An Open Agent Architecture*

Philip R. Cohen
Adam Cheyer
SRI International
(pcohen@ai.sri.com)
Michelle Wang
Stanford University
Soon Cheol Baeg
ETRI

ABSTRACT

The goal of this ongoing project is to develop an open agent architecture and accompanying user interface for networked desktop and handheld machines. The system we are building should support distributed execution of a user's requests, interoperability of multiple application subsystems, addition of new agents, and incorporation of existing applications. It should also be transparent; users should not need to know where their requests are being executed, nor how. Finally, in order to facilitate the user's delegating tasks to agents, the architecture will be served by a multimodal interface, including pen, voice, and direct manipulation. Design considerations taken to support this functionality will be discussed below.

INTRODUCTION

Agents are all the rage. "Visioneering" videos, such as Apple Computer's Knowledge Navigator, have helped to popularize the notion that programs endowed with agency, if not intelligence, are just around the corner. Soon, users need not themselves wade into the vast swamp of data in search of information, but rather the desired, or better yet, needed information will be presented to the user by an intelligent agent in the most comprehensible form, at just the right time.

Although such rosy scenarios are easy to come by, intelligent agents are considerably more difficult to obtain. Still, substantial progress is being made on a variety of aspects of the agent story. At least three general conceptions of agent-based software systems can be found in current thinking:

1. Agents are programs sent out over the network to be executed on a remote machine.
2. Agents are programs on a given machine that offer services to others.
3. Agents are programs that assist the user in performing a task.

Each of these models can be found to some extent in present-day software products, for example, in (1) General Magic's emerging TELESCRIPT interpreter, (2) Microsoft's OLE 2.0 and (3) Apple Computer's Newton and Hewlett Packard's New Wave desktop, respectively. Given this space of conceptualizations, we need to be specific about ours.

Definitions and Objectives

Listed below are characteristics of what we are terming agents followed by an example of those characteristics as found in our system:

- *Delegation* — e.g., the ability to receive a task to be performed without the user's having to state all the details
- *Data-directed Execution* — e.g., the ability to monitor local or remote events, such as database updates, OS, or network activities, determining for itself the appropriate time to execute.
- *Communication* — e.g., the ability to enlist other agents (including people) in order to accomplish a task.
- *Reasoning* — e.g., the ability to prove whether its invocation condition is true, and to determine what are its arguments.
- *Planning* — e.g., the ability to determine which agent capabilities can be combined in order to achieve a goal.

Our initial prototype includes agents that exhibit aspects of all the above capabilities, except planning (but see [7]). Our goal is to develop an open agent architecture for networked desktop and handheld machines. The system we are building should support distributed execution of a user's requests, interoperability of multiple application subsystems, addition of new agents, and incorporation of existing applications. Finally, it should be transparent; users should not need to know where their requests are being executed, nor how.

*This paper was supported by a contract from the Electronics and Telecommunications Research Institute (Korea). Our thanks are also extended to AT&T for use of their text-to-speech system.

المنارة للاستشارات
www.manaraa.com

AGENT ARCHITECTURE

Based loosely on Schwartz's FLiPSiDE system [17], the Open Agent Architecture is a blackboard-based framework allowing individual software "client" agents to communicate by means of goals posted on a blackboard controlled by a "Server" process.

The Server is responsible both for storing data that is global to the agents, for identifying agents that can achieve various goals, and for scheduling and maintaining the flow of communication during distributed computation. All communication between client agents must pass through the blackboard. An extension of Prolog has been chosen as the interagent communication language (ICL) to take advantage of unification and backtracking when posting queries. The primary job of the Server is to decompose ICL expressions and route them to agents who have indicated a capability in resolving them. Thus, agents can communicate in an undirected fashion, with the blackboard acting as a broker. Communication can also take place also in a directed mode if the originating agent specifies the identity of a target agent.

An agent consists of a Prolog meta-layer above a knowledge layer written in Prolog, C or Lisp. The knowledge layer, in turn, may lie on top of existing standalone applications (e.g. mailers, calendar programs, databases). The knowledge layer can access the functionality of the underlying application through the manipulation of files (e.g., mail spool, calendar datafiles), through calls to an application's API interface (e.g. MAPI in Microsoft Windows), through a scripting language, or through interpretation of an operating system's message events (Apple Events or Microsoft Windows Messages).

