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.

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.

[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.

- *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.

[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.

telephone to check on our appointments, and we use the telephone to notify us of the arrival and content of important e-mail when we are away from our computers.

This flexibility has also proven quite advantageous in accommodating hardware failure. For example, moving the PC for one demonstration of the system shook loose a connection on the video card. The UI agent detected that no monitor was present, and used the text-to-speech agent to generate the output that was normally displayed graphically.

In another project's demonstration (CommandTalk), the designated computer was nonfunctional, and an underpowered computer had to be substituted. Using the OAA's innate capabilities, the application's components were distributed to other computers on the net. However, the application had been designed and tested using the microphone on the local computer, and the substitute had none. The solution was to add the Telephone agent that had been created for other applications: it automatically replaced the microphone as the input to the speech recognizer.

Learning the System

One of the well-known problems with systems that utilize natural language is in communicating to the user what can and cannot be said. A good solution to this is an open research problem. Our approach has been to use the design of the GUI to help illustrate what can be said: All the simple operations can also be invoked through traditional GUI items, such as menus, that cover much of the vocabulary.

OAA AGENTS

Overview

OAA agents communicate with each other in a high-level logical language called the Interagent Communication Language (ICL). ICL is similar in style and functionality to the Knowledge Query and Manipulation Language (KQML) of the DARPA Knowledge Sharing Effort. The differences are a result of our focus on the user interface: ICL was designed to be compatible with the output of our natural language understanding systems, thereby simplifying transforming a user's query or command into one that can be handled by the automated agents.

We have developed in initial set of tools (the Agent Development Toolkit) to assist in the creation of agents [11]. These tools guide the developer through the process, and automatically generate code templates from specifications (in the style of various commercial CASE tools). These tools are implemented as OAA agents, so they can interact with, and build upon, existing agents. The common agent support routines have been packaged as libraries, with coordinated libraries for the various languages that we support.[5]

These tools support building both entirely new agents and creating agents from existing applications, including legacy systems. These latter agents are called *wrappers* (or transducers); they convert between ICL and the application's API (or other interface if there is no API).

The Facilitator Agent

In the OAA framework, the *Facilitator* agents play a key role. When an agent is added to the application, it registers its capabilities with the Facilitator. Part of this registration is the natural language vocabulary that can be used to talk about the tasks that the agent can perform. When an agent needs work done by other agents within the application, it sends a request to the Facilitator, which then delegates it to an agent, or agents, that have registered that they can handle the needed tasks. The ability of the Facilitator to handle complex requests from agents is an important attribute of the OAA design. The goal is to minimize the information and assumptions that the developer must embed in an agent, thereby making it easier to reuse agents in disparate applications.

The OAA supports direct communication between application agents, but this has not been heavily utilized in our implementations because our focus has been on aspects of applications in which the role of the Facilitator is crucial. First, we are interested in user interfaces that support interactions with the broader community of agents, and the Facilitator is key to handling complex queries. The Facilitator (and supporting agents) handle the translation of the user's model of the task into the system model (analogous to how natural language interfaces to databases handle transforming the user's model into the database's schemas). Second, the Facilitator simplifies reusing agents in new applications. If a community of agents is assembled using agents acquired from other communities, those agents cannot be assumed to all make atomic requests that can be handled by other agents: simple requests in one application may be implemented by a combination of agents in another application. The Facilitator is responsible for decomposing complex requests and translating the terminology used. This translation is typically handled by delegating it to another agent.

In the OAA, the Facilitator is a potential bottleneck if there is a high volume of communication between the agents. Our focus has been on supporting a natural user interface to a very large community of intelligent agents, and this environment produces relatively low volume through the Facilitator. In the CommandTalk application (discussed later), the multiagent system is actually partitioned into two communities: the user interface and the simulator. The simulator has very high volume interaction and a carefully crafted communication channel and appears as a single agent to the Facilitator and the user interface agents.

Triggers

In an increasing variety of conventional applications, users can set *triggers* (also called monitors, daemons or watchdogs) to take specific action when an event occurs. However, the possible actions are limited to those provided in

[5] A release of a version of this software is planned. The announcement will appear on http://www.ai.sri.com/∼oaa/.

that application. The OAA supports triggers in which both the condition and action parts of a request can cover the full range of functionality represented by the agents dynamically connected to the network.

