Multi-Agent Integration of Information Gathering and Decision Support

Katia Sycara[1] and **Dajun Zeng**[2]

Abstract.

We are investigating techniques for developing distributed and adaptive collections of information agents that coordinate to retrieve, filter and fuse information relevant to the user, task and situation. In our system of agents, information gathering is seamlessly integrated with decision support. In this paper we present the distributed system architecture, agent collaboration interactions, and a reusable set of software components for structuring agents. The system has three types of agents: *Interface agents* interact with the user receiving user specifications and delivering results. They acquire, model, and utilize user preferences to guide system coordination in support of the user's tasks. *Task agents* help users perform tasks by formulating problem solving plans and carrying out these plans through querying and exchanging information with other software agents. *Information agents* provide intelligent access to a heterogeneous collection of information sources. We have implemented this system framework and are developing collaborating agents in diverse complex real world tasks, such as organizational decision making, and financial portfolio management.

1 Introduction

The use of the Internet has accelerated at an unprecedented pace. However, effective use of the Internet by humans or decision support machine systems has been hampered by some dominant characteristics of the Infosphere. First, information available from the net is unorganized, multi-modal, and distributed on server sites all over the world. Second, the number and variety of data sources and services is dramatically increasing every day. Furthermore, the availability, type and reliability of information services are constantly changing. Third, the same piece of information can be accessible from a variety of different information sources. Fourth, information is ambiguous and possibly erroneous due to the dynamic nature of the information sources and potential information updating and maintenance problems. Therefore, information is becoming increasingly more difficult for a person or machine system to collect, filter, evaluate, and use in problem solving. As a result, the problem of locating information sources, accessing, filtering, and integrating information in support of decision making, as well as coordinating information retrieval and problem solving efforts of information sources and decision-making systems has become a very critical task.

The notion of Intelligent Software Agents (e.g., [12, 15, 6, 14]) has been proposed to address this challenge. Although a precise definition of an intelligent agent is still forthcoming, the current working notion is that Intelligent Software Agents are programs that act on behalf of their human users in order to perform laborious information gathering tasks, such as locating and accessing information from various on-line information sources, filter away irrelevant or unwanted information, and adapt over time to their human users' information needs and the shape of the Infosphere. Most current agent-oriented approaches have focussed on what we call *interface agents*—a single agent with simple knowledge and problem solving capabilities whose main task is information filtering to alleviate the user's cognitive overload (e.g., [8, 9]). Another type of agent is the *SoftBot* ([3]), a single agent with general knowledge that performs a wide range of user-delegated information-finding tasks. We believe that such centralized approaches have several limitations. A single general agent would need an enormous amount of knowledge to be able to deal effectively with user information requests that cover a variety of tasks. In addition, a centralized system constitutes a processing bottleneck and a "single point of failure". Furthermore, unless the agent has beyond the state of the art learning capabilities, it would need considerable reprogramming to deal with the appearance of new agents and information sources in the environment. Finally, because of the complexity of the information finding and filtering task, and the large amount of information, the required processing would overwhelm a single agent.

Another proposed solution is to use multi-agent computer systems to access, filter, evaluate, and integrate this information [14, 10]. Such multi-agent systems can compartmentalize specialized task knowledge, organize themselves to avoid processing bottlenecks, and can be built expressly to deal with dynamic changes in the agent and information-source landscape. In addition, Multiple Intelligent Coordinating Agents are ideally suited to the predominant characteristics of the Infosphere, such as the heterogeneity of the information sources, and the presence of multiple users with related information needs. We therefore believe that a distributed approach is superior, and possibly the only one that would work for information gathering and coherent information fusion.

The context of multi-agent systems widens the notion of intelligent agent in at least two general ways. First, an agent's "user" that imparts goals to it and delegates tasks can be not only a human but also another agent. Second, an agent must have been designed with explicit mechanisms for communicating and interacting with other agents. Our notion is that such multi agent systems may comprise *interface agents* tied closely to an individual human's goals, *task agents* involved in the processes associated with arbitrary problem-solving tasks, and *information agents* that are closely tied to a source or sources of data.

