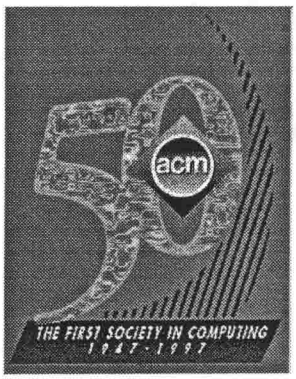
IUI 97

1997 International Conference on Intelligent User Interfaces



Editors

Johanna Moore, Papers & Program Chair
Ernest Edmonds, Panels
Angel Puerta, Papers & Debate

Sponsored by:
ACM SIGART - Special Interest Group on Artificial Intelligence
ACM SIGCHI - Special Interest Group on Computer-Human Interaction

The Association for Computing Machinery
1515 Broadway
New York, N.Y. 10036

ACM ISBN: 0-89791-839-8

Additional copies may be ordered prepaid from:

ACM Order Department
PO Box 12114
Church Street Station
New York NY 10257

ACM European Service Center
108 Cowley Road
Oxford OX 4 1JF UK
Phone: +44-1-865-382338
Fax: +44-1-865-3811338
E-mail: acm_europe@acm.org
URL: http://www.acm.org

Phone: 1-800-342-6626
(USA and Canada)
+1-212-944-1318
Fax: +1-21-944-1318
E-mail: acmpubs@acm.org

ACM Order Number: 608970

Printed in the U.S.A.

Multimodal User Interfaces in the Open Agent Architecture

Douglas B. Moran
Adam J. Cheyer
Luc E. Julia
David L. Martin
SRI International
333 Ravenswood Avenue
Menlo Park CA 94025 USA
+1 415 859 6486
{moran,cheyer,julia,martin}@ai.sri.com

Sangkyu Park
Artificial Intelligence Section
Electronics and Telecommunications
Research Institute (ETRI)
161 Kajong-Dong
Yusong-Gu, Taejon 305-350 KOREA
+82 42 860 5641
skpark@com.etri.re.kr

ABSTRACT

The design and development of the Open Agent Architecture (OAA)[1] system has focused on providing access to agent-based applications through an intelligent, cooperative, distributed, and multimodal agent-based user interfaces. The current multimodal interface supports a mix of spoken language, handwriting and gesture, and is adaptable to the user's preferences, resources and environment. Only the primary user interface agents need run on the local computer, thereby simplifying the task of using a range of applications from a variety of platforms, especially low-powered computers such as Personal Digital Assistants (PDAs). An important consideration in the design of the OAA was to facilitate mix-and-match: to facilitate the reuse of agents in new and unanticipated applications, and to support rapid prototyping by facilitating the replacement of agents by better versions.

The utility of the agents and tools developed as part of this ongoing research project has been demonstrated by their use as infrastructure in unrelated projects.

Keywords: agent architecture, multimodal, speech, gesture, handwriting, natural language

INTRODUCTION

A major component of our research on multiagent systems is in the user interface to large communities of agents. We have developed agent-based multimodal user interfaces using the same agent architecture used to build the back ends of these applications. We describe these interfaces and the larger architecture, and outline some of the applications that have been built using this architecture and interface agents.

IUI 97, Orlando Florida USA

OVERVIEW OF OPEN AGENT ARCHITECTURE

The Open Agent Architecture (OAA) is a multiagent system that focuses on supporting the creation of applications from agents that were *not* designed to work together, thereby facilitating the wider *reuse* of the expertise embodied by an agent. Part of this focus is the user interface to these applications, which can be viewed as supporting the access of human agents to the automated agents. Key attributes of the OAA are

- *Open*: The OAA supports agents written in multiple languages and on multiple platforms. Currently supported languages are C, Prolog, Lisp, Java, Microsoft's Visual Basic and Borland's Delphi. Currently supported platforms are PCs (Windows 3.1 and 95), Sun Workstations (Solaris 1.1 and 2.x) and SGIs.
- *Distributed*: The agents that compose an application can run on multiple platforms.
- *Extensible*: Agents can be added to the system while it is running, and their capabilities will become immediately available to the rest of the agents. Similarly, agents can be dynamically removed from the system (intentionally or not).
- *Mobile*: OAA-based applications can be run from a lightweight portable computer (or PDA) because only the user interface agents need run on the portable. They provide the user with access to a range of agents running on other platforms.
- *Collaborative*: The user interface is implemented with agents, and thus the user appears to be just another agent to the automated agents. This greatly simplies creating systems where multiple humans and automated agents cooperate.
- *Multiple Modalities*: The user interface supports handwriting, gesture and spoken language in addition to the traditional graphical user interface modalities.

