Intelligent mobile agents in large distributed autonomous cooperative systems

Johnny S.K. Wong a,*, Armin R. Mikler b,1

a Department of Computer Science, Iowa State University, Ames, Iowa 50011, USA
b Department of Computer Sciences, University of North Texas, Denton, TX 76203, USA

Abstract

A large distributed autonomous cooperative system is a system that provides fundamental services for integrating high-level services in a large distributed system with local autonomy for individual system platforms and processes. This paper discusses the role and the functions of Intelligent Mobile Agents (IMAs) in large distributed autonomous cooperative systems. Rather than providing services to a user at the application level, IMAs considered in this paper are deemed an integral part of system level software and perform tasks that are considered central to the distributed system. A variety of solutions to problems that are inherent to the distributed nature of the computing infrastructure may be implemented through a system of IMAs. These problems include, but are by no means limited to, load balancing, scheduling, information retrieval and management, distributed decision support, routing and flow control, security and intrusion detection. In this paper we discuss some of characteristics of IMA based systems central to solving some of these problems. The effectiveness of IMAs in large distributed systems clearly depends on the design and implementation of the underlying IMA architecture. This paper discusses the features that must be provided by the IMA infrastructure in support of IMAs that are an integral part of the large distributed autonomous cooperative system. © 1999 Elsevier Science Inc. All rights reserved.

1. Introduction

We view a decentralized autonomous cooperative system (DACS) as a collection of independent computing resources, which cooperate to provide fundamental functionality for integrating high-level services upon which a distributed system can be built (Chow and Johnson, 1997). The system is decentralized in that there is not one dedicated entity, which can coordinate the cooperative effort to provide this functionality. Autonomy comes from the fact that the system represents an open computing environment in which participating hosts make autonomous decisions about actions necessary to perform the required functions. In order to coordinate their efforts, individual hosts must be able to acquire information about the state of the system and communicate actions and decisions among each other. This, however, represents a non-trivial task, particularly if the system is composed of hundreds or even thousands of loosely coupled hosts, distributed over a large geographic region.

The effectiveness of coordination and the soundness of individual decisions in the global context are a function of the precision and the uncertainty of information upon which these decisions are based. Due to the fact, that knowledge acquisition takes a finite amount of time and the message delay in a large decentralized infrastructure is non-negligible, the views of the system, as established by the participating hosts, are often incomplete and incoherent. In addition, there are generally limitations on the availability of resources that can be extended to the acquisition and representation of a global state view. In general, we would expect the individual entities in the decentralized infrastructure to display some type of rational behavior. Rational behavior, defined informally, is the ability to choose the most preferable from a set of available. In terms of utility-functions, an entity will select an action in the attempt to maximize the expected utility (Luce and Raiffa, 1985). The meaning of utility or preferences in the context of a decentralized autonomous cooperative system has not been defined and is not at all obvious. In fact, we have to recognize that the constituent sub-systems of the DACS, such as the underlying communication network, are themselves a DACS with their own concept of preference.
Clearly, a DACS may display rational behavior at a number of different and not necessarily consistent tiers. At the user level, users may expect the system to perform, so as to optimize the current application. At the systems level, we may expect the system to attempt to optimize performance over all the applications that are currently active. In addition, the DACS must be able to optimize system-level activities for it to provide basic functionality and to attempt optimality. It will be necessary to tradeoff among the preferences of the individual tiers in order to make the system cooperative at multiple levels. An example of such tradeoff is the necessity to delay or reduce application-oriented computation in order to optimize throughput at the communication level. In general, the participating entities in the DACS must be able to categorize and rank-order all possible actions based on information about the global state of the system. The emerging decisions will generally have multiple objectives.

Knowledge Acquisition (KA) and Knowledge Representation (KR), associated with the computation of a global view of the state of the system are directly impacted by the magnitude and complexity of the DACS. New KA and KR mechanisms are needed that are capable of discriminating among knowledge sources in the system so as to generate a sufficiently precise view of the system. Resource monitoring, performance monitoring, resource discovery, and system load management are examples of tasks that generally rely on the performance of the underlying KA and KR mechanisms.