In this paper, we report on our work on developing distributed collections of intelligent software agents that cooperate asynchronously

[1] The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, U.S.A.
[2] The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, U.S.A.

ECAI 96. *12th European Conference on Artificial Intelligence*
Edited by W. Wahlster
Published in 1996 by John Wiley & Sons, Ltd.

to perform goal-directed information retrieval and information integration in support of performing a variety of decision making tasks. In particular, we will address research issues involved in designing multiple Intelligent Agents that coordinate with each other to support information gathering, filtering, and integration.

We will focus on three crucial characteristics of our architecture that differentiate our work from others: (1) *multi-agent* system, (2) the agents *actively* seek out information, and (3) the information gathering is *seamlessly integrated* with problem solving. We will present the overall architectural framework, our agent design commitments, and agent architecture to enable the above characteristics.

2 Different Types of Information and Decision Making Agents

Our distributed agent-based architecture has three types of agents (see Figure 1): *interface* agents, *task* agents, and *information* agents. The architecture of all these agents follows the general BDI type philosophy [12], however, each of them embodies particular architectural design commitments to make them effective in dealing with the particular category of issues of its type. As our point of departure, we use the Task Control Architecture (TCA) framework [13] which we extend and specialize for real-time user interaction, information gathering, and decision support tasks in the Infosphere. Before we present the general agent architecture and coordination in Section 3, we discuss first the characteristics of the different types of agents.

Interface agents interact with the user receiving user specifications and delivering results. They acquire, model, and utilize user preferences to guide system coordination in support of the user's tasks. For example, an agent that filters electronic mail according to its user's preferences is an interface agent. Task agents support decision making by formulating problem solving plans and carrying out these plans through querying and exchanging information with other software agents. Task agents have knowledge of the task domain, and which other task assistants or information assistants are relevant to performing various parts of the task. For example, an agent that makes stock buy or sell recommendations is a task agent. Information agents provide intelligent access to a heterogeneous collection of information sources depicted at the bottom of Figure 1. They find information in response to queries and also actively monitor the Infosphere for specified conditions. Information assistants have models of the associated information resources, and strategies for source selection, information access, conflict resolution and information fusion. For example, an agent that monitors stock prices of the New York Stock Exchange is an information agent.

The crucial factors influencing the determination of the type of an agent are: (1) its functional and informational scope, (2) predominant types of agent interactions, and (3) constituent reusable agent architecture components.

2.1 Agent Organization, Coordination and Interactions

Agents are directly activated based on the top-down elaboration of the current situation. These agent activations dynamically form an organizational structure "on-demand" that fits in with the task, the user's information needs, and resulting decomposed information requests from related software agents. This task-based organization may change over time (if, for example, some task characteristics were to change for a given task), but will also remain relatively static for extended periods. Notice that the agent organization will

