International Journal of Cooperative Information Systems

COORDINATION OF MULTIPLE INTELLIGENT SOFTWARE AGENTS

KATIA SYCARA*
The Robotics Institute, Carnegie Mellon University
Pittsburgh, PA 15213, United States of America

and

DAJUN ZENG†
The Robotics Institute, Carnegie Mellon University
Pittsburgh, PA 15213, United States of America

Received
Revised

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, as well as anticipate 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 structuring agents. The system architecture 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, investment counseling, health care, and electronic commerce.

Keywords: Software Agent; Distributed AI; Coordination; Information Agent; Internet Applications

*Email Address: katia@cs.cmu.edu, Web Homepage: http://www.cs.cmu.edu/~sycara/
†Email Address: dajun.zeng@cs.cmu.edu, Web Homepage: http://www.cs.cmu.edu/~zeng/

1. Introduction

Current networking technology and the ready availability of vast amounts of data and information on the Internet-based Infosphere present great opportunities for bringing to decision makers and decision support systems more abundant and accurate information. 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 has been proposed to address this challenge[1,2,3,4,5,6]. 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, resolve inconsistencies in the retrieved information, filter away irrelevant or unwanted information, integrate information from heterogeneous information sources, 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[7,8]. Another type of agent is the *SoftBot* [9], 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 [6,10]. 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, 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 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 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 such multiple Intelligent Agents. We will focus on three crucial characteristics of our architecture that differentiate our work from others: (1) *multi-agent* system where the agents operate asynchronously and collaborate with each other and their users, (2) the agents *actively* seek out information, and (3) 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), Financial Portfolio Management (the WARREN INC system), Emergency Medical Care (the HIN system), and Electronic Commerce.

The rest of the paper is organized as follows. Section 2 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. Section 3 presents in detail the reusable agent architecture and discusses planning, control, and execution monitoring in agent operations. Section 4 presents related work. The application of our implemented framework in various domains is described in Section 5. In this section, a detailed scenario of everyday organizational decision making is given to illustrate the interactions among software agents. Section 6 presents concluding

remarks.

2. Distributed Architecture for Intelligent Information Processing and Problem Solving

In this section, we motivate and describe the distributed architecture and coordination mechanisms of the coordinating agents for intelligent information retrieval and problem solving. This distributed architecture 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 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 existing information sources have been developed 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 more preferable way of updating is to construct agent wrappers around existing systems. These agent wrappers interface the information sources and information consumers and provide a uniform way of accessing the data as well as offer additional functionalities such as monitoring 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 user 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 deemphasized 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.

There have been several proposals in the literature for generic agent frameworks (see Section 4). These architectures typically have not been tested in truly multi-agent environments. They provide very high level design guidance that is not very informative for structuring real time systems that operate efficiently for real complex tasks involving coordination, information gathering, and user interaction. This is the major reason why we have decided to differentiate agents in our system architecture in terms of their functionality and communications needs. In particular, we distinguish three different types of agents, *interface agents*, *task agents*, and

information agents. The architecture of all these agents follows the general BDI type philosophy [2], however, each of them embodies particular architectural design commitments to make them effective in dealing with the particular category of issues of its type. We believe that it is easier and more productive to develop, implement and test in real information environments, reusable agent architectures for each of these agent categories. As our point of departure, we use the Task Control Architecture (TCA) framework [11] 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 the characteristics of the different types of agents.

2.1. Agent Types

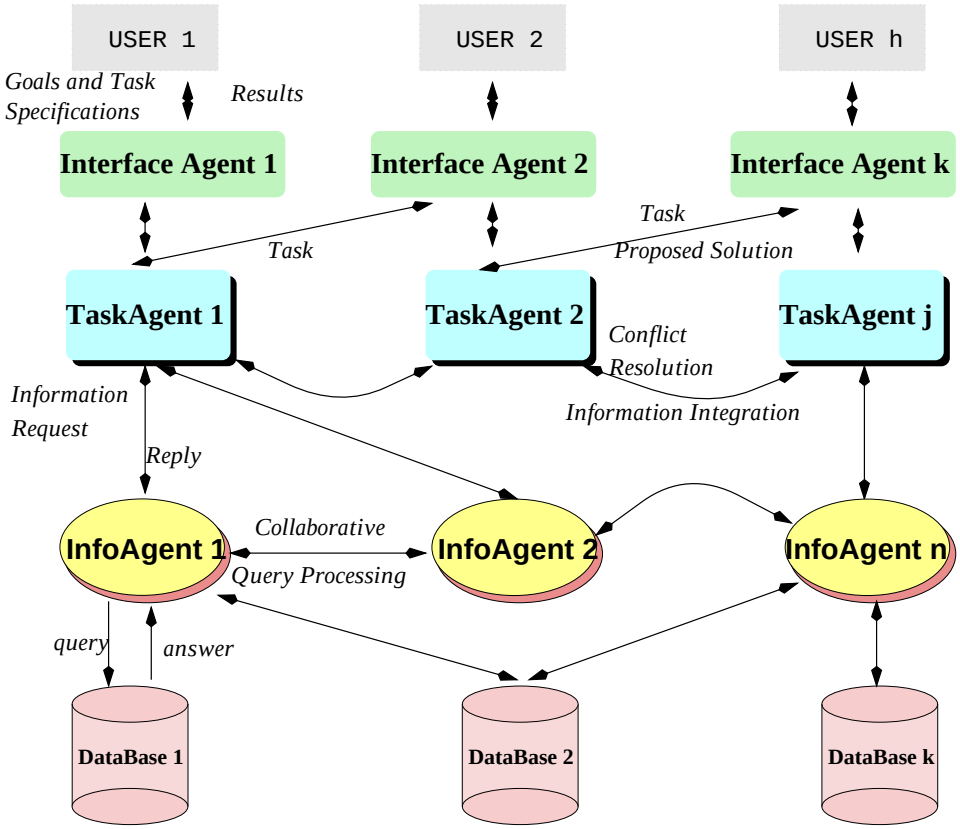

Figure 1: Distributed System Architecture

Our design decision to differentiate interface agents, task-specific agents, and information-specific agents (See Figure 1) was based on the following observations:

1. In real world applications, there seems to be a natural distinction between information sources and user delegated decision making components. In a multi-agent architecture, delegating different types of responsibilities, namely, information accessing and user task-oriented decision making, to different types of software agents tasks is natural. This differentiation also aids user intuition in understanding the functioning of the system.

