
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)
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 separetely or properly. Fig(s) ____
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) ____
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) ____
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) ____
10. CHARACTER OF LINES, NUMBERS, & LETTERS. 37 CFR 1.84(l)
    ____ Lines, numbers & letters not uniformly thick and well defined, clean, durable, and black (poor line quality). Fig(s) ____
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) ____
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) ____
13. LEAD LINES. 37 CFR 1.84(q)
    ____ Lead lines cross each other. Fig(s) ____
    ____ Lead lines missing. Fig(s) ____
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) ____
15. NUMBERING OF VIEWS. 37 CFR 1.84(u)
    ____ Views not numbered consecutively, and in Arabic numerals, beginning with number 1. Fig(s) ____
16. CORRECTIONS. 37 CFR 1.84(w)
    ____ Corrections not made from prior PTO-948 dated ____
17. DESIGN DRAWINGS. 37 CFR 1.152
    ____ Surface shading shown not appropriate. Fig(s) ____
    ____ Solid black shading not used for color contrast. Fig(s) ____

COMMENTS

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	Cheyer, 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.

IPE JC48
MAY 1 4 1999
PATENT & TRADEMARK OFFICE

Form 1449 (Modified) **Information Disclosure Statement By Applicant** (Use Several Sheets if Necessary)	Atty Docket No. SRI1P016 Applicant: Cheyer et al. Filing Date: January 5. 1999	Serial No.: 09/225,198 Group 2755

RECEIVED
MAY 2 0 1999
Group 2700

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						

Foreign Patent or Published Foreign Patent Application

Examiner Initial	No.	Document No.	Publication Date	Country or Patent Office	Class	Sub-class	Translation Yes	Translation No
	L							
	M							
	N							
	O							
	P							

Other Documents

Examiner Initial	No.	Author, Title, Date, Place (e.g. Journal) of Publication
fcb	R	MORAN, Douglas B. and CHEYER, Adam J., "Intelligent Agent-based User Interfaces", Article Intelligence center, SRI International
fcb	S	MARTIN, David L., CHEYER, Adam J. and MORAN, Douglas B., "Building Distributed Software Systems with the Open Agent Architecture"
fcb	T	COHEN, Philip R. and CHEYER, Adam, SRI International, WANG, Michelle, Stanford University, BAEG, Soon Cheol, ETRI, "An Open Agent Architecture"

Examiner	Date Considered
[signature] Jr	2/11/02

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.

O I P E
JC48
MAY 1 4 1999
PATENT & TRADEMARK OFFICE

Form 1449 (Modified) **Information Disclosure Statement By Applicant** (Use Several Sheets if Necessary)	Atty Docket No. SRI1P016 Applicant: Cheyer et al. Filing Date: January 5. 1999	Serial No.: 09/225,198 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	Translation No
	L							
	M							
	N							
	O							
	P							

Other Documents

Examiner Initial	No.	Author, Title, Date, Place (e.g. Journal) of Publication
JAB	R	JULIA, Luc E. and CHEYER, Adam J., SRI International "Cooperative Agents and Recognition Systems (CARS) for Drivers and Passengers",
JAB	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",
JAB	T	CHEYER, Adam and LULIA, Luc, SRI International "Multimodal Maps: An Agent-based Approach",

Examiner	Date Considered
[illegible] A. Bullock Jr	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.

Form 1449 (Modified) **Information Disclosure Statement By Applicant** (Use Several Sheets if Necessary)	Atty Docket No. SRI1P016 Applicant: Cheyer et al. Filing Date: January 5. 1999	Serial No.: 09/225,198 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						

Foreign Patent or Published Foreign Patent Application

Examiner Initial	No.	Document No.	Publication Date	Country or Patent Office	Class	Sub-class	Translation Yes	Translation No
	L							
	M							
	N							
	O							
	P							

Other Documents

Examiner Initial	No.	Author, Title, Date, Place (e.g. Journal) of Publication
JAB	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",
JAB	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
JAB	T	CHEYER, Adam, MARTIN, David and MORAN, Douglas, "The Open Agent architecture™", SRI International, AI Center

Examiner	Date Considered
Kevin A. Bullock Jr	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.

PE JC48 MAY 1 4 1999 PATENT & TRADEMARK OFFICE

Form 1449 (Modified) **Information Disclosure Statement By Applicant** (Use Several Sheets if Necessary)	Atty Docket No. SRI1P016 Applicant: Cheyer et al. Filing Date: January 5. 1999	Serial No.: 09/225,198 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 2 0 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	Translation No
	L							
	M							
	N							
	O							
	P							

Other Documents

Examiner Initial	No.	Author, Title, Date, Place (e.g. Journal) of Publication
JAB	R	Dejima, Inc., http://www.dejima.com/
JAB	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
JAB	T	MARTIN, David; OOHAMA, Hiroki; MORAN, Douglas; CHEYER, Adam; "Information Brokering in an Agent Architecture," Proceeding of the 2nd International Conference on Practical Application of Intelligent Agents & Multi-Agent Technology, London, April 1997

Examiner	Date Considered
Louis A Bullock Jr	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

Attorney Docket No. 59501-8016.US01

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

OIPE AUG 13 2002 JC109 PATENT & TRADEMARK OFFICE

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

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

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.