Distributed system research over the last two decades has resulted in a variety of innovative mechanisms, which enable distributed hosts to cooperate in forming a common, ubiquitous computing platform. While the algorithms and tools developed in recent years support the realization of distributed systems, they generally are not designed to easily adapt to an ever changing computing infrastructure with often unpredictable dynamics. Also, these mechanisms are not suitable for ad-hoc collaborative environments, as the availability of mechanisms is usually a precursor for becoming a participant in the distributed computing infrastructure. However, as the computing infrastructures grow it is difficult to take various goals and preferences of the subsystems into consideration. In most cases, there does not exist a single optimal strategy, which yields optimal decisions for all the subsystems. For reasons mentioned above, hosts generally do not have access to global state information and it is often impossible or impractical to compute a consistent preference structure among the hosts. This warrants the reevaluation of the existing mechanisms in distributed computing in the context of large decentralized autonomous systems. In addition to scalable KA and KR implementation, new mechanisms for the coordination of actions and the cooperation among the constituent computing resources are needed.

Recent research emphasis on mobile code in the form of intelligent mobile agents (Bradshaw, 1997) and active networks (Tennenhouse and Wetherall, 1996), offers new exciting possibilities for developing solutions that scale well for a variety of Decentralized Autonomous Cooperative Systems. The merging of the two research areas is deemed to yield new mechanisms for building very large, robust distributed systems. Hence, we should revisit the various research areas in distributed computing system and reexamine existing solutions to determine whether they can be improved through IMA-based mechanisms (Smith, 1998).

2. Software agents and their execution environment

In general, an agent is a computer program (thread) that acts autonomously on behalf of a person or organization. Agents can be stationary (executes only on the system where it began execution) or mobile (can move to a remote system for execution). When an agent travels, it transports its state and code with it. In this context, the agent state can be either its execution state, or the agent attribute values that help it determine what to do when it resumes execution at its destination. An agent’s execution state is its runtime state, including its program counter and frame stacks (Bradshaw, 1997; OMG, 1998).

An agent’s authority identifies the person or organization on whose behalf the agent acts. Constituent system components must be able to authenticate this authority. Agents require names so that they can be identified in management operations, and can be located via a directory service. Agents are named by their authority and identity. An agent’s identity is a unique value within the scope of the authority that identifies a particular agent instance.

The location of an agent is the address of a place. A place resides within an agent system. Therefore, an agent location should contain the network address (for example, IP address and port number) of the agent system where the agent resides and a place name. An agent system is a platform that can create, interpret, execute, transfer and terminate agents. Like an agent, it is associated with an authority that identifies the person or organization on whose behalf the agent system acts. A host can contain one or more agent systems. An agent system can contain one or more places and a place can contain one or more agents. An agent system type describes the capability of an agent. For example, if the agent system type is Aglet, the agent system is implemented by IBM, supports Java as the Agent Language, uses Itinerary for travel, and uses Java Archive Format for its serialization. All communication between agent systems goes through the Communication Infrastructure (CI).
When an agent transfers from one system to another, it travels between execution environments called places. A place is a context within an agent system in which an agent can execute. This context can provide functions such as access control, for example. The source place and the destination place can reside on the same agent system, or on different agent systems that support the same agent profile.

A region is a set of agent systems that have the same authority, but are not necessarily of the same agent system type. The concept of region allows more than one agent system to represent the same person or organization. Regions allow scalability because we can distribute the load across multiple agent systems. Multiple regions can be interconnected via one or more networks. They may or may not share a Directory Service, this depends on the agreement between region authorities and the implementation of these regions. A non-agent system can also communicate with the agent systems within any region as long as it has the authorization to do so. A region can contain one or more agent systems. The systems outside of a region may access the region via the agent systems that are exposed to the outside world, similar to a firewall configuration. These agent systems are called region access points.

Agents, Software Agents, Intelligent Mobile Agents (IMA), and Softbots are terms, which describe the concept of mobile computing or mobile code (Bradshaw, 1997). Mobile code is orthogonal to the well-known Remote Procedure Call (RPC), in that the program is migrating to the data, rather than the data being transferred to the executing program. Over the last decade, the AI community has put forth efforts to define the characteristics of software agents in general and IMA in particular. One can dichotomize IMA according to four characteristics, namely, intelligence, cooperative behavior, autonomy, and mobility (Rothermel and Baumann, 1997). While most computer scientists would agree that these characteristics are intrinsic to an IMA, there is little consensus on how and to what degree the IMA will manifest each of them. We define the characteristics of an IMA as follows:

- **Intelligence**: It is the ability to adapt to circumstances brought upon by the dynamics of the system. This adaptation may be limited to each individual agent or encompass the entire agent system. Examples of adaptive behavior are changes in agent’s itineraries as a function of resource dynamics and variation in size of the agent population as a function of system dynamics.
- **Cooperative behavior**: It describes the ability to share knowledge among agents and/or negotiate a common strategy that yields actions that lead to an overall acceptable performance (Rosenschein and Zlotkin, 1994). Systems composed of multiple agents are generally referred to as ‘Multi-Agent Systems’.
- **Autonomy**: It allows agents to execute without the interference of users. An agent displays autonomous behavior if it is able to complete an assigned task by appropriately choosing from a set of possible strategies.
- **Mobility**: It describes an agent’s ability to transfer between different hosts. We need to distinguish two types of mobility: remote execution and migration. For remote execution, an agent transfers to a remote site and executes until task completion. A migrating agent may utilize a number of hosts for the completion of the task, that is, the agent may suspend its execution, move to a different host, and resume execution there.

Although the particular manifestation of these characteristics in specific applications may differ, we believe that the above definitions are general enough to classify the agent’s functional modules.

Most software agents that have been developed in recent years perform on behalf of the user. Examples of such agents are e-mail agents, filter agents, and search agents. These agents usually require interaction with the user. In contrast, the proposed program emphasizes the experimental design, implementation, and analysis of IMAs as an integral part of the DACS. These agents implement system functions and will execute without explicit user control. For these agents, autonomy is clearly an important characteristic. Instead of executing on behalf of a user, the IMAs for a DACS are considered system agents, which execute on behalf or in support of the system itself. In general, the IMA approach is deemed appropriate for problems that require a system to autonomously fulfill a variety of tasks in a dynamic, unpredictable environment (Maes, 1994).

The tremendous increase in research activities that focus on agents-based mechanisms is indicative of the versatility of the concept at large. Different scientific communities are investigating a large variety of issues that are related to agents (Baumann, 1995). While the AI community extends research efforts towards the principles of agent behavior, agent cooperation, and multi-agent systems, current systems research emphasizes on the design and the deployment of agents to meet specific functional goals (Schelen and Pink, 1997). Most notably, recent research effort in the area of active networks (Tennenhouse and Wetherall, 1996) is indicative for the trend to deploy mobile agents thereby enabling system components to perform functions that otherwise cannot be provided. With the emergence of active networks, applications (and users) will be able to install customized code in network nodes, thereby biasing basic network function for certain classes of traffic. In the context of this paper, this customized code may be viewed as a mobile agent. The idea of deploying mobile agents in telecommunication systems was introduced in the mid 1990s and led to a research
area generally referred to as intelligent networks (IN) (Magendanz et al., 1996). In this context, however, mobile agents have been used as a catalyst for providing intelligent network services, very similar to the concept of enabling routers in active networks to execute user/application specific code. The next generation of mobile agent research in the field of communication aimed at the deployment of mobile code in support of network management. This triggered the design and implementation of new service architectures that provide the necessary mechanisms to accommodate mobile agents (Krause, 1997; Magendanz et al., 1996). In the research domain of computer networking, agents have been designed to interact with existing network protocols, such as RSVP and SNMP, for the purpose of network resource reservation and management. Intelligent agents form the basis for the development of new mechanisms that enable the network to maintain the requested quality of service (QOS) (Schelen and Pink, 1998).

The significance of agents, mobile or stationary, being able to adapt to the often unpredictable dynamics of the environment has been investigated (Mikler et al., 1996). The use of utility-theoretic heuristics for message routing in large grid networks prone to failure, has resulted in a framework for designing and analyzing routing algorithms in general, and their knowledge acquisition and knowledge representation mechanisms in particular. One of the major results of this work was the design and implementation of intelligent routing heuristics, which weighed the significance of network state information as a function of distance, thereby reducing the overhead that is associated with the knowledge acquisition process.

Recently, a number of research projects have emerged aiming at the design and implementation of system oriented agents. As systems increase in size and complexity, more control is delegated to the system itself. In the network domain, the ever-increasing heterogeneity of large communication networks has triggered the deployment of agents to provide uniform capabilities. An example of this approach is the research work on Tunnel Agents (de Meer et al., 1998). This work has focused on the use of tunnel agents to provide end-to-end quality of service guarantees across a heterogeneous network infrastructure, even though some of the intermediate networks do not provide the support of the RSVP reservation protocol.