Individual agents can respond to requests for information, perform actions for the user or for another agent, and can install triggers to monitor whether a condition is satisfied. Triggers may make reference to blackboard messages (e.g. when a remote computation is completed), blackboard data, or agent-specific test conditions (e.g. "when mail arrives...").

The creation of new agents is facilitated by a client library furnishing common functionality to all agents. This library provides methods for defining an agent's capabilities (used by the blackboard to determine when this agent should participate in the solving of a subgoal), natural language vocabulary (used by the interface agent), and polling status. It also provides functionality allowing an agent to read and write information to the blackboard, to receive requests for information or action, and to post such requests to the blackboard, a specific agent, or an entire population of appropriate agents.

When attempting to solve a goal, an agent may find itself lacking certain necessary information. The agent can either post a request of a specific agent for the information, or it may post a general request on the blackboard. In the latter case, all agents who can contribute to the search will send solutions to the blackboard for routing to the originator of the request. The agent initiating the search may choose either to wait until all answers return before continuing processing, or may set a trigger indicating that when the remote computation is finished, a notification should interrupt local work in progress. An agent also has access to primitives permitting distributed AND and OR-parallel solving of a list of goals.

Distributed Blackboard Architecture

As discussed above, the Open Agent Architecture contains one blackboard "server" process, and many client agents; client agents are permitted to execute on different host machines. We are investigating an architecture in which a server may itself be a client in a hierarchy of servers; if none of its client agents can solve a particular goal, this goal may be passed further along in the hierarchy. Following Gelerntner's LINDA model [8], blackboard systems themselves can be structured in a hierarchy, which could be distributed over a network (see Figure 1).[1]

When a goal (G) is requested to be posted on a local blackboard (BB1), and the blackboard server agent at BB1 determines that none of its child agents has the requisite capabilities to achieve the goal, it propagates the goal to a more senior blackboard server agent (BB4) in the hierarchy. BB4 maintains a knowledge base of the predicates that its lower level blackboards can evaluate. When a senior server receives such a request, it in turn will propagate the request down to its subsidiary servers. These subsidiary servers either have immediate client agents who can evaluate the goal, or can themselves pass on the goal to another subsidiary server. In the case illustrated in Figure 1, BB4 determines that none of its subsidiary blackboards can handle the goal, and thus sends the goal to its superior agent (BB5). BB5 passes the goal to BB6, who in turn passes it to BB9. When such a referred goal is passed through the hierarchy of blackboards, it is accompanied by information about the originating blackboard (indicated by the BB1 subscript on G), including information identifying its input port, host machine, etc. This continuation information will enable a return communication (with answers or failure) to be routed to the originating blackboard. Also, the identity of the responding knowledge source BB9 can be sent back to the originator, so that future queries of the same type from BB1 may be addressed directly to BB9 without passing through the hierarchy of blackboards.

Operational Agents

A variety of agents have been integrated into the Open Agent Architecture:

[1]This is referred to as a "federation architecture" in [9].