2. From the software engineering point of view, because there are many interactions between information sources and information consumers, assigning separate software components to manage information sources and to handle user-specific and task-specific decision making, respectively, can help enhance software modularity and reuseability. This becomes more obvious in situations where a certain information source is accessed by many information consumers. Having one (or a small set of) information-specific agent responsible for that information source provides a more sensible engineering solution than building the capability of accessing and managing the information source into every possible consumer agent.

3. One of the primary usages of information-specific agents is to make legacy information sources available to the society of intelligent agents. Backward compatibility in terms of taking advantage of data available in legacy data sources is critical for the success of the agent-based approach. Differentiating between interface agents, information-specific and task-specific agents offers an abstraction tool enabling an incremental style of system building in which system builders may want to focus on developing information-specific agents first and then start to utilize the available information before implementing other advanced decision-making components.

The crucial factors influencing the determination of the type of an agent are: (1) what are the functional and informational scopes of these types of agents in a distributed architecture, (2) what kinds of interactions, coordination and activation are predominant among these agents, and (3) what reusable agent components can constitute agent structuring and what functionalities these components will support.

In the following subsections, we explore in detail the different design commitments with respect to interface agents, task agents and information agents along various dimensions, such as their functionality, knowledge, coordination, interactions, and reusable agent architecture.

2.1.1. Agent Functionality

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, interacting only through a relevant interface agent for a task hides the

underlying distributed information gathering and problem solving complexity. 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.

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 information-specific agent primarily provides intelligent information services. The simpler of these services is one shot retrieval of information in response to a query; a more enhanced information service is constant monitoring of available information sources for the occurrence of predefined information patterns (e.g. addition of a new record in a data base). Traditional Data Base Management Systems (DBMS) can be considered very simple forms of information agents. A more advanced form could be a DBMS with communication capabilities, using KQML, for example. An even more advanced information agent can, in addition to communication with other agents, monitor its data base for the appearance of particular patterns. The capability that makes a system a full fledged information agent is its additional ability to locate an information source and retrieve information (or monitor for information patterns) from information sources that are *detached* from it, i.e., the agent is used as an AI-enhanced gateway or wrapper to externally available information sources. A related capability is locating other agents that can perform some advertised service, i.e., provide *yellow page services*. Such agents have been called facilitators by Genesereth[12]. In our system, the notion of information agent subsumes facilitators. By viewing facilitators as information agents, an interesting new capability, not having been mentioned in the intelligent agents literature so far is, *pro-active monitoring for the appearance of new agents* with particular capabilities.

An advanced 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 [13,14].

2.1.2. Knowledge held by the Agents

In order to realize their functionality, the agents need pertinent knowledge in their belief data base and algorithms that utilize the knowledge. In Section 3, we will describe reusable agent components that store and process the knowledge.

An interface agent has the following knowledge: (1) a model of the user's goals and preferences pertaining to a task, (2) knowledge of the relevant task assistants

that can perform the task, (3) knowledge of what must be displayed to the user and in what way, (4) protocols for interacting with relevant task assistants. User models and preferences could be automatically acquired [15,16].

A task agent has the following knowledge: (1) model of the task domain, (2) knowledge for performing the task (e.g. query decomposition, sequencing of task steps), (3) information gathering needs associated with the task model, (4) knowledge about relevant task- or information-specific agents that it must coordinate with in support of its particular task, (5) protocols that enable coordination with the other relevant agents, and (6) strategies for conflict resolution and information fusion.

A typical *information-specific* agent knows: (1) model and associated meta-level information of the data bases that it is associated with, such as size, average time it takes to answer a query and monetary cost of query processing, (2) procedures for accessing databases, (3) conflict resolution and information fusion strategies, and (4) protocols for coordination with other relevant software agents.

2.2. Agent Organization, Coordination and Interactions

In our distributed intelligent agent organization 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 (for example, while monitoring currently held investments during stable market 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 assistants that each agent communicates with.
- The information assistants are responsible for recognizing important information, information filtering, and checking information quality.
- The task assistants are responsible for resolving information conflicts and integrating information from heterogeneous information sources for their respective tasks.

- The task assistants are responsible for activating relevant information assistants 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 in our multi-agent system (For an extended illustrative scenario, see Section 5.1.1). 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 finally reporting to the interface agent which conveys the results to the user. 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. The plans that task-specific agents have (see Section 3) include information gathering goals, which, in turn are satisfied through relevant plans for information retrieval. A task-specific agent decomposes an information goal into subgoals that are delegated to information agents. The information agents form plans to achieve these goals, and proceed to execute and monitor them. A task-specific agent can decide when to actively seek new information and, in turn, utilize retrieved information for problem solving. This type of intelligent agent differs from traditional AI systems since information-seeking during problem solving is an inherently built-in 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-assistant 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.

An information agent works differently. Its 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, rather than just waiting for and servicing one-shot information queries.

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 [17] 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 and handles different communication channels, different data and query formats, etc.

We have also implemented an *advertisement* mechanism and *services registries* that can be accessed by task-specific agents to help determine availability and location of desired information and services.