In a practical real-world example, one of the authors successfully used agent triggers to find a new home. The local rental housing market is very tight, with all desirable offerings being taken immediately. Thus, you need to be among the first to respond to a new listing. Several of the local newspapers provide on-line versions of their advertisements before the printed versions are available, but there is considerable variability in when they actually become accessible. To automatically check for suitable candidates, the author made the following request to the agent system: *"When a house for rent is available in Menlo Park for less than 1800 dollars, notify me immediately."* This natural language request installed a trigger on an agent knowledgeable about the domain of World Wide Web sources for house rental listings. At regular intervals, the agent instructs a Web retrieval agent to scan data from three on-line newspaper databases. When an advertisement meeting the specified criteria is detected, a request is sent to the Facilitator for a notify action to be delegated to the appropriate other agents.

The notify action involves a complex series of interactions between several agents, coordinated by the Notify and Facilitator agents. For example, if the user is in a meeting in a conference room, the Notify agent first determines his current location by checking his calendar (if no listing is found, the default location is his office, which is found from another database). The Notify agent then requests contact information for the conference room, and finds only a telephone number. Subsequent requests create a spoken version of the advertisement and retrieve the user's confirmation password. When all required information is collected, the Facilitator contacts the Telephone agent with a request to dial the telephone, ask for the user, confirm his identity with password (entered by TouchTone), and finally play the message. Other media, including FAX, e-mail and pager, can be considered by the Notify agent if agents for handling these services happen to be connected to the network.

DISTRIBUTED SYSTEMS

Multiple Platforms

The OAA applications that we have implemented run on a variety of platforms, and the exact location of individual agents is easily changed. We currently support PCs (Windows 3.1 and 95) and Sun and SGI workstations. Our primary user interface platform is the PC, partly because it currently offers better support for pen-based computing and partly because of our emphasis on providing user interfaces on lightweight computers (portable PCs and PDAs in near future). PCs also have the advantage of mass-market GUI-building packages such as Visual Basic and Delphi. A lesser version of the user interface has been implemented under X for UNIX workstations.

Even when the UI is on a PC, some of the agents in the UI package are running elsewhere. Our preferred speech recognizer requires a UNIX workstation, and our natural language agents and Modality Coordination agent have been written for UNIX systems.

Mobile Computing

We view mobile computing not only as people moving about with portable computers using wireless communication, but also people moving *between* computers. Today's user may have a workstation in his office, a personal computer at home, and a portable or PDA for meetings. In additional, when the user meets with management, colleagues and customers ("customers" in the broad sense of the people who require his services), their computers may be different platforms. From each of these environments, the user should be able to access his data and run his applications.

The OAA facilitates supporting multiple platforms because only the primary user interface agents need to be running on the local computer, thereby simplifying the problem of porting to new platforms and modality devices. Also, since only a minimal set of agents need to be run locally, lightweight computers (portables, PDA, and older systems) have the resources needed to be able to utilize heavyweight, resource-hungry applications.

COLLABORATION

One of the major advantages of having an agent-based interface to a multiagent application is that it greatly simplifies the interactions between the user and the application: application agents may interact with a human in the same way they interact with any other agent.

This advantage is readily seen when building collaborative systems. Perhaps the simplest form of collaboration is to allow users to share input and output to each other's applications. This form of cooperation is inherent in the design of the OAA: it facilitates the interoperation of software developed by distributed communities, especially disparate user communities (different platforms, different conventions).

We are currently integrating more sophisticated styles of collaboration into the OAA framework, using the synchronous collaborative technology [5] built by another group within our organization. In the resulting systems, humans can communicate with agents, agents can work with other automated agents, and humans can interact in realtime with other humans users.

APPLICATIONS AND REUSE

Two applications, the *Office Assistant* and *Map-based Tourist Information* have been the primary experimental environments for this research project. The agent architecture and the specific agents developed on this research project have proved to be so useful that they are being used by an expanding set of other projects within our organization. These other internal projects are helping us improve the documentation and packaging of our toolkits and libraries, and we are hoping to release a version in the near future.

Some of the projects adopting the OAA have been motivated by the availability of various agents, especially the user interface agents. Some projects have gone further and used the OAA to integrate the major software components being developed on those projects.

