R. Patil, W. Zhang, W. Shen

USC / Information Sciences Institute, Marina del Rey CA, USA

Review Paper

An Information Mediator Network for Tasks in Dynamic Environments

Abstract: Coordination of activities among information workers and services, tracking and managing activities, and intelligent distribution of information are essential to the efficient operation of any large enterprise. This is particularly important in the health-care domain, where many different organizations must cooperate to provide patient care reliably in a dynamically changing environment. In this review paper we present a distributed system that supports cooperative problem solving, activity management, and intelligent delivery of information in dynamic and unreliable environments. The system consists of a network of task/context managers (TCMs). Each TCM manages a group of related agents. It maintains up-to-date information on availability, operational status, and activities of participating agents, and it acts as a mediator between service requesters and service providers. In addition, the TCM acts as a representative for its agents with other TCMs, allowing different groups of agents to collaborate with one another. This paper describes the system architecture, its implementation and capabilities including matchmaking, plan monitoring, and failure recovery. Our system has been used in prehospital emergency patient information management applications.

Keywords: Distributed Agents, Medical Workstation, Software Engineering.

1. Introduction

The increasing use of workstations by health-care providers, administrative personnel, and ancillary services has given rise to the need for a general and reliable mechanism for coordinating activities and flow of information among agents in the dynamic, distributed, heterogeneous health-care delivery environment.

In a distributed collaborative environment, individual problem solvers or agents must cooperate to achieve common and individual goals. Unfortunately, distributed environments in the real world are dynamic and unreliable, imposing many challenges and difficulties in developing multi-agent systems. In such environments, agents may not be available or operational all the time, or communication between agents may not be possible at a given time. Yet, the system as a whole must continue to operate with the resources available, and fail gracefully when some activity cannot be performed due to unavailability of needed service agents.

To take a vivid, though somewhat extreme example, imagine the task of providing emergency medical services in the chaotic and dynamic environments of a battlefield or that of a national or regional disaster. We hastily assemble a team and deploy it. Each team member has different capabilities, and must cooperate with others in an unreliable communication environment. Consider a combat medical-support group consisting of surgical teams, battalion aid stations, transportation teams, intensive care units, laboratories, and other support units. They must move frequently to keep up with the changing conditions of the battlefield. Yet, each of the medical units must use the services and resources of the others in order to deliver life-saving health care. For example, a battalion aid station needs a transportation team to evacuate a patient. However, the designated transportation unit may be temporarily out of service, or out of communication reach. Just as this is happening, another transport unit returning from troop movement may be available to provide a "lift-of-opportunity". In such an evolving and constantly changing distributed environment, it is difficult, if not impossible, for an agent to keep track of all available agents and resources in planning and executing its tasks.

We have developed a system to address these needs. It consists of a

network of *task/context managers* (TCMs), which act as mediators between service requesters and service providers. A TCM keeps track of the current status of a group of agents, matches and routes service requests to the most appropriate agent(s), monitors the progress of involved agent(s), recovers from agent failure if it occurs and, finally, composes the results based on the agent responses and delivers the results. Furthermore, when a TCM is unable to resolve a service request using the local agents it manages, it can delegate the request to other TCMs. The ability to network TCMs together allows the system to be scaled and/or reconfigured during operation.

Three main aspects need to be addressed when building a distributed multi-agent system: agent theories, system architectures, and communication languages [14]. Agent theories define agents, their properties and reasoning schemes. System architectures deal with structures in which agents are organized, and how they communicate with one another. When environments are dynamic, the most important feature of a system's architectures is reconfigurability; an agent must be able to join the system at any time, to leave the system gracefully or due to a failure, and to rejoin the system later. Agent communication languages provide the interface for inter-agent and human-agent communication, as well as the task primitives and methods by which agent programs are compiled and executed. Although all three aspects are important, the system architecture is critical for effective operation in dynamic environments.

In this paper, we describe our system, with a focus on agent properties and system architectures. In section 2, we discuss required agent properties for cooperative problem solving. In section 3, we introduce Task/Context Managers and discuss their functions. We then consider how TCMs can be connected in order to deliver a broader coverage of services in section 4. We present an application of our system for patient-information management in section 5, discuss related work in section 6, and finally conclude in section 7.