3. Agent Engineering: How To Structure An Agent?

As our point of departure in structuring an agent, we use the *Task Control Architecture* (TCA) [18] which we extend and specialize for real-time user interaction, information gathering, and decision support tasks in the Infosphere. The control constructs available in TCA are used to integrate, coordinate, and monitor planning and plan execution, and to incrementally improve the efficiency and robustness of the multi-agent information system. These control constructs are part of the reusable agent architecture. 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 planning module of the task agents can be a full-fledged planner, whereas the planning module of the interface agents and the information agents is much simpler consisting of retrieval and instantiation of plan templates. In our initial implementation, the information agent planning component is a simple plan retrieval mechanism that instantiates a new task structure for each goal. Thus it is extremely fast but lacks flexibility. Every plan step has an (optional) execution deadline.

The key component of this architecture is a hierarchical representation of task-/subtask relationships[11], on which we rely heavily in our information software agent architecture. This representation, called a *task tree*, has goals as non-terminal nodes, and executable actions and execution monitoring mechanisms at the leaves. Tem-

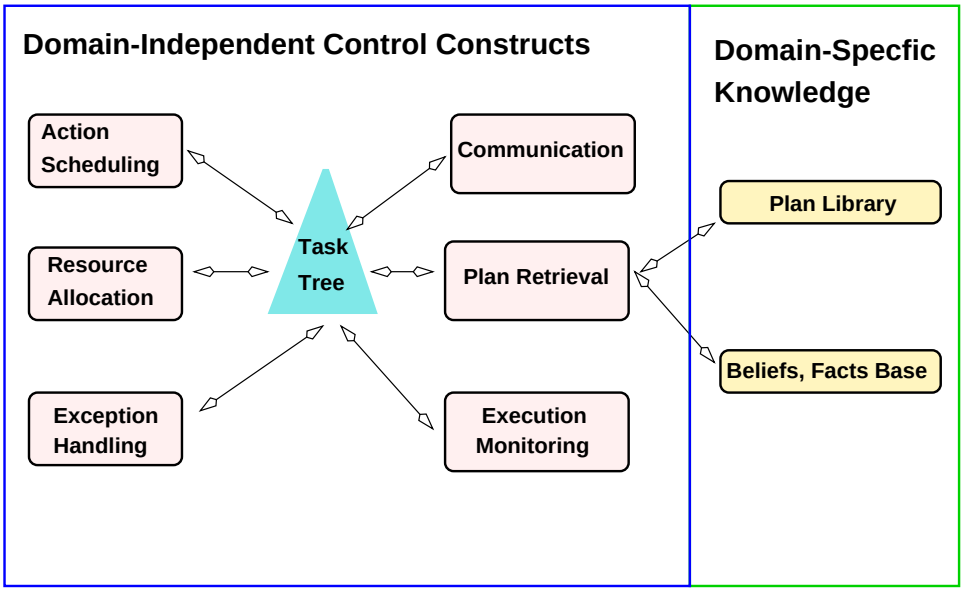

Figure 2: Agent Architecture

poral 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 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.

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 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—e.g. 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. Mechanisms for reactivity in the agent architecture provide a systematic and reusable way of engineering these uncertainty handling mechanism into software agents. A simple example is a timeout mechanism. 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 [19].

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 takes as input a query and returns as output a set of complete records. The local database internal to an information agent provides efficient query processing and retrieval and can be reused for any information source. This allows the external query interface to be very 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 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, finding 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.

