Distributed Intelligent Agents

Katia Sycara Keith Decker Anandeep Pannu
Mike Williamson
Dajun Zeng
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.
Voice: (412) 268-8825 Fax: (412) 268-5569
URL: `http://www.cs.cmu.edu/~softagents/`

Abstract

We are investigating techniques for developing distributed and adaptive collections of agents that coordinate to retrieve, filter and fuse information relevant to the user, task and situation, as well as anticipate a user's information needs. In our system of agents, information gathering is seamlessly integrated with decision support. The task for which particular information is requested of the agents does not remain in the user's head but it is explicitly represented and supported through agent collaboration. In this paper we present the distributed system architecture, agent collaboration interactions, and a reusable set of software components for constructing agents. We call this reusable multi-agent computational infrastructure RETSINA (Reusable Task Structure-based Intelligent Network Agents). It 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 (the PLEIADES system), and financial portfolio management (the WARREN system).

1 Introduction

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, 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 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., [1, 19, 20, 25, 13, 22]) 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, resolving inconsistencies in the retrieved information, filtering away irrelevant or unwanted information, integrating information from heterogeneous information sources and adapting 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., [15, 16]). Another type of agent is the *Softbot* ([6]), 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". 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 address the problem by using multi-agent systems to access, filter, evaluate, and integrate this information [23, 17]. 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, the diversity of information gathering and problem solving tasks that the gathered information supports, 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 might 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. An information agent is different from an interface agent in that an information agent is tied more closely to the data that it is providing, while an interface agent closely interacts with the user. Typically, a single information agent will serve the information needs of many other agents (humans or intelligent software agents). An information agent is also quite different from a typical World Wide Web (WWW) service that provides data to multiple users. Besides the obvious interface differences, an information agent can reason about the way it will handle external requests and the order in which it will carry them out (WWW services are typically blindly concurrent). Moreover, information agents not only perform information gathering in response to queries but also can carry out long-term interactions that involve monitoring the Infosphere for particular conditions, as well as information updating.

In this paper, we report on our work on developing distributed collections of intelligent software agents that cooperate asynchronously to perform goal-directed information retrieval and information integration in support of performing a variety of decision making tasks [23, 2]. We have been developing RETSINA, an open society of reusable agents that self organize and cooperate in response to task requirements. In particular, we will focus on three crucial characteristics of the overall framework that differentiate our work from others:

- ours is a *multi-agent* system where the agents operate asynchronously and collaborate with each other and their users,
- the agents *actively* seek out information,
- the information gathering is *seamlessly integrated* with problem solving and decision support

We will present the overall architectural framework, our agent design commitments, and agent architecture to enable the above characteristics. We will draw examples from our work on Intelligent Agents in the domains of organizational decision making (the PLEIADES system), and financial portfolio management (the WARREN system).

The rest of the paper is organized as follows. Section 2 briefly lists some agent characteristics we consider desirable. Section 3 motivates the distributed architecture for intelligent information retrieval and problem solving, and presents an overview of the system architecture, the different types of agents in the proposed multi agent organization, and agent coordination mechanisms. Section 4 presents in detail the reusable agent architecture and discusses planning, control, and execution monitoring in agent operations. Description and examples from the application of RETSINA to everyday organizational decision making and financial portfolio management are given in Section 5. Section 6 presents concluding remarks.

2 Desirable Agent Characteristics

Many different definitions of intelligent agents have been proposed. In this section, we give a brief list of what we see as essential characteristics of intelligent agents.

- *taskable*. By "taskable" we mean agents that can take direction from humans or other agents.
- *network-centric*: by this we mean that agents should be distributed and self organizing. When situations warrant it, agent mobility may also be desirable.
- *semi-autonomous* rather than under direct human control all the time. For example, in an information gathering task, because of the large amount of potential requests for information, humans would be swamped, if they had to initiate every single information request. The amount of agent autonomy should be user controllable.
- *persistent*, i.e. capable of long periods of unattended operation.
- *trustworthy*: An agent should serve users' needs in a reliable way so that users will develop trust in its performance.
- *anticipatory*: An agent should anticipate user information needs through task, role and situational models as well as learning to serve as an intelligent cache, acquiring and holding information likely to be needed.
- *active*: An agent should initiate problem solving activities (e.g. monitor the infosphere for the occurrence of given patterns), anticipate user information needs and bring to the attention of users situation-appropriate information, deciding when to fuse information or present "raw" information.
- *collaborative* with humans and with other machine agents. Collaborative agent interactions allow them to increase their local knowledge, resolve conflicts and inconsistencies in information, current task and world models, thus improving their decision support capabilities.

- *able to deal with heterogeneity* of other agents and information resources.

- *adaptive* to changing user needs, and task environment.

3 Distributed Intelligent Agents in Information Processing and Problem Solving

In this section, we motivate and describe the distributed agent framework for intelligent information retrieval and problem solving, and then present the agent coordination mechanisms. The issues of how to engineer these agents are the topics of Section 4. RETSINA has been motivated by the following considerations:

- *Distributed information sources:* Information sources available on-line are inherently distributed. Furthermore, these sources typically are of different modalities. Therefore it is natural to adopt a distributed architecture consisting of many software agents specialized for different heterogeneous information sources.

- *Sharability:* Typically, user applications need to access several services or resources in an asynchronous manner in support of a variety of tasks. It would be wasteful to replicate agent information gathering or problem solving capabilities for each user and each application. It is desirable that the architecture support sharability of agent capabilities and retrieved information.

- *Complexity hiding:* Often information retrieval in support of a task involves quite complex coordination of many different agents. To avoid overloading the user with a confusing array of different agents and agent interfaces, it is necessary to develop an architecture that hides the underlying distributed information gathering and problem solving complexity from the user.

- *Modularity and Reuseability:* Although software agents will be operating on behalf of their *individual patrons*—human users, or other agents, pieces of agent code for a particular task can be copied from one agent to another and can be customized for new users to take into consideration particular users' preferences or idiosyncrasies. One of the basic ideas behind the distributed agent-based approach is that software agents will be kept simple for ease of maintenance, initialization and customization. Another facet of reuseability is that pre-existing information services, whose implementation, query language and communication channels are beyond the control of user applications, could be easily incorporated in problem-solving.