المنارة للاستشارات
www.manaraa.com

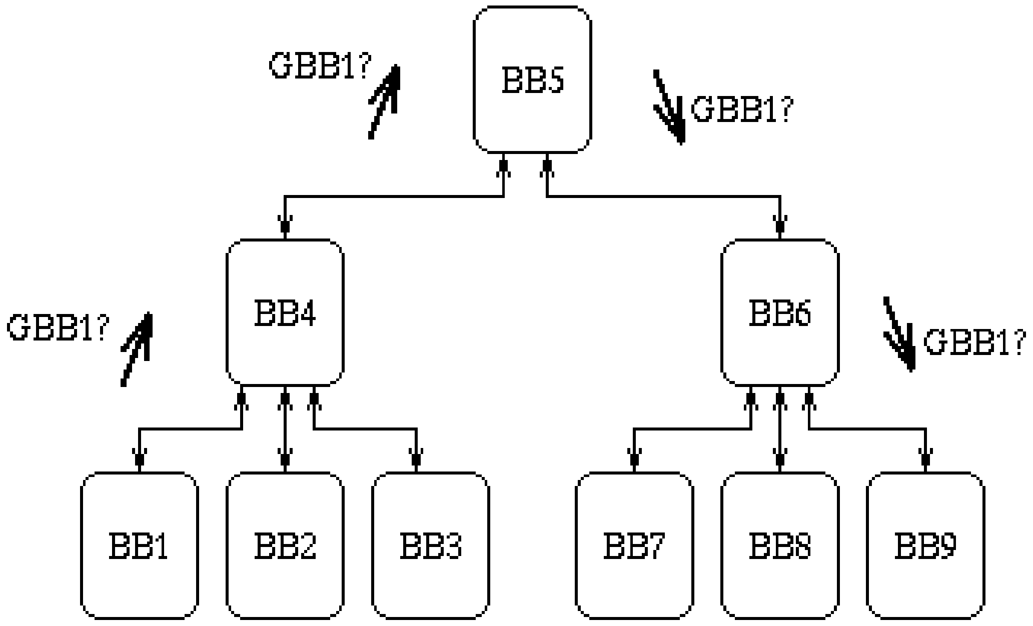

Figure 1: Hierarchy of Blackboard Servers

- a *User-interface* agent that accepts spoken or typed (and soon, handwritten) natural language queries from the user and presents responses to the queries.
- a *Database* agent, written in C, that interacts with a remote X.500 Directory System Agent database containing directory information.
- a *Calendar* agent, which can report upon where a person might be, or when they might be performing a particular action. This information is retrieved from data created by Sun Microsystem's CalenTool application.
- a *Mail* agent that can monitor incoming electronic messages, and forward or file them appropriately. The mail agent works with any Unix-compatible mail application (e.g. Sun's MailTool).
- a *News* agent that scans Internet newsgroups searching for specified topics or articles.
- a *Telephone* agent, that can dial a telephone using a ComputerPhone controller, and can communicate with users in English, using NewTTS, AT&T's text-to-speech system.

Communication Language

The key to a functioning agent architecture is the interagent communication language. We explain ours in terms of its form and content. Regarding the former, three speech act types are currently supported: `Solve` (i.e., a question), `Do` (a request) and `Post` (an assertion to the blackboard). For the time being, we have adopted little of the sophisticated semantics known to underlie such speech acts [5, 18, 19]. However, in attempting to protect an agent's internal state from being overwritten by uninvited information, we do not allow one agent to change another's internal state directly — only an agent that chooses to accept a speech act can do so. For example, a fact posted to the blackboard does not necessarily get placed in the database agent's files unless it so chooses, by placing a trigger on the blackboard asking to be notified of certain changes in certain predicates (analogous to Apple Computer's Publish and Subscribe protocol).

Although our interagent communication language is still evolving, we have adopted Horn clauses as the basic predicates that serve as arguments to the speech act types. However, for reasons discussed below, we have augmented the language beyond ordinary Prolog to include temporal information.

Because delegated tasks and rules will be executed at distant times and places, users may not be able simply to use direct manipulation techniques to select the items of interest, as those items may not yet exist, or their identities may be unknown. Rather, users will need to be able to *describe* arguments and invocation conditions, preferably in a natural language. Because these expressions will characterize events and their relationships, we expect natural language tense and aspect to be heavily employed [6]. Consequently, the meaning representation (or "logical form") produced by the multimodal interface will need to incorporate temporal

www.manaraa.com

information, which we do by extending a Horn clause representation with time-indexed predicates and temporal constraints. The blackboard server will need to decompose these expressions, distribute pieces to the various relevant agents, and engage in temporal reasoning to determine if the appropriate constraints are satisfied.

With regard to the content of the language, we need to specify the language of predicates that will be shared among the agents. For example, if one agent needs to know the location of the user, it will post an expression, such as `solve(location(user,U))`, that another agent knows how to evaluate. Here, agreement among agents would be needed that the predicate name is `location`, and its arguments are a person and a location. The language of nonlogical predicates need not be fixed in advance, it need only be common. Achieving such commonality across developers and applications is among the goals of the ARPA "Knowledge Sharing Initiative," [13] and a similar effort is underway by the "Object Management Group" (OMG) CORBA initiative to determine a common set of objects.