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

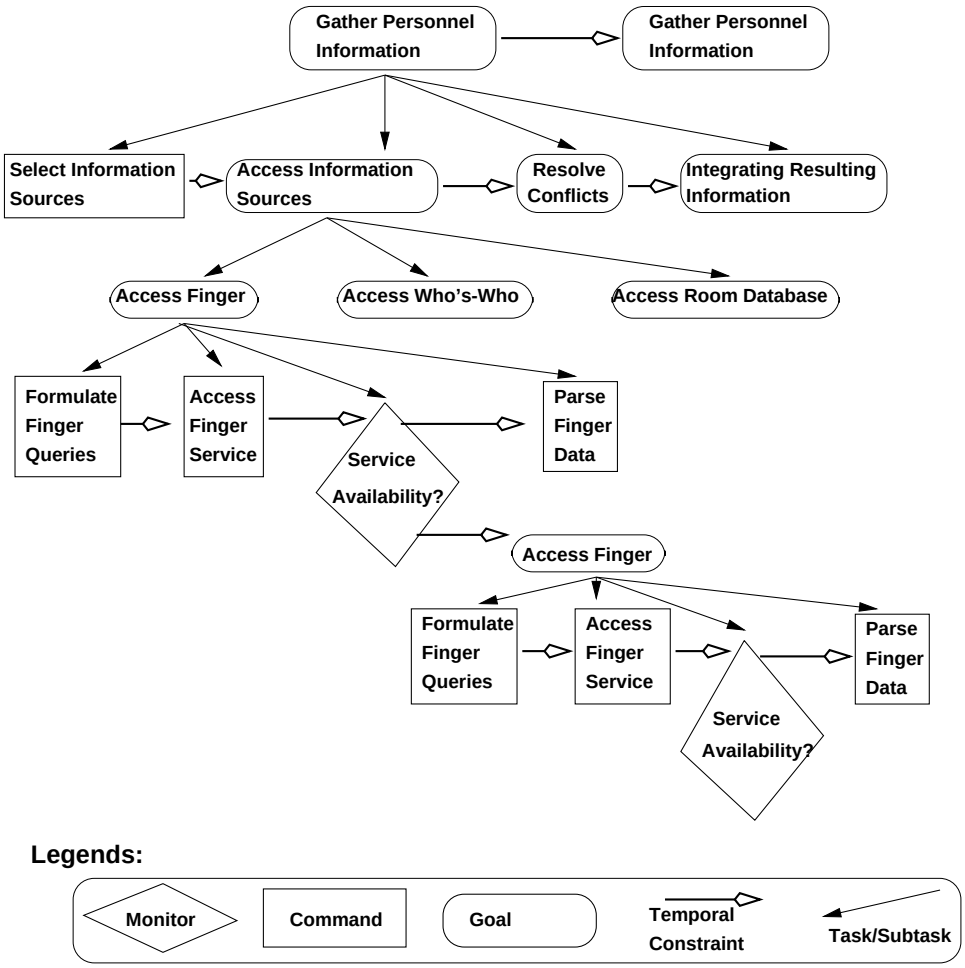

Figure 3: Task Tree for *Personnel Finder*

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. For example, it is possible to abort the "Access Finger" if not completed in, for example, five minutes or to stop accessing "Access Who's-Who" after three tries.

4. Related Work

Recent work in Distributed AI has demonstrated the usefulness of DAI techniques for cooperative plan recognition among geographically distributed sensor agents [20], distributed decision making by a team of specialists [21] and distributed search [22,23]. Additional DAI research includes problem decomposition and task allocation to a network of agents [24], distributed control [25], distributed planning [20], agent orga-

nization [26], and negotiation [27,28]. More recently, the currently small but rapidly growing research community on Intelligent Agents has to-date focused on interesting and important issues, such as interfacing these agents with users [15,5], issues of agent trustworthiness [7], user acceptability [29], characteristics of agenthood [9], agent architecture and organization [30,12], and information retrieval from Internet-based resources [9,31,32]. Different control architectures have also been explored for intelligent software agents. For example, a centralized control architecture has been developed for performing useful UNIX file management tasks [33]. Although these so-called "SoftBot", or "UserBot", or "Personal Assistant" share many characteristics with our task-specific agents, our underlying infrastructure is fundamentally different from theirs since we assume and take advantage of a distributive and collaborative operating environment.

The BDI (belief-desire-intention) architecture developed by Rao and Georgeff [34,2] is a logical framework for agent theory based on a branching model of time. They also implemented a system based on the BDI architecture, called the Procedural Reasoning System (PRS) [35]. Shoham proposed agent-oriented programming paradigm and developed a prototypic language AGENT0 [3]. Pollack *et. al.* addressed the issue of bridging the gap between formalizations of BDI architectures and system implementations [36]. Brazier *et. al.* developed a framework called DESIRE [37] in which they modeled task hierarchy, allocation, and control along the same lines as our work. Jennings *et. al.* discussed the issues with development of intelligent multi-agent systems in real world settings [38]. Our work differs from the above-mentioned agent frameworks mainly in two aspects: (1) our system is aimed at truly multi-agent environments reflected by the fact that the agents in our system model other relevant agents explicitly, and (2) we instantiate the general agent architecture by differentiating between *interface*, *task*, and *information* agents for structuring real time systems that operate efficiently for real complex tasks involving coordination, information gathering and user interaction.

5. Application Domains

We have implemented distributed cooperating intelligent agents using the concepts, architecture, and reusable components we have described in a variety of application domains: everyday organizational decision making, financial portfolio management, emergency health care and electronic commerce. In this section, we describe in detail the first application, illustrate the functioning of the distributed, cooperative Intelligent Agents through an extended scenario and give brief descriptions of the remaining application domains.

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 [7], calendar management [15], and filtering news [5]. 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 our distributed agent architecture to such multi user tasks of increased complexity, such as distributed, collaborative meeting scheduling among multiple human attendees [39], finding people information on the Internet, hosting a visitor to Carnegie Mellon University [6], 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 [13].

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 been explored by Kautz and his colleagues at Bell Labs [30].

Our system consisting of a collection of agents, collectively referred to as *Visitor Hosting system* (see Figure 4), supports the visitor hosting task. Carnegie Mellon University has many requests for visits every year from academic, industrial and government personnel. Currently, an administrative staff person is responsible for receiving the visit requests and creating the visitor's schedule in coordination with CMU faculty and students that match the visitor's area of interest, and are willing and free to meet with him/her. Even in today's world of electronic mail, making arrangements for a person's visit is a time consuming task and would benefit from support by intelligent agents technology.

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 meeting 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. The Visitor Hoster task agent forms plans for achieving the visitor hosting goal and coordinates with appropriate software agents for plan execution, monitoring, and results composition. The system 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,