Office Assistant

The OAA has been used as the framework for a number of applications in several domain areas. In the first OAA-based system, a multifunctional "office assistant", fourteen autonomous agents provide information retrieval and communication services for a group of coworkers in a networked computing environment ([4]). This system makes use of a multimodal user interface running on a pen-enabled portable PC, and allows for the use of a telephone to give spoken commands to the system. Services are provided by agents running on UNIX workstations, many of which were created by providing agent wrappers for legacy applications.

In a typical scenario, agents with expertise in e-mail processing, text-to-speech translation, notification planning, calendar and database access, and telephone control cooperate to find a user and alert him or her of an important message. The office assistant system provides a compelling demonstration of how new services can arise from the synergistic combination of the capabilities of components that were originally intended to operate in isolation. In addition, as described earlier, it demonstrates the combination of two basic styles of user interaction — one that directly involves a particular agent as the primary point of contact, and one that anonymously delegates requests across a collection of agents — in a way that allows the user to switch freely between the two.

In the interface for this system, the initial screen portrays an office, in which familiar objects are associated with the appropriate functionality, as provided by some agent. For instance, clicking on a wall clock brings up a dialogue that allows one to interact with the calendar agent (that is, browsing and editing one's appointments). In this style of interaction, even though the calendar agent may call on other agents in responding to some request, it has primary responsibility, in that all requests through that dialogue are handled by it.

The alternative style of interaction is one in which the user might speak *"Where will I be at 2:00 this afternoon?"*. In this case, the delegation of the request to the appropriate agents — which is done by the User Interface agent in concert with a Facilitator agent — reflects a style that is less direct and more anonymous.

Map-based Tourist Information

In a number of domains, access to information can very naturally be organized around a map-based interface. In creating such interfaces for several different systems, we have found the agent-based approach to multimodality to be extremely useful. In these systems, all the components share a common interface—the map—and the fact that there are many agents is entirely invisible to the user.

One example is a map-based system to provide tourist information about San Francisco. Requests expressed in a variety of modalities can control the scrolling and zoom level of the map, retrieve information about locations and distances, display hotels or attractions meeting a user's preferences, or present detailed information in a variety of media about particular hotels or attractions. Where appropriate, this information is derived and updated regularly from WWW sources.

Map-based interfaces provide a rich setting in which to explore the coordination of gesture with speech and traditional GUI modalities. The tourist information system accommodates the use of a variety of familiar pen gestures, such as circling objects or regions, drawing arrows, **X**'ing positions or objects, and striking out objects. Depending on context and timing considerations, requests can be derived from single gestures, multiple gestures interpreted together, spoken or handwritten input, point-and-click, or some combination of these operations.

For example, an arrow drawn across a map from right to left (which itself is recognized from two or three pen strokes) is interpreted as a request to scroll the map. The same effect may be achieved by speaking *"scroll left"*. Display of hotels can be obtained by writing or speaking *"Show hotels"*, or, perhaps, *"Show hotels with a pool"*. The distance between two objects or locations may be obtained by circling, **X**'ing, or clicking on each of them, and then drawing a straight line between them. Alternatively, one can speak *"Show the distance from here to here"*, while selecting two locations, or one can write *"distance"* either before or after selecting two objects.

This system, and the organization of the input recognition agents, is described in detail in [2]. A related system is described in [15].

CommandTalk

CommandTalk, a system in quite a different domain than tourism, was able to make use of the same approach to the map-based integration of speech with other modalities.[6] In the CommandTalk system, currently installed at the Marine Corps Air Ground Combat Center at Twentynine Palms, CA, a collection of OAA-enabled agents provides a spoken-English interface to a map-based simulation of armed forces [12]. CommandTalk has proven useful in providing realism to scenarios used in training military commanders. The simulator is roughly 500,000 lines of code that was provided to the interface developers. Within 2 weeks of receiving the simulator code, they were able to demonstrate a spoken language interface to the basic functionality of the package by creating an agent interface to that portion of the simulator's

[6] In the case of CommandTalk, gesture has not yet been a factor, but there has been an emphasis on the comprehensive use of speech, in combination with traditional GUI modalities.

functionality and then adapting the existing user interface agents to that domain. After the early prototype had demonstrated the utility of the concept, a more extensive analysis was conducted of the task and the commands used, and more capable prototypes were developed. One of the significant enhancements was the replacement of our simplest natural language agent (DCG-based) with our most sophisticated (based on GEMINI [6, 7]).