- *Flexibility:* Software agents can interact in new configurations "on-demand", depending on the information requirements of a particular decision making task.

- *Robustness:* When information and control is distributed, the system is able to degrade gracefully even when some of the agents are out of service temporarily. This feature of the system has significant practical implications because of the dynamic and unstable nature of on-line information services.

- *Quality of Information:* The existence of (usually partial) overlapping of available information items from multiple information sources offers the opportunity to ensure (and probably enhance) the correctness of data through cross-validation. Software agents providing the same piece of information can interact and negotiate to find the most accurate data.

- *Legacy Data:* Many information sources exist prior to the emergence of the Internet-based agent technology. New functionalities and access methods are necessary for them to become full-fledged members of the new information era. Directly updating these systems, however, is a nontrivial task. A preferable way of updating is to construct agent wrappers around existing systems. These agent wrappers interface to the information sources and information consumers and provide a uniform way of accessing the data as well as offer additional functionalities such as monitoring for changes. This agent wrapper approach offers much flexibility and extensibility. Practically speaking, it is also easier to implement since the internal data structure and updating mechanism of the legacy information systems don't need to be modified.

The above considerations clearly motivate the development of systems of distributed software agents for information gathering and decision support in the Internet-based information environment. The critical question then is how to structure and organize these multiple software agents. Our major research goal is to construct reusable software components in such a way that building software agents for new tasks and applications and organizing them can be relatively easy. It seems difficult to engineer a general agent paradigm which can cover *in an efficient manner* a broad range of different tasks including interaction with the user, acquisition of user preferences, information retrieval, and task-specific decision making. For example, in building an agent that is primarily concerned with interacting with a human user, we need to emphasize acquisition, modeling and utilization of user information needs and preferences. On the other hand, in developing an agent that interacts with information sources, issues of acquiring user preferences are de-emphasized and, instead, issues of information source availability, efficiency of data access, data quality and information source reliability become critical. Therefore, reusable software components must efficiently address the critical issues associated with each of these three agent categories.

3.1 Agent Types

In the RETSINA framework, each user is associated with a set of agents which collaborate to support him/her in various tasks and act on the user's behalf. The agents are distributed and run across different machines. The agents have access to models of the user and of other agents as well as the task and information gathering needs associated with different steps of the task. Based on this knowledge, the agents decide how to decompose and delegate tasks, what information is needed at each decision point, and when to initiate collaborative searches with other agents to get, fuse and evaluate the information. In this way, the information gathering activities of the agents are automatically activated by models of the task and processing needs of the agents rather than wholly initiated by the user. The user can leave some of the information gathering decisions to the discretion of the agents. This saves user time and cognitive load and increases user productivity. The degree of agent autonomy is user-controlled. As a user gains more confidence in the agents' capabilities, more latitude over decisions is given over to them. During search, the agents communicate with each other to request or provide information, find information sources, filter or integrate information, and negotiate to resolve conflicts in information and task models. The returned information is communicated to display agents for appropriate display to the user.

RETSINA has three types of agents (see Figure 1): *interface* agents, *task* agents and *information* 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. The main functions of an interface agent include: (1) collecting relevant information from the user to initiate a task, (2) presenting relevant information including results and explanations, (3) asking the user for additional information during problem solving, and (4) asking for user confirmation, when necessary. From the user's viewpoint, having the user interact only through a relevant interface agent for a task hides the underlying distributed information gathering and problem solving complexity and frees the user from having to know of, access and interact with a potentially large number of task agents and information seeking agents in support of a task. For example, the task of hosting a visitor in a university (see Section 5.1), one of the tasks supported by our intelligent agents, involves more than 10 agents. However, the user interacts directly only with the visitor hoster 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. In addition, task assistants have strategies for resolving conflicts and fusing information retrieved by information agents. A task agent performs most of the autonomous problem solving. It exhibits a higher level of sophistication and complexity than either an interface or an information agent. A task agent (1) receives user delegated task specifications from an interface agent, (2) interprets the specifications and extracts problem solving goals, (3) forms plans to satisfy these goals, (4) identifies information seeking subgoals that are present in its plans, (5) decomposes the plans and coordinates with appropriate task agents or information agents for plan execution, monitoring and results composition. An example of a task agent from the financial portfolio management domain is one that makes recommendations to buy or sell stocks.

Information agents provide intelligent access to a heterogeneous collection of information sources depicted at the bottom of Figure 1. Information agents 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. An information agent's activities are initiated either top down, by a user or a task agent through *queries*, or bottom up through *monitoring* information sources for the occurrence of particular information patterns (e.g., a particular stock price has exceeded a predefined threshold). Once the monitored-for condition has been observed, the information agent sends notification messages to agents that have registered interest in the occurrence of particular information patterns (See Section 5.2). For example, in the financial domain, a human or machine agent may be interested in being notified every time a given stock price has risen by 10%. Thus, information agents are active, in the sense that they actively monitor information sources and *proactively* deliver the information, rather than just waiting for and servicing one-shot information queries.

An information agent may receive in messages from other agents 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").

A useful capability that can be added to all types of agents is *learning*. The agents can retain useful information from their interactions as training examples and utilize various machine learning techniques to adapt to new situations and improve their performance [18, 26, 16].