A difficult question is how the user interface can know about the English vocabulary of the various agents. When agents enter the system, they not only register their functional capabilities with the blackboard, they also post their natural language vocabulary to the the blackboard, where it can be read by the user interface. Although conceptually reasonable for local servers (and somewhat problematic for remote servers) the merging of vocabulary and knowledge is a difficult problem. In the last section, we comment on how we anticipate building agents to enforce communication and knowledge representation standards.

Example Scenario

The following is an example of an operational demonstration scenario that illustrates inter-agent communication (see Figure 2).

The user tells the interface agent (in spoken language) that "When mail arrives for me about a security break, get it to me". The interface agent translates this statement into a logical expression, and posts the expression to the blackboard. The blackboard server determines that a trigger should be installed on the mail agent, causing it to poll the user's mail database. Once the mail agent has determined that a message matching the requested topic has arrived for the user, it posts a query to find out the user's current location. The calendar agent responds, noting that the user is supposed to be in a meeting which is being held in a particular room; the database agent is then queried for the phone number of the room. Finally, the telephone agent is instructed to call the number, ask for the user (using voice synthesis), perform an identification verification by requesting a touchtone password, and then read the message to the user. We intend to add agents that would increase the number of ways in which a user might be contacted: agents to control fax machines, automatic pagers, and a notify agent that uses planning to determine which communication method is most appropriate in a given situation.

Comparison with Other Agent Architectures

The most similar agent architectures are FLiPSiDE [17] and that of Genesereth and Singh [9]. Like FLiPSiDE (Framework for Logic Programming Systems with Distributed Execution), our Open Agent Architecture uses Prolog as the interagent communication language, and introduces a uniform meta-layer between the blackboard Server and the individual agents. Some aspects of FLiPSiDE's blackboard architecture are more complex than in our system. It uses a multi-level locking scheme to try to reduce deadlock and minimize conflicts in blackboard access during moments of high concurrency. The system also uses separate knowledge sources for controlling triggers, ranking priorities and scheduling the executing of knowledge sources, whereas we incorporate these sorts of actions directly into the blackboard server. Some features important to our system that are not addressed by FLiPSiDE are the ability to handle temporal contraints over variables, and the possibility for an agent to explicitly request AND and OR-parallel solvingof a list of distributed goals.

Genesereth and Singh's architecture is more ambitious than ours in its employing a full first-order logic as the interagent communication language. As yet, we have not needed to expand our language beyond Horn clauses with temporal constraints, but this step may well be necessary. Genesereth and Singh use KIF (Knowledge Interchange Format) [13] as their basic language of predicates and as a knowledge integration strategy. Because of our user interface considerations, which in turn are heavily influenced by the form-factor constraints of future handheld devices, we will need to be able to merge contributions by different agents of their natural language vocabulary, related pronunciations, and semantic mappings of those vocabulary items to underlying predicates.

MAIL MANAGEMENT

In our earlier scenario, the mail agent was rather limited. To test our user interface and agent architecture more fully, we are creating a more substantive mail management agent, MAILTALK.

It has become common to develop mail managers that manipulate messages as they arrive according to a set of user-specified rules. The virtue of such systems is that users can make mail management decisions once, rather than consider each message in turn. However, a number of problems exist for such systems, as well as for all agent systems that we know of, especially when considered as tools for the general population.