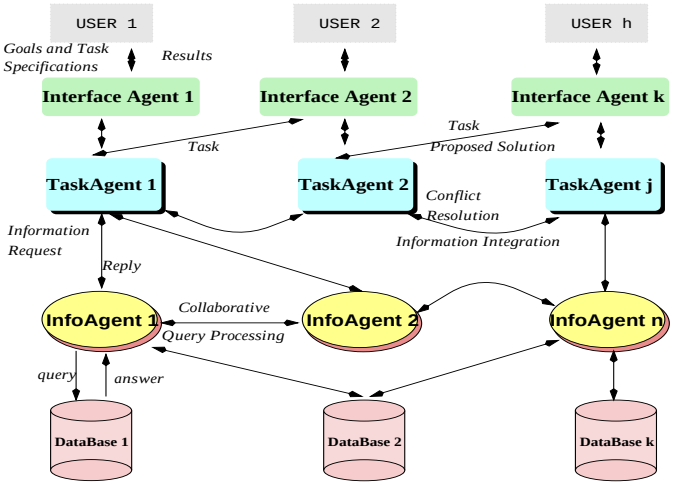

Figure 1. Distributed System Architecture

not change as a result of appearance or disappearance of information sources but the agent interactions could be affected by appearance (or disappearance) of agents that are capable of fulfilling task subgoals in new ways. Information that is important for decision-making (and thus might cause an eventual change in organizational structuring) is monitored at the lowest levels of the organization and passed upward when necessary. In this type of organization, task-specific agents continually interleave planning, scheduling, coordination, and the execution of domain-level problem-solving actions.

Obviously, one of the major issues involved in multi-agent systems is the problem of interoperability and communication between the agents. In our framework, we use KQML [4] for inter-agent communication. In order to incorporate and utilize pre-existing software agents or information services that have been developed by others, we adopt the following strategy: If the agent is under our control, it will be built using KQML as a communication language. If not, we build a gateway agent that connects the legacy system to our agent.

3 Agent Architecture: Usable Control Constructs

In order to operate in rich, dynamic, multi-agent environments, software agents must be able to effectively utilize and coordinate their limited computational resources. As our point of departure in structuring an agent, we extend and specialize the Task Control Architecture (TCA) [13] for real-time user interaction, information gathering, and decision support. The overall architectural design of a TCA-based agent is shown in Figure 2.

The planning module takes as input a set of goals and produces a plan that satisfies the goals. The key component of this architecture is a hierarchical representation of task/subtask relationships[13]. This representation, called a *task tree*, has goals as non-terminal nodes, and executable actions and execution monitoring mechanisms at the leaves. Temporal constraints between nodes are used to schedule task planning and execution: actions are queued until their temporal constraints are satisfied. For example, a *sequential-achievement* constraint between two nodes implies that all actions associated under the first node must be handled before any of those under the second node; whereas a *parallel-achievement* constraint allows that the actions under the first node can be parallelly executed along with the

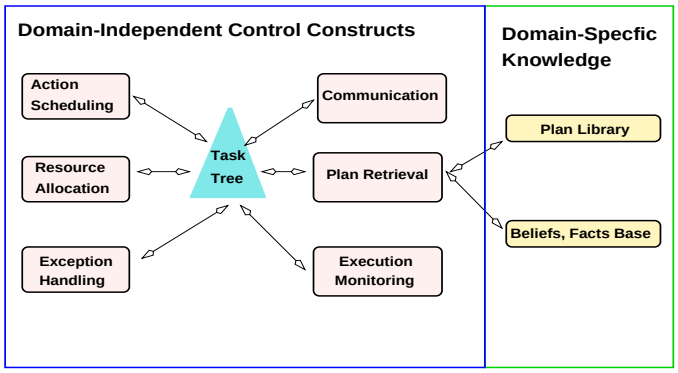

Figure 2. Agent Architecture

actions under the second node. This combination of hierarchical task decomposition and temporal constraints form the agent's representation of plans. Either a first principle general planner or a plan retrieval component plus domain-specific plan fragments can be used to generate plans. We adopt the plan retrieval approach in our implementation because of efficiency considerations.

We have extended the original TCA architecture with a communication module that accepts and interprets messages from other agents in KQML. In addition, interface agents also accept and interpret e-mail messages. We have found that e-mail is a convenient medium of communicating with the user and/or other interface agents (e.g., agents that provide event notification services). Messages can contain request for services. These requests become goals of the recipient agent.

For the information agents, there are three important types of goals; (1) Answering a one-shot query about associated information sources, (2) Answering periodic queries that will be run repeatedly, and the results sent to the requester each time (e.g., "tell me the price of IBM every 30 minutes"), and (3) Monitoring an information source for a change in a piece of information (e.g., "tell me if the price of IBM drops below $80 within 15 minutes of the occurrence of that event").