3.2 Agent Organization and Coordination

In RETSINA, agents are distributed across different machines and are directly activated based on the top-down elaboration of the current situation (as opposed to indirect activation via manager or matchmaker agents [12],

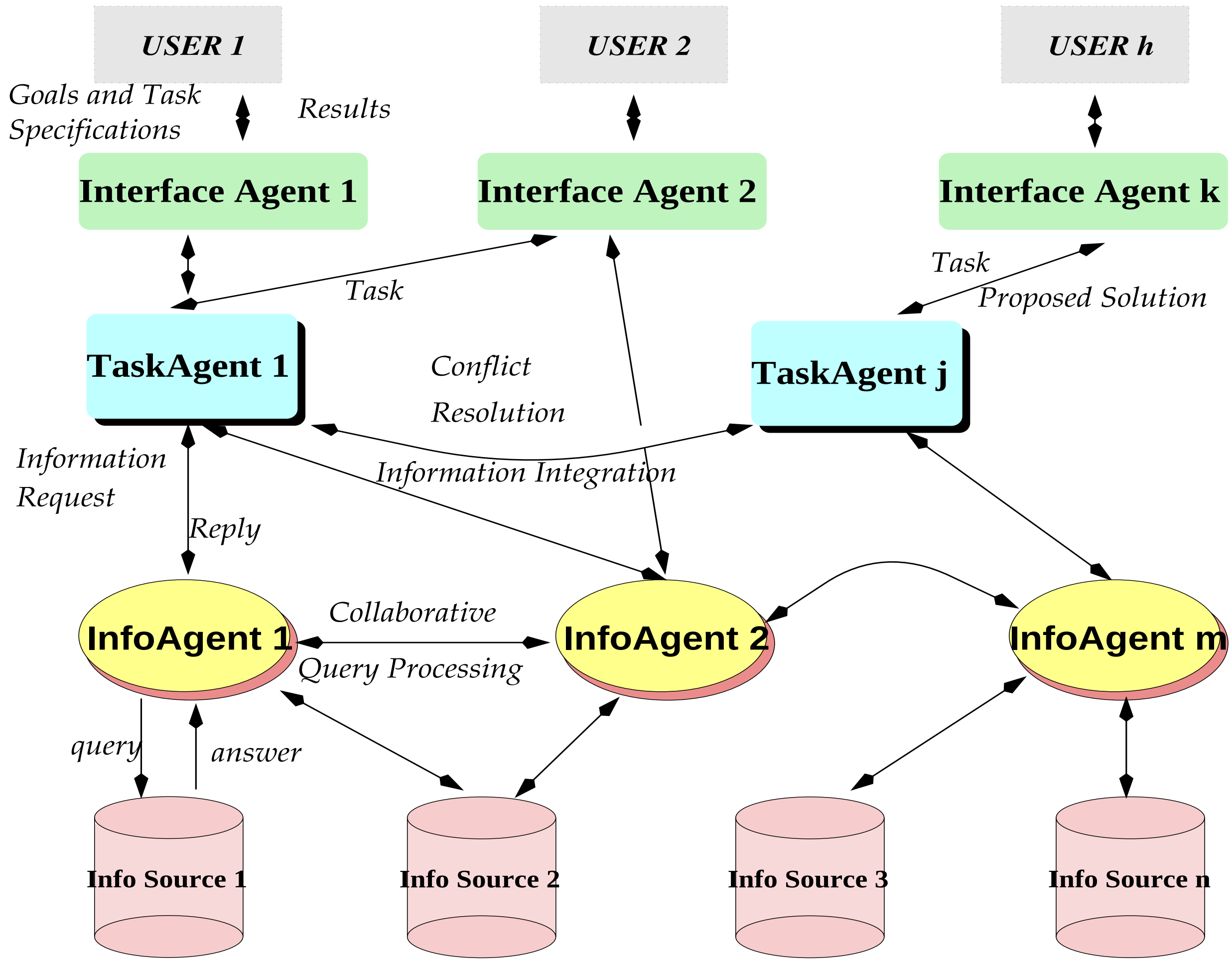


Figure 1: The RETSINA Distributed Agent Organization

or self-directed activation)[1]. 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, but will also remain relatively static for extended periods. Notice that the agent organization will 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.

This system organization has the following characteristics:

- There is a finite number of task agents that each agent communicates with.
- The task agents are eventually responsible for resolving information conflicts and integrating information from heterogeneous information sources for their respective tasks.
- The task agents are responsible for activating relevant information agents and coordinating the information finding and filtering activity for their task.

In our organization, the majority of interactions of interface agents are with the human user, the most frequent interactions of information agents are with information sources, whereas task agents spend most of their processing interacting with other task agents and information agents. We briefly describe the distributed coordination processes. When a task-specific agent receives a task from an interface agent or from another task-specific agent, it decomposes the task based on the domain knowledge it has and then delegates the subtasks to other task-specific agents or directly to information-specific agents. The task-specific agent will take responsibility for collecting data, resolving conflicts, coordinating among the related agents and reporting

[1]Matchmaking is, however used for locating agents.

to whoever initiated the task. The agents who are responsible for assigned sub-tasks will either decompose these sub-tasks further, or perform data retrieval (or possibly other domain-specific local problem solving activities).

When information sources are partially replicated with varying degrees of reliability, cost and processing time, information agents must optimize information source selection. If the chosen information sources fail to provide a useful answer, the information agent should seek and try other sources to re-do the data query. Because of these complexities, we view information retrieval as a planning task itself[11]. The plans that task-specific agents have (see 4) include information gathering goals, which, in turn are satisfied through relevant plans for information retrieval. This type of intelligent agent differs from traditional AI systems since information-seeking during problem solving is an inherent part of the system. In effect, the planning and execution stages are interleaved since the retrieved information may change the planner's view of the outside world or alter the planner's inner belief system.

Information is filtered and fused incrementally by information or task agents as the goals and plans of the various tasks and subtasks dictate, before it is passed on to other agents. This incremental information fusion and conflict resolution increases efficiency and potential scalability (e.g., inconsistencies detected at the information-assistant level may be resolved at that level and not propagated to the task agent level) and robustness (e.g., whatever inconsistencies were not detected during information assistant interaction can be detected at the task-assistant level). A task agent can be said to be proactive in the sense that it actively generates information seeking goals and in turn activates other relevant agents.

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 the KQML language [7] 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 organization and handles different communication channels, different data and query formats, etc.