www.manaraa.com

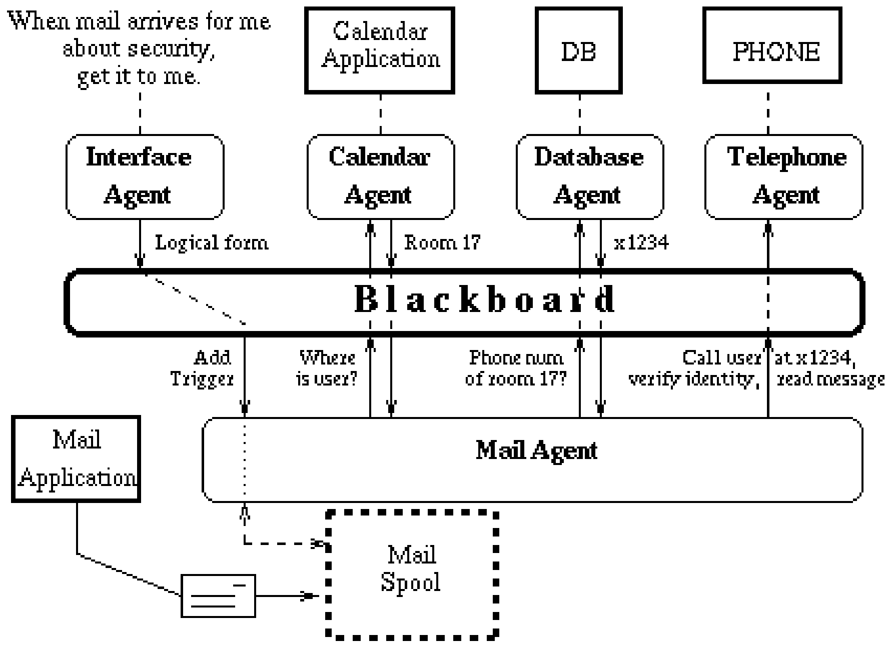

Figure 2: Example of agent interaction

- End users cannot easily specify the rules. In a number of current systems, a scripting language needs to be employed [1, 20], and in one system, users were required to write rules in a temporal query language [10]. We believe such methods for rule creation effectively eliminates the class of nontechnical users. Other systems employ templates that the user fills out [12]. Although this technique may work in many cases, it limits the power of the rules that users can create because they must search for an icon at which to point in order to specify the contents of a slot. Otherwise, they need to know or select the special syntax or concept name required. However, the selection of items from long menus is infeasible for handheld devices with little screen territory.
- End users cannot determine in advance how the *collection* of rules will behave once a new rule is added. This lack of predictability and the lack of debugging tools will undermine the utility of agent-based systems, especially in a networked environment.
- End users cannot easily determine what happened. Generally, little or no history of the database of events and rule firings is kept, and few tools are provided for reviewing that history.[2]
- The mail manager is a special purpose system, interacting loosely, if at all, with other components. Without tighter integration, the architecture and user interface for dealing with mail rules may diverge from what is offered for other agents.

[2]An exception to this is the use of "Mission Status Reports" in the Envoy agent framework [15].

Our prototype MAILTALK was built to address these concerns.

Rule specification. Based on technology developed for the SHOPTALK factory simulation system [2, 3, 4], MAILTALK permits users to specify rules by describing complex invocation conditions, and arguments with a multimodal interface featuring typed and spoken natural language, combined with direct manipulation. For example, the user can delegate to the mail agent as follows: "When Jones replies to my message about 'acl tutorials', send his reply to the members of my group." Here, Jones's reply cannot be selected or pointed at since it does not yet exist. The English parser produces expressions in the temporal logic, which are evaluated against various databases (e.g., the mail database, or a simulation database).

Predicting behavior. By giving end users the power to write their own rules means we have given them the freedom to make their own mistakes. Before letting a potentially erroneous collection of agents loose on one's mail (or, more generally, the network), we encourage users to *simulate* the behavior of those agents. Included with MAILTALK is a knowledge-based simulation environment that allows users to create hypothetical worlds, and permits them to send test messages or re-examine old mail files. In re-

www.manaraa.com

`When:`	A message from someone in the AIC has been read
`ARCHIVE`	
`Which message(s):`	it
`In which file:`	<AIC-Mail>

Figure 3: Creating a mail rule

sponse, the system fires the relevant rules, and updates a simulation database with the events that have happened. This database can extend the actual mail file, permitting expressions that depend on the entire database to be evaluated (e.g., "when more than 5 messages from cohen are in < point to icon for mail file>, move them to <icon for 'unimportant mail'>).

Reviewing History. In order to determine if the resulting behavior was in fact desired, users can ask questions about the results of the simulation, can view the simulation graphically, and can rewind the history to interesting times (e.g., when a message was read, or when a message was forwarded to a member of a given mail group). When satisfied with the resulting behavior of the collection of rules, users can install them in the real world to monitor the real mail file. Moreover, users can ask questions about the real mail database, such as "Who has replied to my message of November 26 about budgets?"