The scheduling module schedules each of the plan steps. The agent scheduling process in general takes as input the agent's current set of plan instances, in particular, the set of all executable actions, and decides which action, if any, is to be executed next. Whereas for task agents, scheduling can be very sophisticated, in our initial implementation of information agents, we use a simple earliest-deadline-first schedule execution heuristic.

Agent reactivity considerations are handled by the *execution monitoring* and *exception handling* processes. The agent execution monitoring process takes as input the agent's next intended action and prepares, monitors, and completes its execution. The execution monitor prepares an action for execution by setting up a context (including the results of previous actions, etc.) for the action. It monitors the action by optionally providing the associated computation limited resources—for example, the action may be allowed only a certain amount of time and if the action does not complete before that time is up, the computation is interrupted and the action is marked as having failed.

When an action is marked as failed, the exception handling process takes over to replan from the current execution point to help the agent recover from the failure. For instance, when a certain external information source is out of service temporarily, the agent who needs data from this information source shouldn't just wait passively until the service is back. Instead, the agent might want to try another information source or switch its attention to other tasks for a certain period of time before returning to the original task. Whenever an agent fails to retrieve the information of interest from a certain source within a predetermined time limit, the agent will automatically invoke an exception handling routine, which might invoke a replanning process or simply wait for a particular time interval before re-trying accessing the information. Upon completion of an action, results are recorded, downstream actions are enabled if so indicated, and statistics collected.

The agent's *plan library* contains skeletal plans and plan fragments that are indexed by goals and can be retrieved and instantiated according to the current input parameters. The retrieved and instantiated plan fragments are used to form the agent's task tree that is incrementally executed.

The *belief and facts* data structures contain facts and other knowledge related to the agent's functionality. For example, the belief structures of an interface agent contain the user profile, and the belief structures of an information agent contain a local data base that holds relevant records of external information sources the agent is monitoring. Since an information agent does not have control of information sources on the Internet, it must retrieve and store locally any information that it must monitor. For example, suppose an information agent monitors the Security APL, an Internet source that provides the New York Stock Exchange data, to satisfy another agent's monitoring request, "notify me when the price of IBM exceeds $80". The information agent must periodically retrieve the price of IBM from the Security APL, bring it to its local data base and perform the appropriate comparison. For information agents, the local data base is a major part of their reusable architecture. It is this local database that allows all information agents to present a consistent interface to other agents, and re-use behaviors, even in very different information environments [1].

An agent architecture may also contain components that are not reusable. For example, the architecture of information agents contains a small amount of site-specific external query interface code.

Since task tree management, plan retrieval, action scheduling, execution monitoring, resource allocation, and exception handling are handled by the agent in a domain-independent way, all these control constructs are reusable. Therefore the development of a new agent is simplified and involves the following steps:

- Build the domain-specific plan library
- Develop the domain-specific knowledge-base
- Instantiate the reusable agent control architecture using the domain-specific plan library and knowledge-base

3.1 An Example of Agent Planning and Execution

We present an example of how the agent architecture is used in the control of one of our task agents, called `Personnel Finder`, describing in detail how the task tree models associated with `Personnel Finder` are generated (See Figure 3) and how the control constructs operate on this tree. The basic functionality of `Personnel Finder` is, given a person's name, find relevant personnel information, such as title, phone number, office number, etc. The current implementation of `Personnel Finder` can access a variety of information sources that are either locally available to the Carnegie Mellon community or are distributed over the Internet.