In open world environments, agents in the system are not statically predefined but can dynamically enter and exit an organization. This necessitates mechanisms for agent locating. This is a challenging task, especially in environments that include large numbers of agents, and where information sources, communication links and/or agents may be appearing and disappearing. We have made initial progress in implementing *matchmaker agents* [12, 3] that act as yellow pages[9]. When an agent is instantiated, it *advertises* its capabilities to a matchmaker. An agent that is looking to find another that possesses a particular capability (e.g. can supply particular information, or achieve a problem solving goal) can query a matchmaker. The matchmaker returns appropriate lists of agents that match the query description, or "null" if it does not currently know of any agent that has this capability. Architecturally, matchmakers are information agents. A matchmaker is an information agent who can find other agents rather than finding pieces of information. One nice property that falls out of this matchmaker design is that, if currently a matchmaker does not know of any agent that can provide a particular requested service, the requesting agent can place a monitoring request that directs the matchmaker to keep looking for an agent whose advertised capability matches the service specification of the requesting agent (the customer). When the matchmaker finds such an appropriate agent, it *notifies* the customer.

Matchmaking is advantageous since it allows a system to operate robustly in the face of agent appearance and disappearance, and intermittent communications (the customer can go back to the matchmaker, looking for a new supplier agent). Matchmaking is significant in another respect: it lays the foundation for evolutionary system design where agents with enhanced capabilities can be gracefully integrated into the system.

4 Agent Engineering: How To Structure An Agent?

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 use the *Task Control Architecture* [21] and *TAEMS*[4], which we extend and specialize for real-time user interaction, information gathering, and decision support.

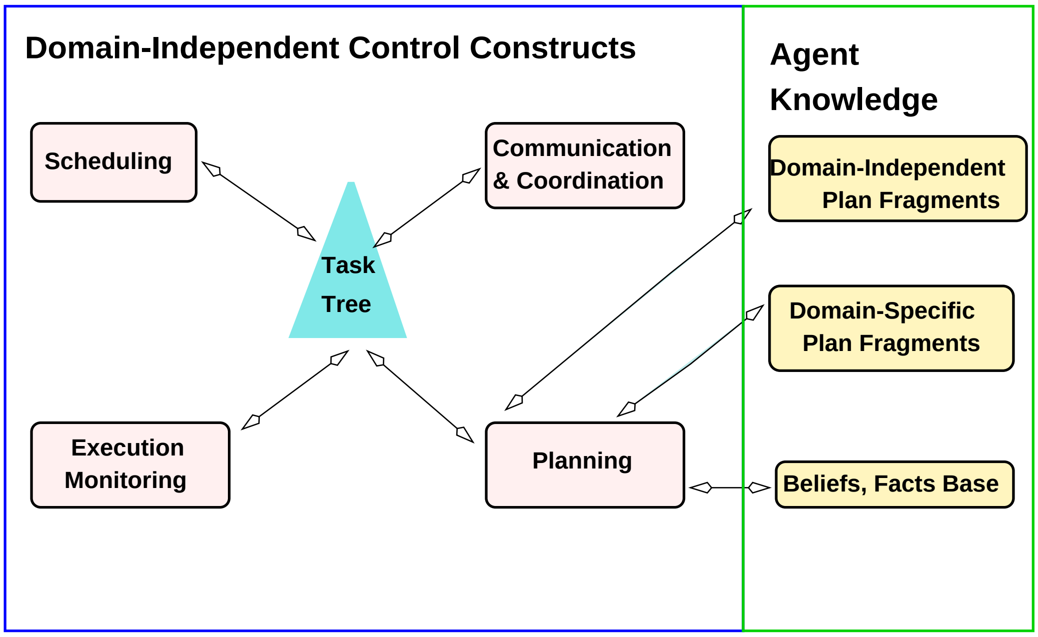


Figure 2: The Agent Architecture: A Functional View

The *planning* module takes as input a set of goals and produces a plan that satisfies the goals. The agent planning process is based on a hierarchical task network (HTN) planning formalism. It takes as input the agent's current set of goals, the current set of task structures, and a library of task reduction schemas. A task reduction schema presents a way of carrying out a task by specifying a set of sub-tasks/actions and describing the information-flow relationships between them. That is, the reduction may specify that the result of one sub-task (e.g. deciding the name of an agent) be provided as an input to another sub-task (e.g. sending a message). Actions may require that certain information be provided before they can be executed, and may also produce information upon execution. For example, the act of sending a KQML messages requires the name of the recipient and the content of the message, while the act of deciding to whom to send some message would produce the name of an agent. An action is *enabled* when all the required inputs have been provided. (See [24] for a complete description of our task network representation.)

The *communication and coordination* module 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, for example agents that provide event notification services. Messages can contain request for services. These requests become goals of the recipient agent.

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. This action is then identified as a fixed intention until it is actually carried out (by the execution component). Whereas for task agents, scheduling can be very sophisticated, in our current implementation of information agents, we use a simple earliest-deadline-first schedule execution heuristic.

To operate in the uncertain, dynamic Infosphere, software agents must be reactive to change for robustness and efficiency considerations. Agent reactivity considerations are handled by the *execution monitoring* process. Execution monitoring 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.

The agent has a *domain-independent library of plan fragments* (task structures) that are indexed by goals, as well as *domain-specific library of plan fragments* from which plan fragments can be retrieved and incrementally instantiated according to the current input parameters. The retrieved and instantiated plan fragments are used to form the agent's instantiated 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 that provides the New York Stock Exchange data is monitoring the Security APL Quote Server web page to satisfy another agent's monitoring request, for example, "notify me when the price of IBM exceeds $80". The information agent must periodically retrieve the price of IBM from the Security APL web page, 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 [2].

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. The external query interface is responsible for actually retrieving data from some external source or sources. The external query interface is usually small and simple, thus minimizing the amount of site-specific code that must be written every time a new information agent is built.

Since task structure management, planning, action scheduling, execution monitoring, 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

5 Application Domains

We have implemented distributed cooperating intelligent agents using the concepts, architecture, and reusable components of the RETSINA multi-agent infrastructure for everyday organizational decision making and for financial portfolio management.