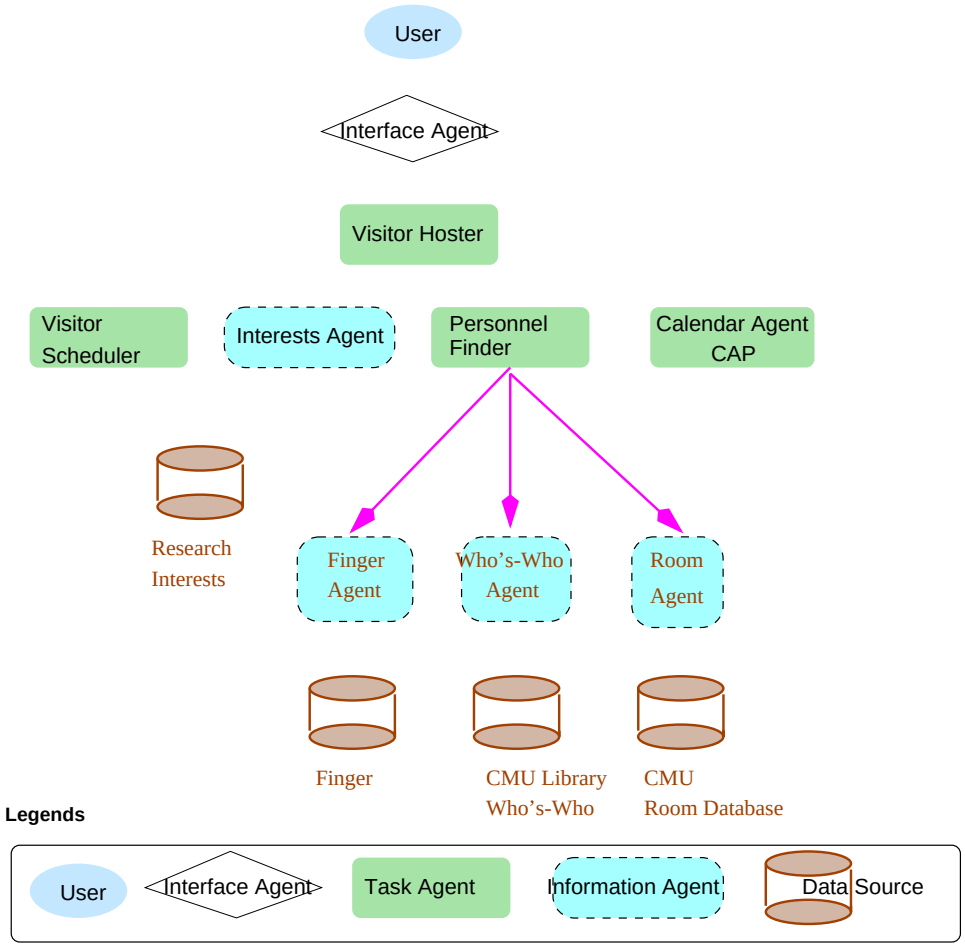

Figure 4: Visitor Hosting System

the Visitor Hosting system has a number of information agents that (1) retrieve information from a CMU data base that has faculty research interests (Research 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 Interface agent for the visitor hosting task.

  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 Interface agent.

- The communication module of the Interface agent processes the visitor request and extracts the visitor's areas of interest, name, and organization and sends them to the Visitor Hoster task agent.

- The planner module of the Visitor Hoster agent formulates a goal ("meet with faculty of same research interests"), and instantiates it with "research-interests = machine learning". This goal has the two subgoals, "collect contact information" and "produce visitor's schedule". The plan operators for collecting contact information are "access-X?" where X? corresponds to known information agents or task agents. The preconditions of "access-X?" are the input required for accessing agent X? and postconditions are retrievable information and results that the contacted agent can generate. The planner module produces a plan for "collect contact information" through means ends analysis and executes it.[a] The plan steps are "access Interests-Agent", followed by "access Personnel-Finder". The Visitor Hoster scheduling module schedules the first step for execution. In executing the first plan step, the communication module formulates a KQML message to the Interests agent conveying the visitor's areas of interest and asks it to find faculty members whose interest areas match the request.

```
(ask
     :language simple-query
     :ontology research-interest
     :receiver Interests-agent
     :sender Visitor-Hoster-agent
```

[a] The results of planning, e.g. the information gathering plans, can be cached in an agent's plan library and reused. This results in greater computational efficiency but is not flexible enough to handle all possible situations.

```
      :in-reply-to visitor-hoster-query-interest-agent-103
      :content (query
                  (research-interest "machine learning")))
```

- The Interests agent queries the faculty interests data base and returns names of CMU faculty whose research matches "machine-learning".

  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, shown in the following message:

```
(tell
   :language simple-query
   :ontology research-interest
   :receiver Visitor-Hoster-agent
   :sender Interests-agent
   :in-reply-to visitor-hoster-query-interest-agent-103
   :content (respond
              (
               ((first-name "Tom") (last-name "Mitchell"))
               ((first-name "Andrew ") (last-name "Moore"))
               ((first-name "Jack ") (last-name "Mostow"))
               ((first-name "Herbert") (last-name "Simon"))
               ((first-name "Katia ") (last-name "Sycara"))
               ((first-name "Manuela") (last-name "Veloso"))
               ...
               )))
```

- After the faculty names have been communicated to the Visitor Hoster by the Interests agent, the execution of the first plan step is complete and the Visitor Hoster executes the second step of the "collect contact information" plan, namely "access Personnel-Finder".

  For example, to find information about Tom Mitchell, the Visitor Hoster agent sends the following message to the Personnel Finder agent:

```
(ask
      :language simple-query
      :ontology personnel-information
      :receiver Personnel-Finder-agent
      :sender Visitor-Hoster-agent
      :in-reply-to visitor-hoster-personnel-finder-agent-201
      :content (query
                  (first-name "Tom")
                  (last-name "Mitchell")
```

```
(organization "CMU")
(organization-type "EDU")))
```

- The communication module of the Personnel Finder processes this message and formulates the planning goal "gather personnel information" that gets planned for and executed as shown in Figure 3. The Personnel Finder 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.

  In order to find the personnel information about Tom Mitchell at CMU, the Personnel Finder agent spawns multiple queries collecting information from various information sources in parallel. After responses from these sources get collected, the Personnel Finder agent tries to resolve conflicts.