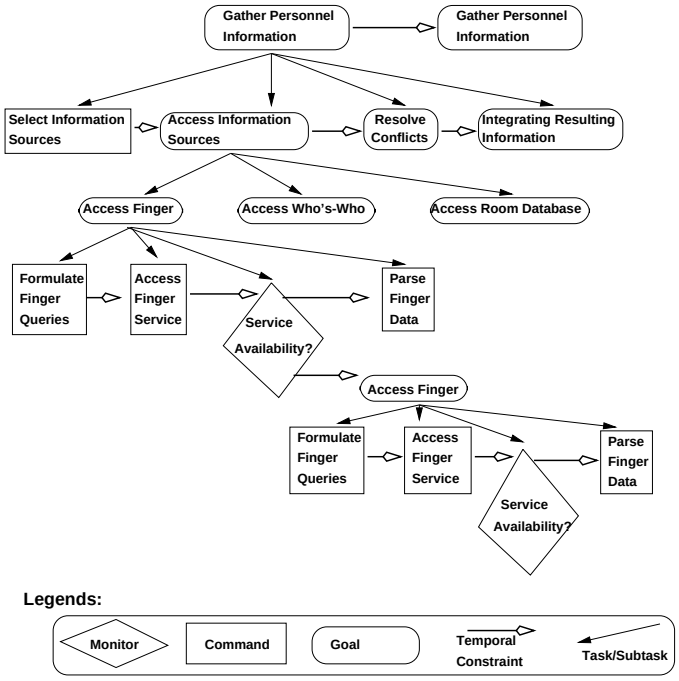

Figure 3. Task Tree for `Personnel Finder`

The `Personnel Finder` receives "Gathering Personnel Information" goal in messages coming from other software agents or from a user interface directly. Since "Gathering Personnel Information" is not a terminal node (it is not directly executable), the communication component forwards the "Gathering Personnel Information" goal message to the plan retrieval component, queuing any other personnel information gathering goals that might also have been received. Based on the goal message and the associated parameters (e.g., a person's name), the plan retrieval component first finds the appropriate plan fragment from the plan library and then instantiates this fragment using these parameters. Once the instantiation is done, the plan retrieval component issues a "Select Information Sources" command, an "Access Information Sources" sub-goal, a "Resolve Conflicts" sub-goal, and an "Integrate Resulting Information" sub-goal, and attaches them to the task tree. The agent immediately executes the "Select Information Sources" since it is a terminal node and readily executable. When this action completes, the communication component sends the "Access Information Sources" goal message to the plan retrieval component to get the instantiated plan fragment capable of accomplishing the information accessing goal. This goal message is also added to the task tree. Note that the task tree always reflects the current state of the plan and plan execution and is updated incrementally.

After the plan retrieval component finishes plan instantiation, it issues a "Access Finger" sub-goal, a "Access Who's-Who" sub-goal, and a "Access Room Database" sub-goal. Besides the finger utility for accessing a person's plan file, Carnegie Mellon University (CMU) has two data bases, the "Who's Who at CMU" which is part of the electronic University Library system, and a database containing room and telephone information for CMU employees. It should be noted that since there is no *sequential-achievement* constraint existing among these sub-goals, they are being handled *concurrently*. We will focus on the first one, "Access Finger". The other two are handled in a similar way. Once again, the plan retrieval component is invoked to decide what to do to accomplish this goal. As a result of plan instantiation, the following nodes are added to the task tree: a "Formulate Finger Query" command (The major functionality of "Formulate Finger Query" is to compose heuristically the email address given the name and affiliation of a given person), a "Access Finger Service" command, a monitor to ensure that the finger action has been carried out properly, and a "Parse Finger Data" command. In turn, these actions are carried out and the monitoring condition is checked. If the finger action reports a failure, an exception is raised and the plan retrieval component is invoked to replan from the current position. As a result of replanning, the "Formulate Finger Query" may try a different email address. If everything goes well, after all "Access Finger", "Access Who's-Who", and "Access Room Database" finish, the agent continues similar *plan retrieval-execution-execution monitoring* cycles for "Resolve Conflicts" and "Integrate Resulting Information" subgoals. After this particular instance of "Gathering Personnel Information" completes, the agent waits for the next "Gathering Personnel Information" cycle. Since the control mechanism is able to monitor the time spent for tasks/subtasks and the depth of the task tree, it is fairly easy to constraint the computational resources dedicated to certain tasks either by enforcing an absolute timeout constraint or limiting the number of retries.