5.1 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. To this end, recent research has produced agents for e-mail filtering, [15], calendar management [5], and filtering news [13]. These tasks involve a single user interacting with a single software agent. There are tasks, however, which have more complex information requirements and possible interaction among many users. A distributed, multi-agent collection of Intelligent Agents is then appropriate and necessary. Within the context of our PLEIADES project, we have applied the distributed RETSINA framework to multi user tasks of increased complexity, such as

- distributed, collaborative meeting scheduling among multiple human attendees [14, 8]
- finding people information on the Internet
- hosting a visitor to Carnegie Mellon University [22]
- 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 [18].

5.1.1 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 in [10].

For expository purposes, we refer to the collection of agents that are involved in the visitor hosting task as the *V*isitor Hosting system. 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, referred to as the Visitor Hoster, which interacts with the person 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.

We present a detailed visitor hosting scenario to illustrate the interactions of the various agents in the Visitor Hosting task.

- The user inputs a visitor request to the Visitor Hoster agent.

  Suppose Marvin Minsky wants to visit CMU CS department. Minsky has requested that he would prefer to meet with CMU faculty interested in machine learning. The user inputs relevant information about Minsky, such as first name, last name, affiliated organization, date and duration of his visit, and his preference as to the interests of faculty he wants to meet with, 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 additional information about the visitor.

- The Personnel Finder agent accesses Internet resources to find requested information about the visitor, such as visitor's title, rank, office address etc. The visitor information is used by faculty calendar software agents, such as CAP (see [16]), to decide level of interest of a faculty member to meet with the visitor.

- Meanwhile, the Interests agent queries the faculty interests data base and returns names of CMU faculty whose research matches the request.

  Using "machine learning" as the keyword to search through faculty interests database, the Interests agent finds a list of faculty whose interest areas match machine learning.

- 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 information sources (`finger`, CMU Who's-Who, CMU Room Database) 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.

Sources: Personnel Info for Tom Mitchell

Info-Attribute-Name	Who-is-Who	Room-Database	Finger
department	✓		
position			✓
office		✓	✗
email	✗		✓
secretary			✓
research			✓

Figure 3: Information Sources and Returned Items

Figure 3 shows in detail the information sources used for querying personnel information about Tom Mitchell, one of the Machine Learning faculty found by the Interests agent, and the information attributes returned by these sources. The columns correspond to different information sources. The rows are the attributes of personnel information that can be obtained from the sources. The checks and cross marks indicate which information sources return answers for which attributes. From this figure, we observe that for some information attributes (e.g., office room number), more than one information source (Room Database and `finger`) offer answers, which may be potentially conflicting. To resolve this conflict, the Personnel Finder applies one of the rules kept in its domain-specific knowledge base saying that the office information based on Room Database is always more relevant and up-to-date than other sources. In this case, the value as to office room number returned by `finger` is overruled by the one returned by Room Database (indicated by the check mark). The cross mark in the "Office" row and "CS-FINGER" column means that although `finger` finds the office information, the retrieved value is overruled by another information source (Room Database).

- 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[2]. The final calendar is shown in Figure 4.

The Visitor Hosting system has many capabilities. It automates information retrievals in terms of finding personnel information of potential appropriate meeting attendees. It accesses various on-line public databases and information resources at the disposal of the visit organizer. It integrates the results obtained from various databases, clarifies ambiguities (e.g., the same entity can be referred by different names in different partially replicated data bases) and resolves the conflicts which might arise from inconsistency between information resources. It creates and manages the visitor's schedule as well as the meeting locations for the various appointments with the faculty members (e.g., a faculty's office, a seminar room). It interacts with the user, getting user input, confirmation or dis-confirmation of suggestions,

[2]For details on the distributed meeting scheduling algorithm, see [14, 8].

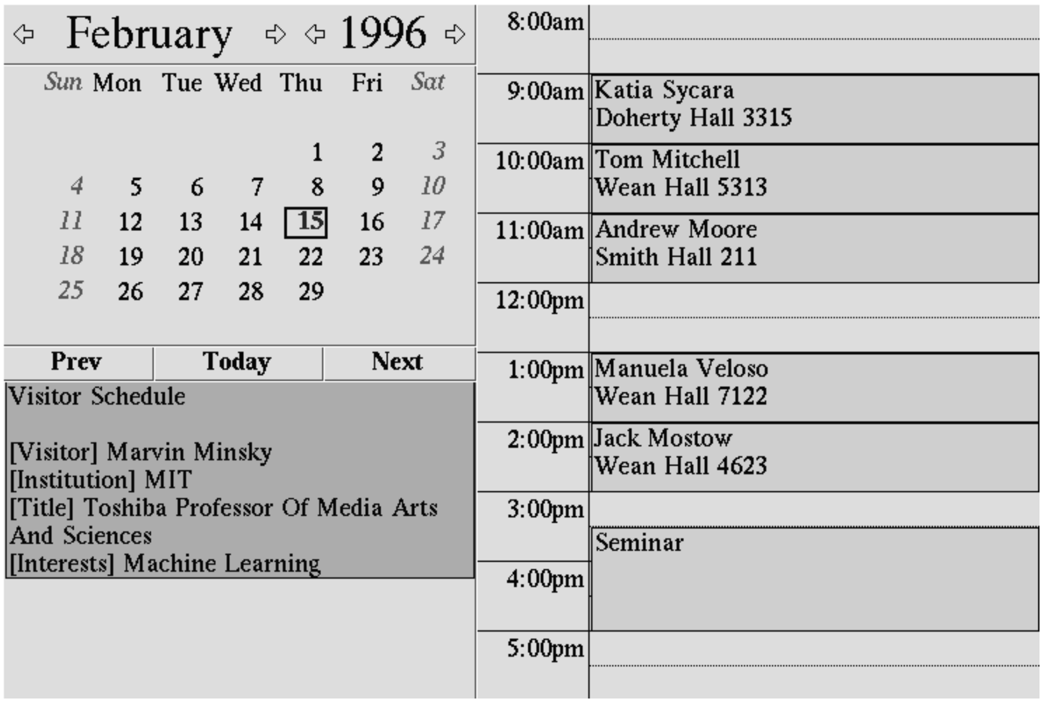


Figure 4: Final Schedule of Minsky's Visit

asking for user advice and advising the user of the state of the system and its progress.