- *Multimodal Interaction*: Users can enter commands with a mix of modalities, for example, a spoken command in which the object to be acted on is identified by a pen gesture (or other graphical pointing operation).

The OAA has been influenced by work being done as part of DARPA's I3 (Intelligent Integration of Information) program (http://isx.com/pub/I3) and Knowledge Sharing Effort (http://www-ksl.stanford.edu/knowledge-sharing/) [13].

THE USER INTERFACE

The User Interface Agent

The user interface is implemented with a set of agents that have at their logical center an agent called the *User Interface (UI) Agent*. The User Interface Agent manages the various modalities and applies additional interpretation to those inputs as needed. Our current system supports speech, handwriting and pen-based gestures in addition to the conventional keyboard and mouse inputs. When speech input is detected, the UI Agent sends a command to the Speech Recognition agent to process the audio input and to return the corresponding text. Three modes are supported for speech input: *open microphone*, *push-to-talk*, and *click-to-start-talking*. Spoken and handwritten inputs can be treated as either raw text, or interpreted by a natural language understanding agent.

There are two basic styles of user interface. The first style parallels the traditional graphical user interface (GUI) for an application: The user selects an application and is presented with a window that has been designed for the application implemented by that agent and that is composed of the familiar GUI-style items. In this style interface, the application is typically implemented as a primary agent, with which the user interacts, and a number of supporting agents that are used by the primary agent, and whose existence is hidden from the user. When text entry is needed, the user may use handwriting or speech instead of the keyboard, and the pen may be used as an alternative to the mouse. Because the UI Agent handles all the alternate modalities, the applications are isolated from the details of which modalities are being used. This simplifies the design of the applications, and simplifies adding new modalities.

In the second basic style of interface, not only is there no primary agent, the individual agents are largely invisible to the user, and the user's requests may involve the cooperative actions of multiple agents. In the systems we have implemented, this interface is based on natural language (for example, English), and is entered with either speech or handwriting. When the UI Agent detects speech or pen-based input, it invokes a speech recognition agent or handwriting recognition agent, and sends the text returned by that agent to a natural language understanding agent, which produces a *logical form* representation of the user's request. This logical form is then passed to a *Facilitator* agent, which identifies the subtasks and delegates them to the appropriate application agents. For example, in our *Map-based Tourist Information* application for the city of San Francisco, the user can ask for the distance between a hotel and sightseeing destination. The locations of the two places are in different databases, which are managed by different agents, and the distance calculation is performed by yet another agent.

These two basic styles of interfaces can be combined in a single interface. In our *Office Assistant* application, the user is presented with a user interface based on the Rooms metaphor and is able to access conventional applications such as e-mail, calendar, and databases in the familiar manner. In addition there is a subwindow for spoken or written natural language commands that can involve multiple agents.

A major focus of our research is multimodal inputs, typically a mix of gesture/pointing with spoken or handwritten language. The UI agent manages the interpretation of the individual modalities and passes the results to a *Modality Coordination* agent, which returns the composite query, which is then passed to the Facilitator agent for delegation to the appropriate application agents (described in subsequent sections).

Speech Recognition

We have used different speech recognition systems, substituting to meet different criteria. We use research systems developed by another laboratory in our organization (http://www-speech.sri.com/) [3] and by a commercial spin-off from that laboratory.[2] We are currently evaluating other speech recognizers, and will create agents to interface to their application programming interfaces (APIs) if they satisfy the requirements for new applications being considered.

Natural Language Understanding

A major advantage of using an agent-based architecture is that it provides simple mix-and-match for the components. In developing systems, we have used three different natural language (NL) systems: a simple one, based on Prolog DCG (Definite Clause Grammar), then an intermediate one, based on CHAT [16], and finally, our most capable research system GEMINI [6, 7]. The ability to trivially substitute one natural language agent for another has been very useful in rapid prototyping of systems. The DCG-based agent is used during the early stages of development because grammars are easily written and modified. Writing grammars for the more sophisticated NL agents requires more effort, but provides better coverage of the language that real users are likely to use, and hence we typically delay upgrading to the more sophisticated agents until the application crosses certain thresholds of maturity and usage.

[1] Open Agent Architecture and OAA are trademarks of SRI International. Other brand names and product names herein are trademarks and registered trademarks of their respective holders.