Summarization of Conversation

A system that summarizes conversations provided a novel opportunity to use two instances of a speech recognition agent, in conjunction with a single instance of a text processing agent ([10]). In this system, MIMI, two Japanese speakers engage in a conversation, such as, for example, an inquiry about room availability at a hotel. Each speaker is on a separate microphone, and each microphone feeds into a separate speech recognition agent. The output streams of these agents are both fed into a text processing agent, adapted especially for this task. Following the completion of the conversation, the text processing agent is able to print out a summary of what was discussed and agreed upon.

In constructing this system, as with CommandTalk, the ability to reuse and reconfigure preexisting user interface agents, in conjunction with newly created agents, afforded a significant savings in system construction time. The English-language speech recognizer was replaced with a Japanese-language version, and the natural language understanding agent that generated commands to the rest of the system was replaced by an agent that analyzed and stored the summary of the conversation.

Air Travel Information System

Web-based interfaces can readily be integrated into an agent-based system. At the same time that the agent system benefits from the universal accessibility of a Web interface, the HTML paradigm is extended and strengthened by the use of persistent interface agents to maintain the state of a sequence of interactions.

In one such system, user interface agents have been used to provide a Web/telephone interface to a spoken language Air Travel Information System (ATIS) [1]. In addition to speech recognition and natural language understanding agents, this system involves a telephone control agent, a response generation agent, and a User Interface agent. The initial version was based on HTML. The current version uses Java to provide more incremental feedback to the user.[7]

Multi-robot Control

SRI's family of mobile robots have been integrated as agents within the OAA framework. As such, robots may access, and be accessed by, existing OAA services, including corporate databases, text-to-speech generation, and telephone interfaces. In the Robot Competition at the 1996 AAAI conference, OAA's capabilities were used by the SRI team to coordinate the activities of three robots. SRI won the Office Navigation task, completing it much faster than any of the other competitors (who were using only single robots) [8].

The multimodal map application was minorly modified to provide monitoring and control of the robots as they navigate a building. The screen displays a blueprint-style map of the area in which the robots operate, and the positions of the movable objects (robots and the objects that they can manipulate) are updated in realtime. Although the input modalities are the same as the earlier application (Map-based Tourist Information), there are noticeable differences. First, the inputs are predominantly commands, instead of information retrieval (queries), Second, some pen gestures mean different things: for example, with the robots, an arrow is used to indicate orientation (*"Robot one, face this direction"*) or direction (*"Move this way"*).

Emergency Response System

Another system for which a map-based interface has been useful is a prototype system of pen-based mobile computing units for use in the field by teams responding to a disaster such as an earthquake. In this system, a database of maps is available on each mobile unit (to avoid having to download sizable bitmaps), but information about specific locations and structures is stored in a centralized set of databases. This information can be retrieved and/or updated as appropriate by each mobile unit. The centralized database server also receives updates from hospitals and clinics as to their status, capacity, and patients being treated.

For example, as a response team learns the condition of the streets and structures in its region, it is able to record this information on the map-based interface, using point-and-click in combination with handwriting or typing, and then upload the data to the central databases. When a street or structure is found to be unsafe, that information can be relayed to all mobile units.

In the case in which an injured person is found, the system allows for the entry of some basic facts about the injury. Following that, an agent operating on the central server makes a determination of what hospital or clinic would be most appropriate for the person, based on current status reports, and this recommendation is then returned to the response team.

This system has both Japanese-language and English-language interfaces.

CONCLUSIONS

The OAA has proven to be useful in constructing sophisticated systems because it provides the flexibility to combine applications that were not originally envisioned as a package. The OAA differs from much of the other research on distributed agents in its focus on providing multimodal user interfaces to systems assembled from disparate agents. This focus results in a tradeoff which is a major limitation of this

[7] It may be accessed at http://www-speech.sri.com/demos/atis.html

architecture: while the Facilitator agent is key to cooperation between independently developed agents, it is a potential bottleneck in systems where agents need high-volume, low-delay interactions (discussed in *The Facilitator Agent*). In one existing application (and one under consideration), a composite approach has provided a viable solution for this limitation.