2. Agents

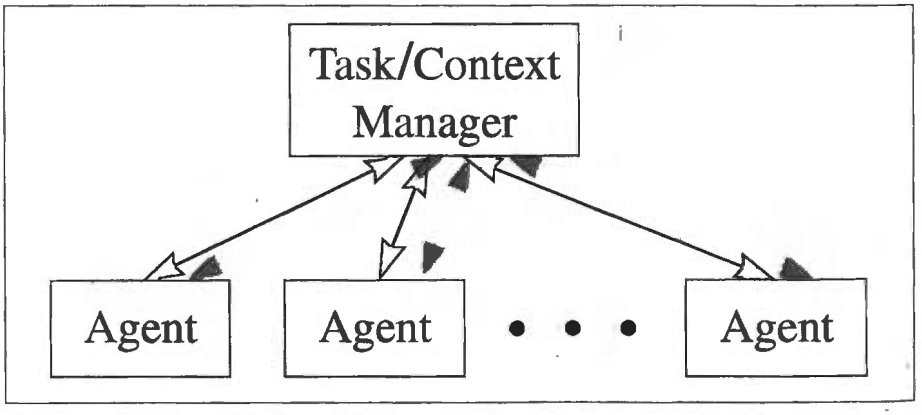
An agent is a problem solver *capable* of performing certain services. In a cooperative system, an agent needs to communicate its capabilities to other agents. Furthermore, an agent should be willing to accept a request if it can perform the requested service. More important, an agent should be honest, not committing to any request that it cannot fulfill. An agent can delegate to other agents a task or a sub-task which it cannot carry out. It may also hold and communicate preferences in selecting service providers. In short, an agent is an entity whose state can be viewed as consisting of mental components such as beliefs, capabilities, choices, and commitments [11].

Agents may communicate with one another directly. Such direct communication is important when large quantities of data, such as image files, need to be exchanged between agents. However, direct communication between agents for the purpose of recruitment (that is, assessment of their current capabilities and availability) is inappropriate for several reasons. First, the agents will be overloaded with status requests. Second, the communication traffic among agents will increase significantly, because each agent will need to poll all other agents in order to identify the most suitable agent to recruit. Finally, an agent may not know which other agents are operational at a particular time, and may thus waste time contacting agents that are not operational. In addition, each agent will need to encode plans for the execution of complex tasks (tasks requiring multiple steps), error-recovery plans when other agents fail, and contingent plans when appropriate agents cannot be found. To overcome these problems we introduce another class of agents, called *Task/Context Manager*. The main idea in adopting TCMs is to organize agents hierarchically into groups and to manage each group using a supervisory agent (the TCM).

3. Task/Context Managers

Agents may be organized in a group along many dimensions. For example, agents may be organized along geographic or organizational lines, they may be organized as a team of complementary agents with specified capabilities (e.g., a multi-specialty team), or as a collection of similar agents (e.g., a single specialty team). After an agent group is identified, a TCM can be introduced to manage and coordinate agents in that group. Each agent registers with the TCM and communi-

Figure 1. Tree structure of TCM-agent connections.



cates directly with it. Two agents communicate their service requests and deliver results to each other through the TCM, as shown in Fig. 1.

The two main functions of a TCM are (a) obtaining and maintaining information about agents, including their capabilities and status; and (b) acting as a mediator between an agent requesting service and agent(s) providing services to the requester.

3.1. Active Bookkeeping

In order to act as a mediator between service requesters and service providers, a TCM first needs to obtain and maintain the information about agent capabilities and status. This can be done in two ways.

The first is called a *TCM-active scheme,* with an assumption that the TCM knows the communication addresses at which agents may listen. The TCM periodically sends a status-inquiry message to these addresses. When a new agent comes along, it uses a pre-assigned communication address and waits for a status-inquiry message from the TCM. Whenever such a message is received, the agent responds to it by providing information about its capabilities and status. When a response comes from a previously inactive address, the TCM creates a new agent record based on the responding message; otherwise, the TCM updates the agent information. This method suffers from two deficiencies. First, it increases the communication needs; second, it requires prior assignment of communication addresses to each agent, thus making agents less mobile, and the system less scalable. On the other hand, this approach does not require each agent to register/unregister with the TCM, and agent malfunctions can be quickly detected and corrected.

The second is called an agent-active scheme. This scheme assumes that the communication address of the TCM is known to each agent in the group managed by the TCM. When an agent wishes to become active or change its status, it sends a status message to the TCM, and the TCM records the agent's status accordingly. This scheme overcomes many of the problems associated with the previous scheme, but it suffers from the problem that agent malfunctions become apparent to the TCM only when it attempts to contact the agent for additional tasks.