5.2 Financial Portfolio Management

The second domain of applying the RETSINA framework is financial portfolio management (the WARREN system [3]). In current practice, portfolio management is carried out by investment houses that employ teams of specialists for finding, filtering and evaluating relevant information. Based on their evaluation and on predictions of the economic future, the specialists make suggestions about buying or selling various financial instruments, such as stocks, bonds, mutual funds etc. Current practice as well as software engineering considerations motivate our multi-agent system architecture. A multi-agent system approach is natural for portfolio management because of the multiplicity of information sources and the different expertise that must be brought to bear to produce a good recommendation (e.g. a stock buy or sell decision).

The overall portfolio management task has several component tasks. These include eliciting (or learning) user profile information, collecting information on the user's initial portfolio position, and suggesting and monitoring a reallocation to meet the user's current profile and investment goals. As time passes, assets in the portfolio will no longer meet the user's needs (and these needs may also be changing as well). Our initial system focuses on the ongoing portfolio monitoring process.

We briefly describe the main agents in the portfolio management task, shown in figure 5:

The portfolio manager agent is an interface agent that interacts graphically and textually with the user to acquire information about the user's profile and goals. *The fundamental analysis agent* is a task assistant that acquires and interprets information about a stock from the viewpoint of a stock's (fundamental) "value". Calculating fundamental value takes into consideration information such as a company's finances, forecasts of sales, earnings, expansion plans etc. The *Technical Analysis agent* uses numerical techniques such as moving averages, curve fitting, complex stochastic models,

[3]The system is named after Warren Buffet, a famous American investor and author about investment strategies.

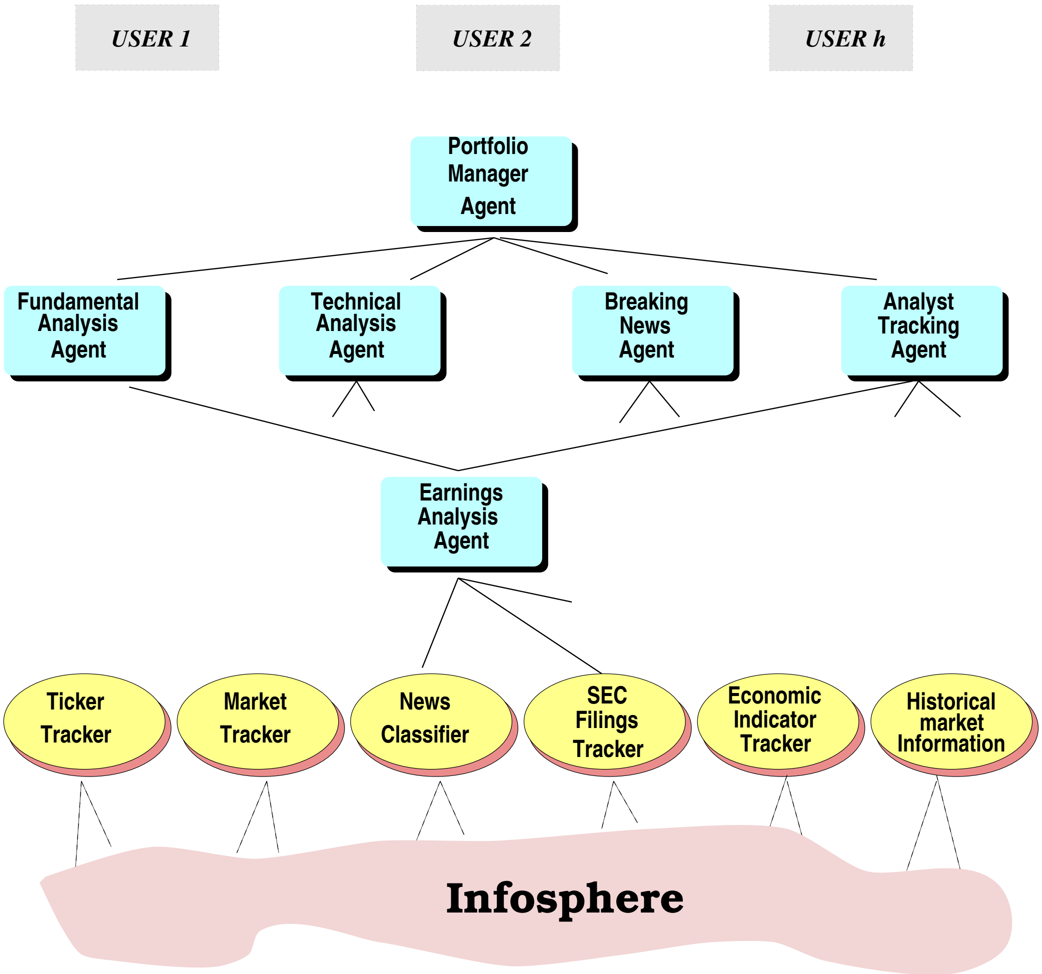


Figure 5: The Portfolio Management Application

neural nets etc., to try to predict the near future in the stock market. The *Breaking News agent* tracks and filters news stories and decides if they are so important that the user needs to know about them immediately, in that the stock price may be immediately affected. The *Analyst Tracking agent* tries to gather intelligence about what human analysts are thinking about a company. These agents gather information through information requests to information agents. The information agents that we have currently implemented are the *Stock Tracker* agents that monitors stock reporting Internet sources, such as the Security APL, the *News Tracking* agents that track and filter Usenet relevant financial news articles (including CMU's Clarinet and Dow Jones news feeds), and the SEC (Securities and Exchange Commission) fillings of companies financial information tracker agent that monitors the EDGAR database. The information retrieved by these information agents is passed to the display agents which display in an integrated fashion the retrieved information to the user.