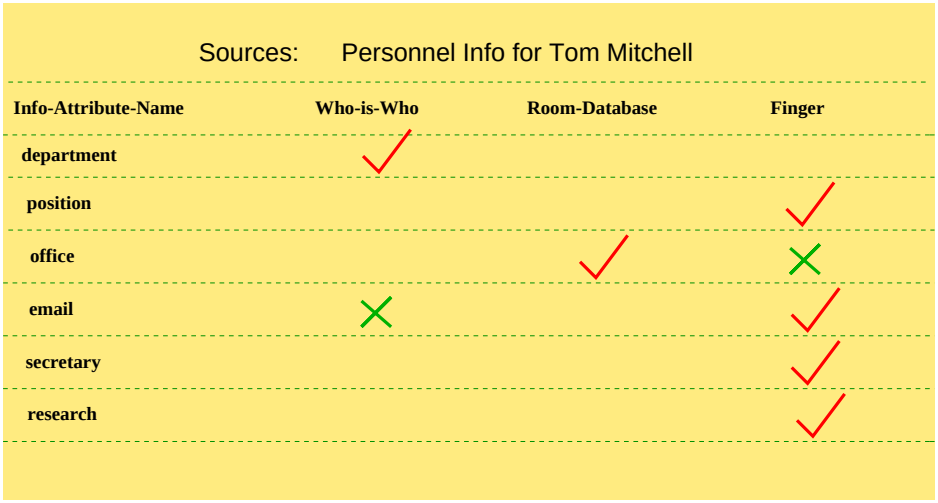

Figure 5: Information Sources and Returned Items

Figure 5 shows in detail the information sources used for querying personnel information about Tom Mitchell 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 checker and cross marks indicate which information sources return answers for which attributes. From Figure 5, 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. 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). In this case, the value returned by Room Database is considered as the correct office information, indicated in the figure by a checker mark in the "Office" row and "SCS-ROOM" column.

- Now the Visitor Hoster expands the "produce visitor's schedule" subgoal. The first plan step for this subgoal is "contact meeting candidates". Based on the information returned by the Personnel Finder, the Visitor Hoster agent selects an initial set of faculty to be contacted.

- 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. A sample e-mail message is shown as follows:

```
To: tom.mitchell@cs.cmu.edu
Subject: Would you like to meet with Marvin Minsky?

A visit is being organized as follows:

[Visitor]      (marvin minsky)
[Institution]  MIT
[Date]         02/15/1996

Please reply to this message, indicating whether you would like
to meet individually for 60 minutes with Marvin Minsky during
his visit.

He is available during the following times:

[Available Meeting Times]  02/15/1996: 9:00-17:00

Would you like a meeting?

Thank you,
The Visitor Hoster agent
```

- The communication module of Mitchell's calendar management agent CAP[15] processes the received message. In order to decide whether Mithcell would be interested in meeting with a visitor, CAP's planner needs additional information about the visitor (e.g. rank in the organization, work title, etc.). CAP

knows that the Personnel Finder can provide such information, so it asks for this additional information.

To collect more information about Minsky, CAP contacts the Personnel Finder agent by sending the following KQML message which contains the search keywords such as the name and organization of the visitor:

```
(ask
    :language simple-query
    :ontology personnel-information
    :receiver Personnel-Finder-agent
    :sender CAP
    :in-reply-to CAP-personnel-finder-agent-188
    :content (query
                (first-name "Marvin")
                (last-name "Minsky")
                (organization "MIT")
                (organization-type "EDU")))
```

- The Personnel Finder agent accesses Internet resources to find more detailed information about the visitor.

  When the Personnel Finder agent receives the query KQML message from the CAP agent, the first thing it needs to decide is what information sources to contact based on the affiliated organization information. For instance, if the interested person is a CMU professor, there are the Finger service available plus two other databases which can be accessed only internally at CMU, including a CS Room Database and a CMU on-line library Who's-Who . If the interested person is associated with other institutes, only the Finger service is available. Since Minsky comes from MIT, the Personnel Finder accesses the Finger service and heuristically parses the returned information by `finger`. As a result, the following message is sent back to the Visitor Hoster agent as the response to the query:

```
(tell
  :language simple-query
  :ontology personnel-information
  :receiver CAP
  :sender Personnel-Finder-agent
  :in-reply-to CAP-personnel-finder-agent-188
  :content (respond
              (first-name "Marvin")
              (last-name "Minsky")
              (organization "MIT")
              (organization-type "EDU")
```

```
(email "MINSKY@MEDIA.MIT.EDU")
(phone  "(617) 253-5864")
(office  "E15-486")
(department "Elec Eng & Comp Sci")
(title
 "Toshiba Professor Of Media Arts And Sciences")
 ))
```

- Having received the information about the visitor from the Personnel Finder, CAP formulates a response to the Visitor Hoster.

  CAP tells the Visitor Hoster that Mitchell is willing to meet with Minsky 10am-11am.

- The Visitor Hoster agent collects responses from all calendar agents 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[39]. The final calendar is shown in Figure 6.