Example

The following is an example of the kind of processing found in MAILTALK. First, the user determines that she wants to test out a mail management rule before installing it. She creates a new "hypothetical world," and proceeds to create a rule by selecting the `Archive` action from a menu. This results in a template's being presented, which she fills out as shown in Figure 3.

The user enters an English expression as the invocation condition, points at the icon for a file (`AIC-Mail`), and deposits it into the destination field.[3] This rule definition is parsed into a Prolog representation, augmented with temporal information and constraints.

The user then proceeds to *digest* an old mail file, which simulates the sending of the old messages, updating the simulated mail database. The animated simulation indicates that the rule has been fired, but just to be certain that the appropriate messages were put into the desired file, the user asks "When did I read a message from someone in the AIC?", followed by "Where are those messages now?" When satisfied, she transfers this rule to the real world, and requests that incoming mail be monitored.

[3] For a discussion of the usability advantages of such templates over simply entering the above in one sentence, please see [3, 14].

It should be noted that the reading of a message creates an event that triggers a rule. In general, that verb (i.e., 'read') could be one that results from an agent's action (e.g., forwarding), and thus a cascade of rule activations would ensue. It is to ensure that users understand such complexities that we offer the simulation facility.

Comparison with Other Mail Managers

Numerous mail managers exist, and space precludes a comprehensive survey. Only the more comparable ones will be discussed below.

The mail management system most similar to our is ISCREEN [16]. It allows a keyword and forms-based creation of rules, and offers a simple simulation capability in which a user can pose test messages. In response, the system applies its rules and explains in English what it would have done. Because mail is filtered using a boolean combination of keywords in various fields, ISCREEN can detect that various rules will conflict, and can ask the user for a prioritization. The user can employ organizational expressions (e.g., "manager"), which the system resolves based on a Prolog-based Corporate Directory database. Our use of the X.500 Directory System Agent offers the same capability based on an emerging international standard.

The TAPESTRY mail system [10] incorporates a mail database (as opposed to just a mail file), that is queried by a temporal query language. MAILTALK share this basic underlying model, but rather than have users write temporal queries, the user interface creates the temporal logic expressions through English language descriptions, which are then evaluated over the mail database.

The INFORMATION LENS system [12] provides various message types, which can enter into filtering rules (e.g., when a message of type `Weekly Sales Report` arrives, forward it to ...), or can become arguments for other actions (e.g., opening a spreadsheet). This approach takes the first step to integrating mail with other agent-like behavior, but a more fuller integration is possible once it is realized that rule-based mail management is analogous to database monitoring (as shown in TAPESTRY), and that a more general agent architecture can subsume mail management as a special case. It is this latter approach that we are following by embedding the mail manager as an agent in the architecture.

IMPLEMENTATION

An initial implementation of each of the pieces described above has been developed (in Prolog and C) on a Unix platform, with the exception of the pen/voice interface, which is being implemented now. Communication is based on TCP/IP. The blackboard architecture has been ported to Windows/NT, and agents that encapsulate Microsoft API's will be developed. Also planned is a port of the blackboard interpreter to the

المنارة للاستشارات
www.manaraa.com

Macintosh. When completed, the architecture will support multiple hardware and software platforms in a distributed environment.

FUTURE PLANS

In addition to the integration activities discussed above, a number of future research activities are needed. In order that an agent be invocable, its capabilities need to be mapped into terms understood by the ensemble of agents, and also by users. Moreover, as discussed earlier, the natural language vocabulary needed to invoke an agent's services, including lexical, syntactic, and semantic properties, will also be posted on the blackboard for use by the user interface. In general, however, this advertising of vocabulary can lead to conflicts among definitions. We intend to develop an API Description Tool, with which the agent designer describes the services provided by that agent. The tool will produce mappings of expressions in ICL into those services, including vocabulary and knowledge representations that can be merged into a common whole. Techniques used in developing natural language database porting tools (e.g., TEAM [11]) will be investigated.