To overcome these problems, the implemented software uses a modified agent-active scheme. Here, the TCM maintains information about the elapsed time since the last communication with an agent. Each time contact is made between the TCM and the agent, the agent's "active" status is updated and the alarm clock is reset. If no contact between the agent and the TCM occurs within a pre-specified time interval, the TCM initiates contact for the purposes of updating the agent status by polling the agent (as in the TCM-active scheme).

In summary, a TCM maintains a table of currently active agents including their capabilities and status. The first time a connection between the TCM and a new agent or a recovered agent is established, the TCM registers the agent by adding it to the active agent table. The TCM updates information on agents whenever new information is available. Thus, each TCM maintains up-to-date information on the availability and status of each of the agents it controls. In the next section we describe how the TCM mediates between agents requesting services and those providing them.

3.2. Mediator

A TCM can be viewed as a blackboard system with a specialized control structure. A service requester can post the request to the blackboard. It is the TCM's responsibility to see that the request is serviced. Servicing a request involves four steps:

1. Matchmaking and/or planning: developing a sequence of actions (tasks) that must be taken to satisfy the request based on the available agents and resources, matching individual tasks to available agents, and routing tasks to selected agents.
2. Monitoring and detecting: monitoring the execution of tasks and the progress of involved agents, and detecting possible failure.
3. Error recovery: recovering from agent error and other resource failure.
4. Responding: integrating responses from service providers and generating a response to the service requester.

In the following subsections we look at each of these steps in greater detail.

3.2.1 Matchmaking and Planning

A request or task may originate from a human operator interacting with an application agent or may be automatically generated by an agent in the course of its operations.

The TCM begins by matching the task statement with the capability descriptions of the available agents. If the match identifies one or more capable agents, the TCM simply acts as a matchmaker. That is, it selects the most appropriate agent among those capable of performing the task, constructs a request for the selected agent to perform the task, and routes the request to the agent.

Task requests that can be executed by a single agent in the current environment are called *primitive*. If a task cannot be performed by any single available agent, it is called *non-primitive*. To perform a non-primitive task, the TCM must use plans that (hierarchically) decompose the task into primitive sub-tasks that can be performed directly by available agents. This can be done in two ways: static or dynamic. In the static method, a set of predefined plans are stored in a database. Given a service request, the TCM

identifies all the applicable plans and orders them, based on its preferences. Preference is based on a number of factors, such as: plan size (the number of primitive tasks in a plan), locality (local agents are preferred over those belonging to other TCMs), estimated task completion time, and cost.

Given an ordered set of plans, the TCM chooses the most promising plan to execute. If the execution of the current plan fails, the next most promising plan is chosen for execution. This continues until a plan succeeds, or all applicable plans are exhausted.

In the dynamic method, a planning system such as [13] is used to generate a plan with the request as its goal based on the current system status, i.e., the available agents and resources. If the plan fails during execution, the planning system can replan around the failure, utilizing as much of the previously executed plan as possible.

Each of the two approaches has advantages. The static method is simpler and more predictable, and it allows organizational work-flow to be encoded and automated. The dynamic approach, on the other hand, is more flexible, can respond more readily to changes in operating conditions, and can handle unusual situations. Our current implementation of TCM uses the static method because of its simplicity.

After a plan is selected or generated, the TCM begins execution of the plan by constructing appropriate agent requests and forwarding the results from one agent to the next as dictated by the plan. Upon successful completion of the plan, it constructs a response to the original request and returns the results to the originating agent. The next section describes the monitoring and failure recovery aspects of plan execution.

3.2.2 Monitoring, Detecting, and Recovery

To facilitate monitoring and recovery from error and failure, the TCM assigns a unique identifier to each agent, which is used by the agent to identify itself at all times, and a unique identifier to each task (transaction). In addition, the TCM and agents independently record their activities in their own log files (or database). An agent needs to record information regarding service requests sent and results received, completed tasks, and the task currently being executed. The TCM needs to record information on service request received, results forwarded, and the plan currently being executed for each pending task.

There are at least three levels at which a failure can occur. The first occurs at the task level. The preconditions or resources of a task may not be valid after it starts. For example, the laboratory equipment an agent was planning to use suffers a breakdown, or the resource the agent was planning to use is re-allocated to a more urgent task. The second occurs at the agent level. An agent may have an internal malfunction or may suffer from communication failure. The third and most critical failure can occur at the TCM level. Similar to an agent, a TCM can become inoperational due to internal malfunction or loss of communication. For robust operation of the system, it is critical that the impact of these failures be minimized. Strategies for recovering from each of these is described below.