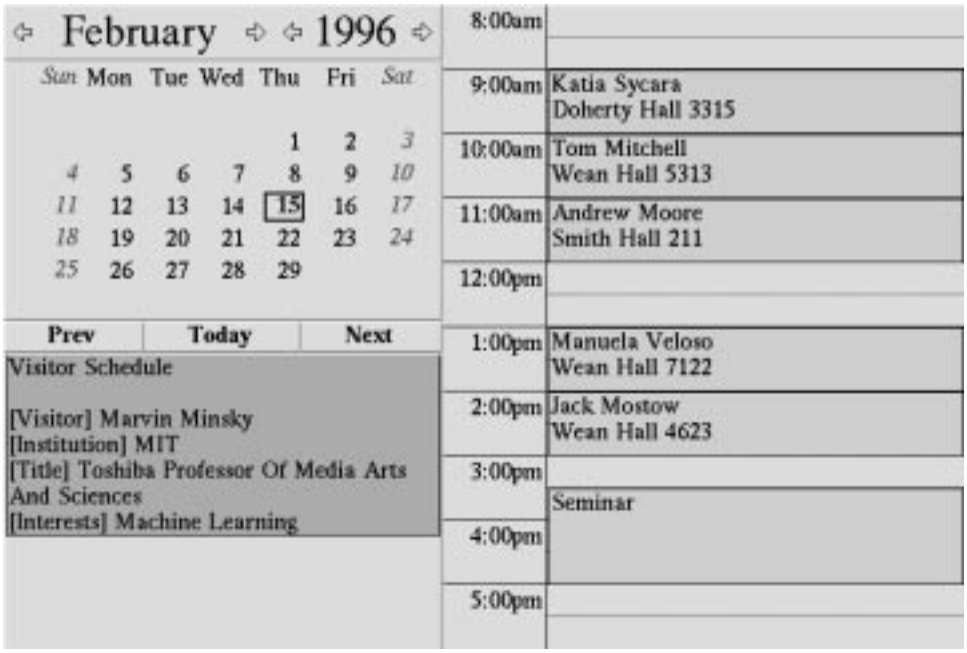

Figure 6: Final Schedule of Minsky's Visit

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

5.2. Financial Portfolio Management

We have recently started developing a multi agent system, called WARREN INC (Web Agents for Retrieval of Reliable Economic News for INvestment Counseling) [b]; for information gathering over Internet-based services in support of managing financial investments. 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. The overall task in the portfolio management domain, as stated by modern portfolio theory [40], is to provide the best possible rate of return for a specified level of risk, or conversely, to achieve a specified rate of return with the lowest possible risk. Risk tolerance is one of the features that characterize the user of our system; other features include the user's investment goals (long-term retirement savings? saving for a house?) and the user's tax situation.

Current practice as well as software engineering considerations motivate our multi agent system architecture. A multi-agent system approach is natural for portfolio monitoring because the multiple threads of control are a natural match for the distributed and ever-changing nature of the underlying sources of data and news that affect higher-level decision-making processes. A multi-agent system can more easily manage the detection and response to important time-critical information that could appear suddenly at any of a large number of different information sources. Finally, a multi-agent system provides a natural mapping of multiple types of expertise that need to be integrated and be brought to bear during any portfolio management decision-making.

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 re-allocation 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

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

as well). Our initial system focuses on the ongoing portfolio monitoring process.

We briefly describe the main agents in the portfolio management task. *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, neural nets, 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, and the *News Tracking* agents that track and filter Usenet relevant financial news articles (including CMU's Clarinet and Dow Jones news feeds). The information retrieved by these information agents is passed to the task assistants and used in making recommendations to the user about hold, buy or sell decisions.

5.3. Emergency Medical Care

Rapid and accurate access to patient information, in particular current and past medications that a patient is taking, as well as allergy information are very important for treatment in medical emergencies. This information is currently either completely unavailable to the emergency physician (e.g. a patient who lives in Pittsburgh has had a car accident in Los Angeles and is brought unconscious into an emergency room there), or is obtained very laboriously through getting in touch with the patient's primary physician and/or hospital facilities where the patient has been previously treated.

We are working with a consortium of companies to develop information services for Emergency Medical Care in both civilian and military settings. The consortium is developing (1) specifications and procedures for capturing data that is useful for emergency medicine, (2) specifications and implementation of schemes for storing the captured data in regional repositories, and (3) an overall system architecture that allows secure data access on a nationwide basis. Within the consortium, CMU is developing Intelligent Agents that

- interoperate with current hospital legacy systems over Wide Area Networks to gather, integrate and filter emergency medicine information to be brought at the right time to the right user (e.g. emergency physician, nurse)
- locate patient emergency data from regional data repositories by making effi-

cient nationwide searches, possibly with incomplete information (e.g. patient's health insurance card has been destroyed in an auto accident)

- respect security procedures and patient confidential information
- provide decision support to the healthcare provider
- monitor and notify the user of pertinent events (e.g. test results that were ordered in the emergency department are available for the physician's evaluation)

5.4. Electronic Commerce

In the area of electronic commerce, we are developing agents that buy and sell engineering products on the Internet. In this project, we are collaborating with IndustryNet, a company that provides on-line catalogs of engineering products on the Internet. Currently, there are 120,000 products represented on IndustryNet. IndustryNet [41] is a design and manufacturing service on the Internet that helps engineers locate product and service information. It represents a "live" working industrial MarketPlace where new electronic commerce ideas can be tested. The collection of agents that we are developing *locate* specified products in on-line catalogs, *select* the appropriate set of products that best meet the specifications and *negotiate* product acquisition and delivery.

Negotiating an agreement involves finding a compromise solution for multiple conflicting goals. This is a complicated problem, not amenable to traditional AI planning techniques [27]. The negotiation process itself is a search of a dynamic problem space where an agent's beliefs about another agent's beliefs over the cycle of proposals continuously changes the space being searched. What was *not* a solution at one point becomes a solution at a later point. In labor negotiations, for example, it is unlikely that either party would accept their eventual compromise, if it were presented at the inception of negotiations.

The process of negotiation starts with an expectation level and utility function for both parties of the negotiation. These expectation levels are relaxed through the process of negotiation. Agents will operate with criteria such as: (1) delivery time, (2) volume discounts, (3) promise of future business, (4) history of previous business, (5) early payment discounts, and (6) discounts for purchase of collateral goods. The rate at which relaxation is applied depends on each agent's utility on time to purchase and urgency to produce cash flow. We have developed an agent-to-agent protocol for making proposals and counter-proposals in order to reach a negotiated agreement.

In addition, we are using our information agent architecture to develop agents that monitor certain locations on the Internet and notify the originator when a change occurs that is of interest to the user. The changes of significance in electronic commerce of engineering products concern the appearance of new products of interest to the user. For example, if a designer is unable to find a part to suit

his stringent specifications, he may tell the monitoring agent to keep looking out over the Internet every other week. Currently, IndustryNet has "interests profiles" of what kinds of products its users are interested in. The monitoring agents can utilize these interests profiles to automatically monitor and notify users for the appearance of products that match their interests.

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, including (1) *Interface agents* which interact with the user 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 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.