4 Everyday Organizational Decision Making

In performing everyday routine tasks, people spend much time in finding, filtering, and processing information. Delegating some of the information processing to Intelligent Agents could increase human productivity and reduce cognitive load. Within the context of our PLEIADES project, we have applied our distributed agent architecture to tasks, such as distributed, collaborative meeting scheduling among multiple human attendees [7], finding people information on the Internet, hosting a visitor to Carnegie Mellon University [14], accessing and filtering information about conference announcements and requests for proposals (RFPs) from funding organizations and notifying Computer Science faculty of RFPs that suit their research interests [11].

5 An Extended Example: The Visitor Hosting Task

We will use the task of hosting a visitor to Carnegie Mellon University (CMU) as an illustrative example of system operation. Hosting a visitor involves arranging the visitor's schedule with faculty whose research interests match the interests that the visitor has expressed in his/her visit request. A different variation of the hosting visitor task has also been explored by Kautz and his colleagues at Bell Labs [5].

Our system consisting of a collection of agents, collectively referred to as *Visitor Hosting system*, supports the visitor hosting task. The Visitor Hosting system takes as input a visit request, the tentative requested days for the meeting and the research interests of the visitor. Its final output is a detailed schedule for the visitor consisting of time, location, and name of attendees. Attendees in these meetings are faculty members whose interests match the ones expressed in the visitor's request and who have been automatically contacted by the agents in the Visitor Hosting system and have agreed to meet with the visitor at times convenient for them. The Visitor Hosting system has an interface agent which interacts with the person who is hosting the visit. It also has the following task agents: (1) a `Personnel`

`Finder` task agent, who finds detailed information about the visitor, and also finds detailed information about CMU faculty for better matching the visitor and the faculty he/she meets, (2) the visitor's Scheduling task agent, and (3) various personal calendar management task agents that manage calendars of various faculty members. In addition, the Visitor Hosting system has a number of information agents that (1) retrieve information from a CMU data base that has faculty research interests (Interests agent), and (2) retrieve personnel and location information from various university data bases.

The interactions of the various agents in the Visitor Hosting task are:

- The user inputs a visitor request to the Visitor Hoster agent.
- The Visitor Hoster agent extracts the visitor's areas of interest and visitor's name and organization.
- The Visitor Hoster agent passes to the Interests agent the visitor's areas of interest and asks the Interest agent to find faculty members whose interest areas match the request.
- The Visitor Hoster agent passes the name and organization of the visitor to the `Personnel Finder` agent and asks it to find more detailed information about the visitor (e.g. rank in the organization, projects the visitor is working on). The visitor information is used by faculty calendar software agents, such as CAP (see [2]), to decide level of interest of a faculty member to meet with the visitor.
- The `Personnel Finder` agent accesses Internet resources to find more detailed information about the visitor (e.g. visitor's rank).
- Meanwhile, the Interests agent queries the faculty interests data base and returns names of CMU faculty whose research matches the request.
- The Visitor Hoster agent passes the returned faculty names to the `Personnel Finder` agent requesting more information on these faculty.
- The `Personnel Finder` agent submits queries to three personnel data bases (Finger, CMU Who's-Who, CMU Room Database), at CMU to find more detailed information about the faculty member (e.g., rank, telephone number, e-mail address), resolves ambiguities in the returned information, and integrates results.
- Based on the information returned by the `Personnel Finder`, the Visitor Hoster agent selects an initial set of faculty to be contacted. The user can participate in this selection process.
- The Visitor Hoster agent automatically composes messages to the calendar assistant agents of the selected faculty asking whether they are willing to meet with the visitor and at what time. For those faculty that do not have machine calendar agents, e-mail is automatically composed and sent.
- The Visitor Hoster agent collects responses and passes them to the visitor's Scheduling agent.
- The visitor's Scheduling agent composes the visitor's schedule through subsequent interaction and negotiation of scheduling conflicts with the attendees' calendar management agents.[3]