ACKNOWLEDGMENTS

This paper is based on work that was supported in part by a contract to SRI from the Electronics and Telecommunications Research Institute (Korea). Philip R. Cohen (now at the Oregon Graduate Institute) was project leader until August 1994, and was responsible for many of the design decisions in the systems described here. Any opinions expressed in this paper are strictly those of the authors.

REFERENCES

1 Harry Bratt, John Dowding, and Kate Hunicke-Smith. The SRI telephone-based ATIS system. In *Proc. of the ARPA Spoken Language System Technology Workshop*, Austin, Texas, January 1995. Also http://www.ai.sri.com/natural-language/projects/arpa-sls/apps.html.

2 Adam Cheyer and Luc Julia. Multimodal maps: An agent-based approach. In *Proc. of the International Conference on Cooperative Multimodal Communication (CMC/95)*, Eindhoven, The Netherlands, May 1995. Also http://www.ai.sri.com/∼oaa/ + "Bibliography".

3 Michael Cohen, Hy Murveit, Jared Bernstein, Patti Price, and Mitchel Weintraub. The DECIPHER speech recognition system. In *IEEE ICASSP*, pages 77–80, 1990.

4 Philip R. Cohen, Adam J. Cheyer, Michelle Wang, and Soon Cheol Baeg. An open agent architecture. In O. Etzioni, editor, *Proc. of the AAAI Spring Symposium Series on Software Agents*, pages 1–8, Stanford, California, March 1994. American Association for Artificial Intelligence.

5 Earl Craighill, Martin Fong, Keith Skinner, Ruth Lang, and Kathryn Gruenefeldt. SCOOT: An object-oriented toolkit for multimedia collaboration. In *Proc. of the ACM MULTIMEDIA 94 Conference*, pages 41–49, San Francisco, CA, October 1994. Also http://www.std.sri.com/public/ftp/ACE/Papers/-SCOOT94.ps.Z.

6 John Dowding, J. Mark Gawron, Douglas Appelt, John Bear, Lynn Cherny, Robert Moore, and Douglas Moran. GEMINI: A natural language system for spoken-language understanding. In *Proc. of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 54–61, Ohio State University, Columbus, Ohio, 22–26 June 1993.

7 John Dowding, Robert Moore, Francois Andry, and Doug las Moran. Interleaving syntax and semantics in an efficient bottom-up parser. In *Proc. of the 32nd Annual Meeting of the Association for Computational Linguistics*, pages 110–116, New Mexico State University, Las Cruces, New Mexico, 27 June – 1 July 1994.

8 Didier Guzzoni, Adam Cheyer, Luc Julia, and Kurt Konolige. Report on the SRI Pioneer Robot Team at AAAI-96 Robotics Competition and Exhibition (tentative title). To appear in *AI Magazine*, Winter 1996 or Spring 1997.

9 Luc Julia and Claudie Faure. Pattern recognition and beautification for a pen based interface. In *ICDAR'95*, pages 58–63, Montreal, Canada, 1995.

10 Megumi Kameyama, Goh Kawai, and Isao Arima. A real-time system for summarizing human-human spontaneous spoken dialogues. In *Proc. of the Fourth International Conference on Spoken Language Processing (ICSLP-96)*, October 1996. Also http://www.ai.sri.com/∼megumi/ + "My publications".

11 David L. Martin, Adam J. Cheyer, and Gowang-Lo Lee. Agent development tools for the Open Agent Architecture. In *Proc. of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, pages 387–404, London, April 1996. The Practical Application Company Ltd.

12 Robert Moore, John Dowding, Harry Bratt, J.M̃ark Gawron, Yonael Gorfu, and Adam Cheyer. Commandtalk: A spoken-language interface for battlefield simulation. Technical report, Artificial Intelligence Center, SRI International, 21 June 1996. Also http://www.ai.sri.com/natural-language/projects/arpa-sls/apps.html.

13 R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3), 1991.

14 S. L. Oviatt. Pen/voice: Complementary multimodal communication. In *Proc. of Speech Tech'92*, pages 238–241, 1992.

15 S. L. Oviatt. Multimodal interfaces for dynamic interactive maps. In *Proc. of CHI 96*, Vancouver, Canada, 1996. Assoc. for Computing Machinery.

16 F. C. N. Pereira. *Logic for Natural Language Analysis*. PhD thesis, U. of Edinburgh, 1983.