08/16/2002 CNGUYEN 00000007 502207 09225198
01 FC:126 180.00 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.

Respectfully submitted,
Perkins Coie LLP

Date: 6 Aug 2002

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

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.

Respectfully submitted,
Perkins Coie LLP

Brian R. Coleman
Registration No. 39,145

Date: 6 Aug 2002

Correspondence Address:

Customer No. 22918
Perkins Coie LLP
P.O. Box 2168
Menlo Park, California 94026
(650) 838-4300

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

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

Published

(60) **Parent Application or Grant**
UNITED VIDEO PROPERTIES, INC. [/]; (). ELLIS, Michael, D. [/]; (). LEMMONS, Thomas, R. [/]; (). THOMAS, William, L. [/]; (). TREYZ, G., Victor ; ().

(54) **Title: CLIENT-SERVER ELECTRONIC PROGRAM GUIDE**
(54) **Titre: GUIDE DE PROGRAMMES ELECTRONIQUE CLIENT-SERVEUR**

(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.

(57) Abrégé

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.

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : H04N 7/16	A1	(11) International Publication Number: WO 00/11869 (43) International Publication Date: 2 March 2000 (02.03.00)

(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).

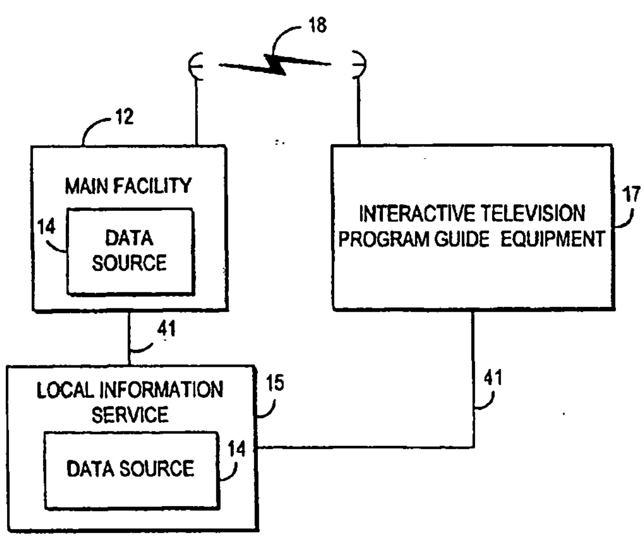
(74) Agents: TREYZ, G., Victor et al.; Fish & Neave, 1251 Avenue of the Americas, New York, NY 10020 (US).

(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).

Published

With international search report.

(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			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

Description

CLIENT-SERVER ELECTRONIC PROGRAM GUIDE

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 table. On-line program guides have been proposed that require users to navigate the Internet to access program listings.

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.

It is therefore an object of the present invention to provide an interactive televison 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.

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 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 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 data for additional services other than television program listings (e.g., weather information, associated Internet web links, computer software, etc.). The main facility (and other sources) may provide the program guide data to the television distribution facilities 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.

Each television distribution facility has a 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 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 servers may also store user data, such as user preference profiles, parental control settings, record and reminder settings, viewing history, and other suitable data.

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 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 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 requirements of the local distribution network.

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 and storage capabilities of the users' television 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 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 by the program guide server or program guide client 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 opportunity to access, modify, or delete the expressions.

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.

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.

Brief Description of the Drawings

FIG. 1 is a schematic block diagram of an illustrative system in accordance with the present invention.

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.

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.

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.

FIG. 5 is an illustrative main menu screen in accordance with the principles of the present invention.

FIG. 6 is an illustrative program listings by time screen in accordance with the principles of the present invention.

FIG. 7 is an illustrative program listings by channel screen in accordance with the principles of the present invention.

FIGS. 8a-8c are illustrative program listings by category screens in accordance with the principles of the present invention.

FIG. 9a is an illustrative boolean type criteria screen in accordance with the principles of the present invention.

FIG. 9b is an illustrative natural language criteria screen in accordance with the principles of the present invention.

FIG. 10 shows an illustrative agents screen in accordance with the principles of the present invention.

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.

FIG. 12 shows an illustrative setup screen in accordance with the principles of the present invention.

FIGS. 13a-13f show illustrative preference profile screens in accordance with the principles of the present invention.

FIG. 14 shows an illustrative profile activation screen in accordance with the principles of the present invention.

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.

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.

FIGS. 17a and 17b show illustrative criteria screens in accordance with the principles of the present invention.

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.

FIGS. 20a and 20b show an illustrative viewer recommendation overlay, in accordance with the principles of the present invention.

FIG. 20c shows an illustrative additional information screen in accordance with the principles of the present invention.

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.

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.

FIG. 23 is a flowchart of illustrative steps involved in processing and using expressions in accordance with the principles of the present invention.

FIG. 24 is a flowchart of illustrative steps involved in tracking and using viewing histories in accordance with the principles of the present invention.

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 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.

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.

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 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 FIG. 1 to avoid over-complicating the drawing.

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 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 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.

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 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 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