In order to generalize the simulation approach in MAILTALK to encompass the entire collection of agents, the API Description Tool also needs to supply information sufficient to allow the agent architecture to simulate an agent's behavior. It will need to characterize the preconditions and effects of agent actions, thereby also providing a basis for a server's planning to incorporate the agent into a complex action that satisfies a user's stated goal [7].

Finally, an interesting question is where to situate the temporal reasoning subsystem. Currently, it is located with the blackboard server, but it could also be distributed as part of the agent layer, enabling other agents to accept complex expressions for evaluation and/or routing. We intend to experiment with various architectures.

References

[1] S.-K. Chang and L. Leung. A knowledge-based message management system. *ACM Transactions on Office Information Systems*, 5(3):213–236, 1987.

[2] P. R. Cohen. Integrated interfaces for decision support with simulation. In B. Nelson, W. D. Kelton, and G. M. Clark, editors, *Proceedings of the Winter Simulation Conference*, pages 1066–1072. Association for Computing Machinery, December 1991. invited paper.

[3] P. R. Cohen. The role of natural language in a multimodal interface. In *The 2nd FRIEND21 International Symposium on Next Generation Human Interface Technologies*, Tokyo, Japan, November 1991. Institute for Personalized Information Environment. Also appears in Proceedings of UIST'92, ACM Press, New York, 1992, 143-149.

[4] P. R. Cohen, M. Dalrymple, D. B. Moran, F. C. N. Pereira, J. W. Sullivan, R. A. Gargan, J. L. Schlossberg, and S. W. Tyler. Synergistic use of direct manipulation and natural language. In *Human Factors in Computing Systems: CHI'89 Conference Proceedings*, pages 227–234, New York, New York, April 1989. ACM, Addison Wesley Publishing Co.

[5] P. R. Cohen and H. J. Levesque. Rational interaction as the basis for communication. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*. MIT Press, Cambridge, Massachusetts, 1990.

[6] M. Dalrymple. The interpretation of tense and aspect in English. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, Buffalo, New York, June 1988.

[7] O. Etzioni, N. Lesh, and R. Segal. Building softbots for UNIX. Department of Computer Science and Engineering, University of Washington, unpublished ms., November 1992.

[8] D. Gelernter. *Mirror Worlds*. Oxford University Press, New York, 1993.

[9] M. Genesereth and N. P. Singh. A knowledge sharing approach to software interoperation. Computer Science Department, Stanford University, unpublished ms., January 1994.

[10] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaboratorive filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, December 1992.

[11] B. J. Grosz, D. Appelt, P. Martin, and F. Pereira. Team: An experiment in the design of transportable natural language interfaces. *Artificial Intelligence*, 32(2):173–244, 1987.

[12] T. W. Malone, K. R. Grant, F. A. Turbak, S. A. Brobst, and M. D. Cohen. Intelligent information-sharing. *Communications of the ACM*, 30(5):390–402, May 1987.

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

[14] S. L. Oviatt, P. R. Cohen, and M. Wang. Reducing linguistic variability in speech and handwriting through selection of presentation format. In K. Shirai, editor, *Proceedings of the International Conference on Spoken Dialogue: New Directions in Human-Machine Communication*, Tokyo, Japan, November 1993.

www.manaraa.com

[15] M. Palaniappan, N. Yankelovitch, G. Fitzmaurice, A. Loomis, B. Haan, J. Coombs, and N. Meyrowitz. The Envoy framework: An open architecture for agents. *ACM Transactions on Information Systems*, 10(3):233–264, July 1992.

[16] S. Pollock. A rule-based message filtering system. *ACM Transactions on Office Information Systems*, 6(3):232–254, July 1988.

[17] D. G. Schwartz. Cooperating heterogeneous systems: A blackboard-based meta approach. Technical Report 93-112, Center for Automation and Intelligent Systems Research, Case Western Reserve University, Cleveland, Ohio, April 1993. Unpublished Ph.D. thesis.

[18] J. R. Searle. *Speech acts: An essay in the philosophy of language*. Cambridge University Press, Cambridge, 1969.

[19] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[20] R. Turlock. SIFT: A Simple Information Filtering Tool. Bellcore, Mountain, New Jersey, 1993.

المنارة للاستشارات
www.manaraa.com