If an agent can detect a task failure, it can report the failure to the TCM, and the TCM can initiate recovery procedures in a fashion similar to that employed for error recovery in multiphase database transactions. If an agent malfunctions, crashes unpredictably or suffers a loss of communication, it cannot report the failure to the TCM. To detect these situations, the TCM must monitor the execution of each task. However, continuous polling of each task's status is not practical due to high computational and communication overheads. Furthermore, a better scheme can be devised based on the fact that a functioning agent assigned to a task is required to either complete the task or report failure.

When a TCM sends a task to an agent, they first negotiate a completion deadline for the agent to deliver the results. The TCM suspects that the agent has failed if it does not receive a response or notification from the agent within this time limit. When an agent failure is suspected, the TCM sends an inquiry to the agent requesting a status update. If the agent does not respond, a failure is assumed and recovery procedures are initiated. If the agent responds, the TCM and the agent can renegotiate the task and set a new deadline. Finally, to facilitate the tracking of tasks by the TCM, the agent may embed the status of outstanding tasks in responses to other (possibly unrelated) messages from the TCM.

An agent failure may cause the TCM to stop executing the current plan. When this happens, the TCM instructs all the involved agents to abandon related tasks, and switches to a new plan. In some cases, it is possible for the TCM to recover from agent failure by replanning and/or reassigning the failed task to other agents. If the recovered plan can still be completed within the required deadline, the process can continue from here. If it cannot, then the TCM must negotiate with the originator of the request for a new deadline or abandon the task.

Since a TCM is the central piece of a group of agents, its failure can be catastrophic. To prevent a disaster caused by a TCM failure, we introduce a secondary TCM. The secondary TCM receives and maintains all the information available to the primary TCM, and runs on a machine other than the one running the primary TCM. It monitors the status of the primary TCM and takes over its role when it detects a failure in the primary TCM. Whenever the TCM writes a

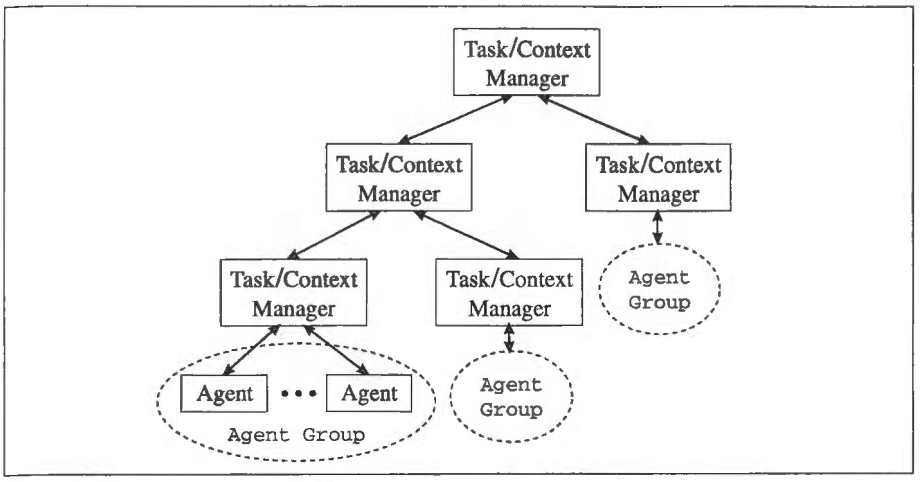


Figure 2. Hierarchical network of TCMs.

new record to its log file, it also forwards the record to the secondary TCM. The secondary TCM enters this information in its log file, and updates the record of the primary TCM's status. When the secondary TCM detects a failure in the primary TCM (i.e., no messages are received and the TCM does not respond to the status request), the secondary TCM changes its status to active TCM and informs the agents in the group that it is now the TCM for the group. When the original TCM recovers from the failure, it can contact the secondary TCM and resume its role by first obtaining the log file and then contacting the agents in the group.

In the next section we describe the organization and operation of a network of TCMs.