6 Conclusions

In this paper, we have described concepts and techniques for structuring and organizing distributed collections of intelligent software agents in a reusable way. We presented the various agent types that we believe are necessary for supporting and seamlessly integrating information gathering from distributed internet-based information sources and decision support. We have also described and illustrated our implemented, distributed system of such collaborating agents. We believe that such flexible distributed architectures, consisting of reusable agent components, will be able to answer many of the challenges that face users as a result of the availability of the new, vast, net-based information environment. These challenges include locating, accessing, filtering and integrating information from disparate information sources, monitoring the Infosphere and notifying the user or an appropriate agent about events of particular interest in performing the user-designated tasks, and incorporating retrieved information into decision support tasks.

[3] For details on the distributed meeting scheduling algorithm, see [7].

ACKNOWLEDGEMENTS

This research has been sponsored in part by ONR Grant #N00014-95-1-1092, by ARPA Grant #F33615-93-1-1330, and by NSF Grant #IRI-9508191.

REFERENCES

[1] Keith Decker and Katia Sycara, 'Designing reusable behaviors for information agents', Technical report, The Robotics Institute, Carnegie Mellon University, Pittsburgh, U.S.A., (1996).

[2] Lisa Dent, Jesus Boticario, John McDermott, Tom Mitchell, and David Zabowski, 'A personal learning apprentice', in *Proceedings of the Tenth National Conference on Artificial Intelligence*. AAAI, (1992).

[3] Oren Etzioni and Daniel Weld, 'A softbot-based interface to the internet', *Communications of the ACM*, **37**(7), (July 1994).

[4] Tim Finin, Rich Fritzson, and Don McKay, 'A language and protocol to support intelligent agent interoperability', in *Proceedings of the CE and CALS Washington 92 Conference*, (June 1992).

[5] Henry A. Kautz, Bart Selman, and Michael Coen, 'Bottom-up design of software agents', *Communications of the ACM*, **37**(7), (July 1994).

[6] Kan Lang, 'Newsweeder: Learning to filter netnews', in *Proceedings of Machine Learning Conference*, (1995).

[7] JyiShane Liu and Katia Sycara, 'Distributed meeting scheduling', in *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, Atlanta, Georgia, (August 13-16 1994).

[8] Pattie Maes, 'Agents that reduce work and information overload', *Communications of the ACM*, **37**(7), (July 1994).

[9] Tom Mitchell, Rich Caruana, Dayne Freitag, John McDermott, and David Zabowski, 'Experience with a learning personal assistant', *Communications of the ACM*, **37**(7), (July 1994).

[10] Tim Oates, M. V. Nagendra Prasad, and Victor R. Lesser, 'Cooperative information gathering: A distributed problem solving approach', Technical Report 94-66, Department of Computer Science, University of Massachusetts, (September 1994).

[11] Anandeep Pannu and Katia Sycara, 'Learning text filtering preferences', in *1996 AAAI Symposium on Machine Learning and Information Access*, (1996).

[12] Anand S. Rao and Michael P. Georgeff, 'A model-theoretic approach to the verification of situated reasoning systems', in *Proceedings of IJCAI-93*, pp. 318–324, Chambery, France, (28 August - 3 September 1993). IJCAI.

[13] Reid Simmons, 'Structured control for autonomous robots', *IEEE Journal of Robotics and Automation*, (1994).

[14] Katia Sycara and Dajun Zeng, 'Towards an intelligent electronic secretary', in *Proceedings of the CIKM-94 (International Conference on Information and Knowledge Management) Workshop on Intelligent Information Agents*, National Institute of Standards and Technology, Gaithersburg, Maryland, (December 1994).

[15] M. Wooldridge and N. R. Jennings, 'Intelligent agents: Theory and practice', *The Knowledge Engineering Review*, **10**(2), 115–152, (1995).