Figure 6 shows an example WARREN portfolio. Currently, a user may interact with his or her own portfolio display (interface) agent via HTML forms and a web browser.[4] The portfolio display consists of a summary of the user's portfolio, including which issues are owned, and for each issue the total number of shares owned, the current price, the date of the last news article, and the current value. Below the portfolio table, the current value of the entire portfolio is displayed along with the portfolio's net gain in equity (current values compared to purchase values minus commissions). The interface also allows the user to buy and sell stocks (Trade) and to request the preparation of a Financial Data Summary (Fetch FDS), which uses historical price, earnings, and revenue information from the SEC's EDGAR database to do a simple fundamental analysis of the stock.

The other display available to the user (by clicking on a stock's current value) is a price/news graph that dynamically integrates intra-day trading prices and news stories about a stock. Figure 7 shows an example for Netscape Communications (NSCP) during the period of roughly December 5 to December 23. Prices are plotted at mostly 1 hour (sometimes 15 minute) intervals, and connected during the trading day (there's no trading at night or during the weekends). The numbers on the graphs correspond to the news articles whose subjects are listed below the graph. The articles are numbered

[4]We are currently constructing a more interactive Java interface.

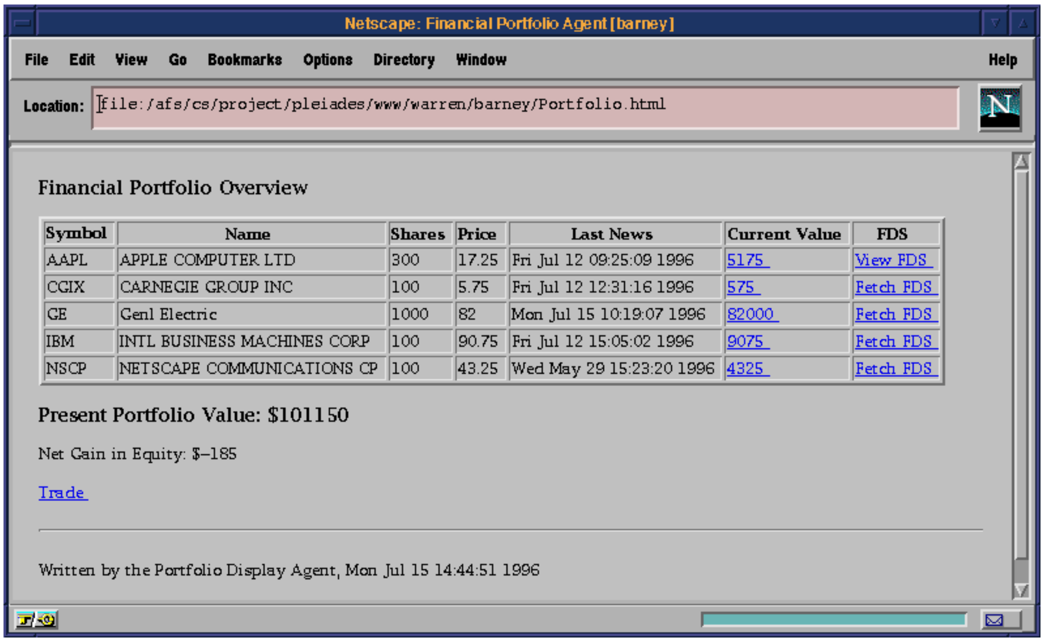


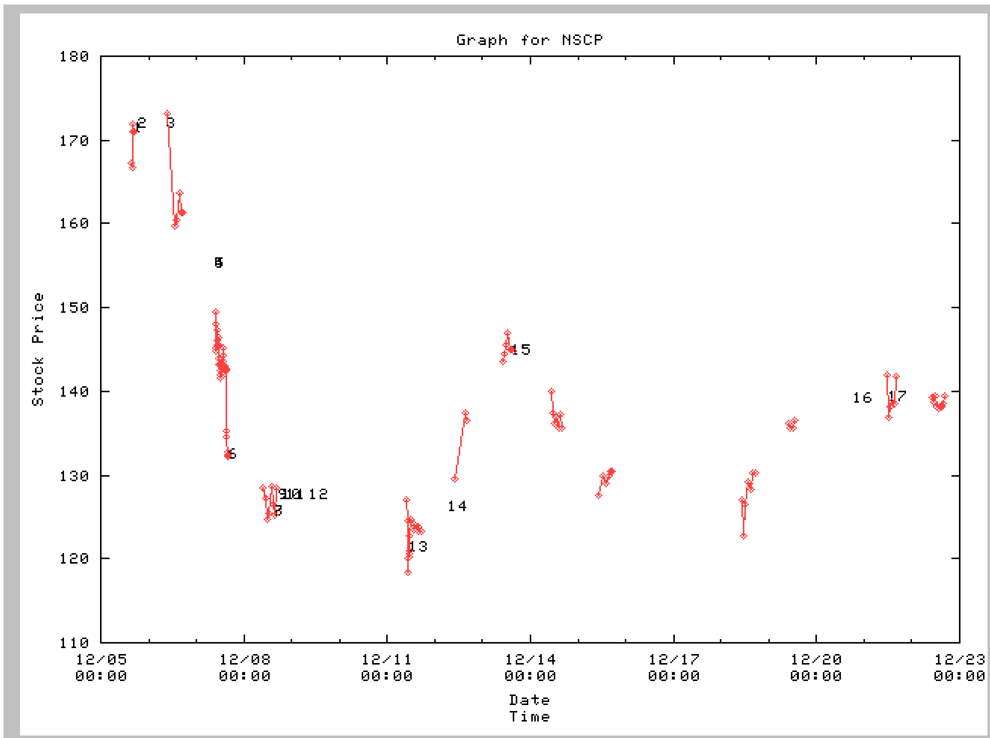
Figure 6: WARREN's Netscape interface.

from earliest to latest (left to right on the graph). Each article number is positioned at the time the news story appeared, and vertically at the approximate price of the stock at that time. The article subjects are hyperlinked to the actual news stories themselves.