4. Information Mediator Network

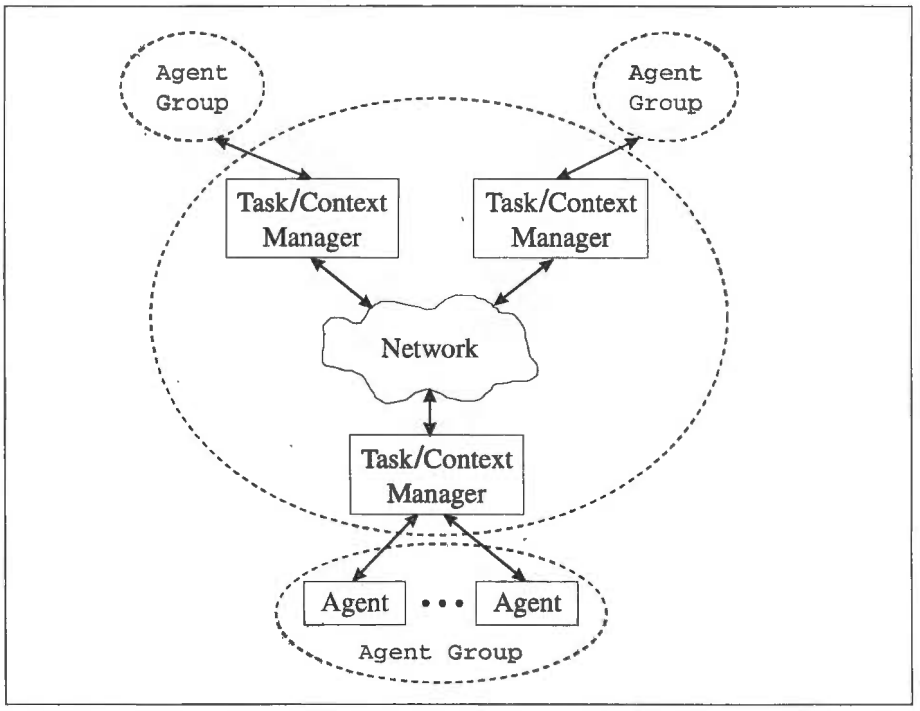
Similar to agent organization, TCMs can also be organized into larger structures. TCMs can be organized in a hierarchical structure or in a fully connected network, shown in Figs 2 and 3. In the first architecture, several TCMs are grouped together under the control of a higher level TCM which mediates between its member TCMs. This grouping can continue at a higher level of TCMs. Whenever a TCM cannot find an agent or a lower level TCM in the group to carry out a task, it passes the task as a service request to its superior TCM, which in turn tries to find an agent or another TCM that can handle the task. In the second architecture, TCMs are networked together as a group of peers. Whenever a TCM cannot resolve a service request within its group of agents, it consults its peer TCMs. If one of the other TCMs can handle the request, the requesting TCM forwards the request to that TCM. The second architecture is more appropriate when the number of TCMs is not very large and required service response time is short.

We believe that as the number of TCMs grows, a combination of the two architectures described above will be needed to provide a proper balance between the response time and communication costs.

5. Applications

The application domain of our system is patient information and activity management. In the civilian domain, we are interested in coordinating team activities involved in pre-hospital emergency medical services, as well as coordinating and managing routine hospital-based patient-care activities such as registering a patient, planning a patient hospital visit, and coordinating activities among physicians, nurses, laboratories, and other ancillary and administrative services. In military domains, we are interested in tasks of mobile hospital control and management, including maintaining dynamic status of battalion aid stations, surgical teams, transportation

Figure 3. Interconnected network of TCMs



teams, different laboratories, and other involved units; and coordinating patient activities.

Our system is implemented using Microsoft Visual C++ and Microsoft Access database software. The network communication is based on the TCP/IP protocol, and inter-agent communication is based on Health Level 7 message format with minor extensions.

The implemented system uses an agent-active scheme to establish a connection between a TCM and an agent (see section 3.1), i.e., the first connection message is initiated by the agent. For ease of implementation we use a database of pre-defined (hand-crafted) plans for task decomposition rather than a planning system to dynamically generate plans at run-time. Furthermore, we use the fully connected network architecture to organize TCMs, since the number of tasks sent across different TCMs is small.

6. Related Work

The area of software agents and distributed multi-agent systems is an active and rapidly growing topic of current research. The available and relevant literature is too vast to be adequately covered here. Interested readers are encouraged to look at [7].