We are applying the distributed agent architecture in a variety of complex real world domains, such as organizational decision making, investment counseling, health care and electronic commerce. We have briefly presented examples of intelligent agent use from these application domains. We are currently extending the capabilities of our agents to include (1) learning strategies for resolving information conflicts and (2) learning capabilities and reliability of inter-operating agents.

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. We want to thank Tom Mitchell, Dana Freitag, Sean Slittery, David Zabowski, Keith Decker, Anadeep Pannu, and other members of the PLEIADES project for interesting discussions. We also want to thank Gilad Amiri and Anandeep Pannu for doing much of the implementation.

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. 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.
3. Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.
4. M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.
5. Kan Lang. Newsweeder: Learning to filter netnews. In *Proceedings of Machine Learning Conference*, 1995.
6. Katia Sycara and Dajun Zeng. Towards an intelligent electronic secretary. In *Proceedings of the CIKM-94 Workshop on Intelligent Information Agents*, National Institute of Standards and Technology, Gaithersburg, Maryland, December 1994.
7. Pattie Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7), July 1994.
8. 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.
9. Oren Etzioni and Daniel Weld. A softbot-based interface to the internet. *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. Reid Simmons. Structured control for autonomous robots. *IEEE Journal of Robotics and Automation*, 1994.
12. M. R. Genesereth and S. P. Katchpel. Software agents. *Communications of the ACM*, 37(7):48–53,147, 1994.
13. Anandeep Pannu and Katia Sycara. Learning text filtering preferences. In *1996 AAAI Symposium on Machine Learning and Information Access*, 1996.
14. Dajun Zeng and Katia Sycara. Bayesian learning in negotiation. In *1996 AAAI Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems*, 1996.
15. 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.
16. Katia Sycara and Kazuo Miyashita. Case-based acquisition of user preferences for solution improvement in ill-structured domains. In *Proceedings of AAAI-94*, Seattle, Washington, August 1994. AAAI.
17. 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.
18. R. Simmons. A theory of debugging plans and interpretations. In *Proceedings of The seventh national conference of Artificial Intelligence*, St. Paul, Min., 1988.
19. Keith Decker and Katia Sycara. Designing reusable behaviors for information agents. Technical report, The Robotics Institute, Carnegie Mellon University, Pittsburgh, U.S.A., 1996.
20. E. H. Durfee. *A Unified Approach to Dynamic Coordination: Plannign Actions and Interactions in a Distributed Problem Solving Network*. PhD thesis, COINS, University of Massachusetts, Amherst, MA., 1987.
21. S. Lander and V. Lesser. Negotiation to resolve conflicts among design experts. In *Proceedings of the AAAI-88 Workshop on AI in Design*, St. Paul, MN., 1988. AAAI.

22. K. Sycara, S. Roth, N. Sadeh, and M. Fox. Distributed constrained heuristic search. *IEEE Transactions on System, Man and Cybernetics*, 21(6):1446–1461, 1991.
23. Sandip Sen and Edmund H. Durfee. The effects of search bias on flexibility in distributed scheduling. In *Proceedings of the Twelfth International Workshop on DAI*, 1993.
24. R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20:63–100, 1983.
25. Daniel D. Corkill and Victor R. Lesser. The use of meta-level control for coordination in a distributed problem solving network. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 748–755, Karlsruhe, Germany, August 1983.
26. T. Ishida, M. Yokoo, and L. Gasser. An organizational approach to adaptive production systems. In *Proceedings of AAAI-90, Boston, Mass.*, 1990.
27. K. Sycara. Negotiation planning: An AI approach. *European Journal of Operational Research*, 46:216–234, 1990.
28. Rosenschein J. and G. Zlotkin. *Rules of Encounter*. MIT Press, Cambridge, Mass., 1994.
29. Donald A. Norman. How might people interact with agents. *Communications of the ACM*, 37(7), July 1994.
30. Henry A. Kautz, Bart Selman, and Michael Coen. Bottom-up design of software agents. *Communications of the ACM*, 37(7), July 1994.
31. Y. Arens, C. Y. Chee, C.-N. Hsu, and C. A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal of Intelligent and Cooperative Information Systems*, 2(2):127–58, June 1993.
32. Robert Armstrong, Dayne Freitag, Thorsten Joachims, and Tom Mitchell. Webwatcher: A learning apprentice for the world wide web. 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.
33. O. Etzioni, K. Golden, and D. Weld. Tractable closed-world reasoning with updates. In *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning*, 1994.
34. A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In *Proceedings of Knowledge Representation and Reasoning*, pages 473–484, 1991.
35. Anand S. Rao and Michael P. Georgeff. An abstract architecture for rational agents. In *Proceedings of the Third Intenatinal Conference on Principles of Knowledge Representation and Reasoning (KR '92)*, pages 439–449, Cambridge, Massachusetts, October 23-29 1992.
36. M. E. Pollack, D. J. Israel, and M. E. Bratman. Towards an architecture for resource-bounded agents. Technical Report 425, SRI International, 1987.
37. F. Brazier, B. D. Keplicz, N. R. Jennings, and J. Treur. Formal specification of multi-agent systems: a real-world case. In *First International Conference on Multi-Agent Systems (ICMAS'95)*, pages 25–32, San Francisco, CA., June 12-14 1995.
38. N. R. Jennings, J. M. Corera, and I. Laresgoiti. Developing industrial multi-agent systems. In *First International Conference on Multi-Agent Systems (ICMAS'95)*, pages 423–430, San Francisco, CA., June 12-14 1995.
39. 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.
40. H. Markowitz. *Portfolio selection: efficient diversification of investments*. B. Blackwell, Cambridge, MA, second edition, 1991.
41. Donald Jones and D. Navin-Chandra. Industrynet: A model of commerce on the World Wide Web. *IEEE Expert*, pages 54–59, October 1995.