The example graph covers a time period just after the $30 price rise in NSCP triggered by the joint Sun and Netscape announcement of JavaScript (2). However, the new record high caused some profit-taking, and then the Dec 7 news hits that Smith Barney had begun coverage of Netscape and recommended SELL (4,5), dropping the stock for the rest of the day. Although our University access is to delayed price and news sources, such information from realtime data feeds is the bread and butter of many types of institutional investment decision-making.

6 Conclusions

In this paper, we have described our implemented, distributed agent framework, RETSINA, 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, including (1) *Interface agents* which interact with the user by receiving user specifications and delivering results, (2) *Task agents* which help users perform tasks by formulating problem solving plans and carrying out these plans through querying and exchanging information with other software agents, and (3) *Information agents* which provide intelligent access to a heterogeneous collection of information sources. 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 vast, new, 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.



Matching News Articles for NSCP

- [1] ****Silicon Graphics/Sun/Netscape To Merge Web Technologies 12/05/95 [Tue Dec 5 16:00:41 1995]
- [2] Netscape Stock Surges on JavaScript Plans [Tue Dec 5 18:00:16 1995]
- [3] Sun Microsystems Unit To Offer Netscape Pdts >SUNW NSCP [Wed Dec 6 08:31:13 1995]
- [4] *Smith Barney Starts Netscape Communications At Sell >NSCP [Thu Dec 7 08:55:38 1995]
- [5] Smith Barney–Internet Stks –2: Initiates 6 Cos.>AMER NSCP [Thu Dec 7 09:23:22 1995]
- [6] CA Announces Internet Products, Partnership With Netscape 12/07/95 [Thu Dec 7 16:00:35 1995]
- [7] Netscape, Other Internet Shares Pressured [Fri Dec 8 15:00:20 1995]
- [8] Netscape vows "dog–fight" vs Microsoft [Fri Dec 8 15:00:32 1995]
- [9] Netscape, other Internet shares pressured [Fri Dec 8 16:50:08 1995]
- [10] Netscape Vows "Dog–Fight" vs Microsoft [Fri Dec 8 18:20:26 1995]
- [11] Netscape Gives 2 $1,000 Apiece [Fri Dec 8 20:30:24 1995]
- [12] Netscape, Other Internet Shares Pressured [Sat Dec 9 07:50:12 1995]
- [13] Hot Stocks In Barron's: CPQ TBCC ITW NSCP MSFT IBM T [Mon Dec 11 10:17:31 1995]
- [14] H–P, Microsoft, Netscape Sign Pact To Develop Web Standards [Tue Dec 12 06:05:22 1995]
- [15] ****HP Teams With Netscape & Microsoft On Internet Printing 12/12/95 [Wed Dec 13 14:20:32 1995]
- [16] ****Netscape Creates Java Conference 12/20/95 [Wed Dec 20 18:11:05 1995]
- [17] Worldview Sys To Use Netscape Software To Create Travel Svc [Thu Dec 21 11:33:43 1995]

Written by the Stock Display Agent, Fri Dec 22 17:03:01 1995

Figure 7: A Price/News graph constructed by the WARREN system for Netscape Communications (ticker symbol NSCP).

7 Acknowledgements

The current research has been sponsored in part by ARPA Grant #F33615-93-1-1330, by ONR Grant #N00014-95-1-1092, and by NSF Grant #IRI-9508191. We want to thank Tom Mitchell, Dana Freitag, Sean Slittery, and David Zabowski for insightful discussions.

References

[1] P. R. Cohen and H. J. Levesque. Intention=choice + commitment. In *Proceedings of AAAI-87*, pages 410–415, Seattle, WA., 1987. AAAI.

[2] K. Decker, K. Sycara, and M. Williamson. Modeling information agents: Advertisements, organizational roles, and dynamic behavior. In *Proceedings of the AAAI-96 Workshop on Agent Modeling*, Portland, Oregon, August 1996. AAAI.

[3] K. Decker, M. Williamson, and K. Sycara. Matchmaking and brokering. In *Proceedings of the Second International Conference in Multi-Agent Systems (ICMAS'96)*, Kyoto, Japan, December 1996.

[4] Keith Decker. *Environment Centered Analysis and Design of Coordination Mechanisms*. PhD thesis, University of Massachusetts, 1995.

[5] 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.

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

[7] 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.

[8] Leonardo Garrido and Katia Sycara. Multi-agent meeting scheduling: Preliminary experimental results. In *Proceedings of the Second International Conference in Multi-Agent Systems (ICMAS'96)*, Kyoto, Japan, December 1996.

[9] M. R. Genesereth and S. P. Katchpel. Software agents. *Communications of the ACM*, 37(7):48–53,147, 1994.

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

[11] Craig K. Knoblock. Integrating planning and execution for information gathering. In Craig Knoblock and Alon Levy, editors, *Working Notes of the AAAI Spring Symposium Series on Information Gathering from Distributed, Heterogeneous Environments*, Stanford, CA, March 1995. AAAI.

[12] D. Kuokka and L. Harada. On using KQML for matchmaking. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 239–245, San Francisco, June 1995. AAAI Press.

[13] Kan Lang. Newsweeder: Learning to filter netnews. In *Proceedings of Machine Learning Conference*, 1995.

[14] 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.

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

[16] 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.

[17] 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.

[18] Anandeep Pannu and Katia Sycara. Learning text filtering preferences. In *1996 AAAI Symposium on Machine Learning and Information Access*, 1996.

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

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

[21] Reid Simmons. Structured control for autonomous robots. *IEEE Journal of Robotics and Automation*, 1994.

[22] 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.

[23] Katia Sycara and Dajun Zeng. Coordination of multiple intelligent software agents. *International Journal of Cooperative Information Systems*, To Appear, 1996.

[24] M. Williamson, K. Decker, and K. Sycara. Unified information and control flow. In *Proceedings of the AAAI-96 Workshop on Theories of Action, Planning and Control: Bridging the Gap*, Portland, Oregon, August 1996. AAAI.

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

[26] Dajun Zeng and Katia Sycara. Bayesian learning in negotiation. In *1996 AAAI Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems*, 1996.