[2] Nuance Corporation (formerly Corona Corp.), Building 110, 333 Ravenswood Avenue, Menlo Park, CA 94025 (domain: coronacorp.com)

Pen Input

We have found that including a pen in the user interface has several significant advantages. First, the gestures that users employ with a pen-based system are substantially richer than those employed by other pointing and tracking systems (*e.g.*, a mouse). Second, handwriting is an important adjunct to spoken language. Speech recognizers (including humans) can have problems with unfamiliar words (*e.g.*, new names). Users can use the pen to correct misspelled words, or may even anticipate the problem and switch from speaking to handwriting. Third, our personal experience is that when a person who has been using a speech-and-gesture interface faces an environment where speech is inappropriate, replacing speech with handwriting is more natural.

Using 2D gestures in the human-computer interaction holds promise for recreating the pen-and-paper situation where the user is able to quickly express visual ideas while she or he is using another modality such as speech. However, to successfully attain a high level of human-computer cooperation, the interpretation of on-line data must be accurate and fast enough to give rapid and correct feedback to the user.

The gestures-recognition engine used in our application is fully described in [9] as the early recognition process. There is no constraint on the number of strokes. The latest evaluations gave better than 96% accuracy, and the recognition was performed in less than half a second on a PC 486/50, satisfying what we judge is required in terms of quality and speed.

In most applications, this engine shares pen data with a handwriting recognizer. The use of the same medium to handle two different modalities is a source of ambiguities that are solved by a competition between both recognizers in order to determine whether the user wrote (a sentence or a command) or produced a gesture. A remaining problem is to solve a mixed input (the user draws and writes in the same set of strokes).

The main strength of the gestures recognition engine is its adaptability and reusability. It allows the developer to easily define the set of gestures according to the application. Each gesture is actually described with a set of parameters such as the number of directions, a broken segment, and so forth. Adding a new gesture consists of finding the description for each parameter. If a conflict appears with an existing object, the discrimination is done by creating a new parameter. For a given application, as few as four parameters are typically required to describe and discriminate the set of gestures.

We can use any handwriting recognizer compatible with Microsoft's PenWindows.[3]

Modality Coordination Agent

Our interface supports a rich set of interactions between natural language (spoken, written, or typed) and gesturing (*e.g.*, pointing, circling)—much richer than that seen in the *put-that-there* systems. Deictic words (*e.g.*, *this, them, here*) can be used to refer to many classes of objects, and also can be used to refer to either individuals or collections of individuals.

The Modality Coordination (MC) agent is responsible for combining the inputs in the different modalities to produce a single meaning that matches the user's intention. It is responsible for resolving references, for filling in missing information for an incoming request, and for resolving ambiguities by using contexts, equivalence or redundancy.

Taking into account contexts implies establishing a hierarchy of rules between them. The importance of each context and the hierarchy may vary during a single session. In the actual system, missing information is extracted from the dialogue context (no graphical context or interaction context).

When the user says *"Show me the photo of this hotel"* and simultaneously points with the pen to a hotel, the MC agent resolves references based on that gesture. If no hotel is explicitly indicated, the MC agent searches the conversation context for an appropriate reference (for example, the hotel may have been selected by a gesture in the previous command). If there is no selected hotel in the current context, the MC Agent will wait a certain amount of time (currently 2 to 3 seconds) before asking the user to identify the hotel intended. This short delay is designed to accommodate different synchronizations of speech and gesture: different users (or a single user in different circumstances) may point before, during or just after speaking.

In another example, the user says *"Show me the distance from the hotel to here"* while pointing at a destination. The previous queries have resulted in a single hotel being focused upon, and the MC agent resolves *"the hotel"* from this context.[4] The gesture provides the MC agent with the referent of *"here"*. Processing the resulting query may involve multiple agents, for example, the location of hotels and sightseeing destinations may well be in a different databases, and these locations may be expressed in different formats, requiring another agent to resolve the differences and then compute the distance.

Flexible Sets of Modalities

The OAA allows the user maximum flexibility in what modalities will be used. Sometimes, the user will be on a computer that does not support the full range of modalities (*e.g.*, no pen or handwriting recognition). Sometimes, the user's environment limits the choice of modalities, for example, spoken commands are inappropriate in a meeting where someone else is speaking, whereas in a moving vehicle, speech is likely to be more reliable than handwriting. And sometimes, the user's choice of modalities is influenced by the data being entered [14].

With this flexibility, the telephone has become our low-end user interface to the system. For example, we can use the

[3] Our preferred recognizer is *Handwriter for Windows* from Communication Intelligence Corp (CIC) of Redwood City, CA.

[4] User feedback about which items are in focus (contextually) is provided by graphically highlighting them.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.