The idea of a Task/Context Manager was originally proposed in the context of a healthcare professional's workstation [10]. The idea of mediators was originally proposed by Wiederhold [12] as a means of organizing heterogeneous database systems. Considerable work in this area has been carried out under the ARPA Knowledge Sharing Effort [9], and under ARPA's Intelligent Integration of Information program. Examples of systems based on mediators include SIMS [1, 2], which is a mediator between database queries and multiple data sources. A special mediator, called matchmaker [6], was used in a groupware system [8].

A mediator network based on facilitators, called a federated system, has been proposed [4]. In such a system a facilitator routes messages to and from agents running on a machine [4, 5] in order to schedule and maintain the flow of communication. Cohen et al. [3] suggest an open architecture that is similar to the one presented in Fig. 2. Their system, however, does not address the issue of dynamic reconfiguration, and does not address issues of failure detection and error recovery.

7. Conclusions

We have presented a multi-agent system for distributed cooperative problem solving. In our system, we introduced task/context managers (TCMs), which act as mediators between service requesters and service providers. A TCM manages a group of agents solving domain-specific problems, maintains information about the capabilities and status of the agents, makes a plan to satisfy a request, matches and routes tasks to agents, monitors the execution of the plan, and detects and recovers from agent failure. We further introduced network architectures that connect many TCMs to provide scalability and collaboration among large groups of agents. The architectures presented in this paper are most suitable for robust distributed collaborative problem solving in dynamically changing environments.

Acknowledgement: This research was supported in part by ARPA contract MDA972-94-2-0010, and NLM grant 1 RO1 LM05324

References

1. Arens Y, Chee CY, Hsu C-N and Knoblock CA. Retrieving and integrating data from multiple information sources. Int J Intell Cooper Inform Syst 1993;2:127-58.
2. Arens Y, Knoblock CA, Shen W-M. Query reformulation for dynamic information integration. J Intell Inform Syst 1996, in press.
3. Cohen PR, Cheyer A, Wang M, Baeg SC. An open agent architecture. In: Etzioni O, et al, eds. *Working Notes of AAAI 1994 Spring Symp on Software Agents.* Stanford CA: American Association of AI, 1994:1-8.
4. Genesereth MR, Ketchpel SP. Software agents. Commun ACM 1994;37:48-53.
5. Genesereth MR. An agent-based approach to software interoperability. *Proc. of the DARPA Software Technology Conf.* Arlington VA: Computer Science Department, Stanford University, Technical Report 1992:2.
6. Kuokka D. and Harada L. 1995. Matchmaking for information agents. In: Mellish CS, ed. *Proc of the 14th Intern Joint Conf on Artificial Intelligence (IJCAI-95).* Montreal: Morgan Kaufmann, 1995:672-8.
7. Lesser V, ed. *First Int Conf on Multi-Agent Systems.* Menlo Park CA:AAAI Press/The MIT Press, 1995.
8. McGuire J, Kuokka D, Weber J, Tenenbaum J, Gruber T, Olsen G. SHADE: Technology for knowledge-based collaborative engineering. J Concurrent Engineering: Research and Applications 1993;1.
9. Patil RS, Fikes RE, Patel-Schneider PF, McKay D, Finnin T, Grube TR, Neches R. The DARPA knowledge sharing effort: Progress report. In: Nebel B, Rich C, Swartout W, eds. *Proc of 3rd Intern Conf on Principles of Knowledge Representation and Reasoning.* Cambridge MA: Morgan Kaufmann, 1992:777-88.
10. Patil RS, Silva J, Swartout W. An architecture for healthcare provider's workstation. Int J Biomed Comput 1994;34:285-99.
11. Shoham Y. Agent-oriented programming. Artif Intell 1993;60:51-92.
12. Wiederhold G. Mediators in the architecture of future information systems. IEEE Computer 1992;25:38-49.
13. Wilkins DE, Myerss KL, Lowarance JD, Wesley LP. Planning and reacting in uncertain and dynamic environments. J Exp Theor Artif Intell 1995;7:121-152.
14. Wooldridge MJ, Jenning NR. Agent theories architectures and languuages: A survey. In: Woolridge MJ, Jennings NR, eds. *Intelligent Aggents: Theories Architectures and Languages.* New York: Springer-Verlag 1995:1-32.

Address of the authors:
Ramesh Patil, Weixiong Zhang, Wei-Min Shen,
USC / Information Sciences Institute,
4676 Admiralty Way,
Marina del Rey, CA 90292,
USA.
E-mail: {ramesh,zhang,shen}@isi.edu