The use of intelligent mobile agents in the context of resource discovery is another area of research, which encompasses a number of different types of systems. In large communication systems, for instance, agents are used to discover the topology of the underlying network infrastructure (Minar et al., 1998). The performance of a single agent represents the baseline, against which the performance of collaborative agents of different types is compared. This work clearly depicts the advantages of agent cooperation in solving system-related tasks, such as resource discovery.

In large decentralized systems, agent collaboration has been exploited for the distributed allocation and sharing of resources. Challenger is a multi-agent system that is capable of managing a large set of distributed resources (Chavez et al., 1997). Agents individually manage single resources and cooperate with each other to allow for global resource sharing, thereby increasing the system utilization. The reported research results are promising, and suggest that the deployment of cooperating agents in a large DACS for resource allocation and sharing is a promising and very feasible approach.

A system of intelligent agents using collaborative information and mobile agent technologies (Bradshaw, 1997; Nwana, 1996) is being developed to implement a prototype intrusion detection system (Crobie and Spafford, 1995). The goals of the system design are to (Helmer et al., 1998):

- learn to detect intrusions on hosts and via networks using individual agents targeted at particular subsystems;
- use agent technologies to intelligently process audit data at the sources by using mobile agents;
- have agents collaborate to share information on suspicious events and determine when to be more vigilant or more relaxed;
- apply data mining techniques to the heterogeneous data and knowledge sources to identify and react to coordinated attacks on multiple subsystems.

A notable feature of the intrusion detection system based on data mining is the support it offers for gathering and operating on data and knowledge sources from the entire observed system. The system could identify sources of concerted or multi-stage attacks, initiate countermeasures in response to the attack, and provide supporting documentation for system administrators that would help in procedural or legal action taken against the attacker.

An example of an attack involving more than one subsystem would be a combined NFS and rlogin attack. In the first step, an attacker would determine an NFS file handle for an .rhosts file or /etc/hosts.equiv (assuming the appropriate file systems are exported by the UNIX system) (van Doorn, 1994). Using the NFS file handle, the attacker would re-write the file to give himself login privileges to the attacked host. Then, using rlogin from the formerly untrusted host, the attacker would be able to login to an account on the attacked host, since the attacked host now mistakenly trusts the attacker. At this point, the attacker may be able to further compromise the system. The intrusion detection system based on data mining would be able to correlate these attacks, help identify the origin of the attack, and support system management in responding to the attack.
There are many other on-going research and development efforts based on software agents. In what follows, we will be discussing several issues related to the design and implementation of an IMA-based DACS. These issues include requirements for a system level agent framework, agent-based knowledge acquisition, resource monitoring, inter-agent communication, and security.

3. Towards IMA-based DACS

Intelligent autonomous multi-agent systems represent a new programming paradigm for designing, analyzing and implementing complex and sophisticated software systems. Agent-based systems have been claimed as ‘the next significant break-through in software development’ (Sargent, 1998), ‘the new software revolution in software’ (Jenning and Wooldridge, 1998), and ‘another emerging concept in the evolution of modern operating systems’ (Chow and Johnson, 1997). The key abstraction used in this new software paradigm is the concept of an agent, as defined previously.

At this moment, there are two major obstacles for the widespread use of the multi-agent technology, namely, the lack of a systematic design methodology to specify the multi-agent system for software systems and the lack of available industrial-strength multi-agent toolkits (Jenning et al., 1998). Furthermore, intelligent mobile agents are not the silver bullet that result in unprecedented improvements in the performance of software systems. As Wooldridge and Jennings (1998) pointed out, it would be naive to assume that the benefits of agent technology are implicit. As a matter of fact, it can be shown that whether or not agent-based solution constitute an improvement over conventional approaches depends (among other things) on the type of problem to be solved, the topology of the distributed computing environment, and the availability of computational resources. As mentioned above, most agent-based systems are defined and implemented at the application level. This constitutes an abstraction, which allows the environment to be modeled to the agent’s ability. For agent-based service integration at the system level, the agents and their respective execution environment must be modeled to the parameters of the communication and computing infrastructure. That is, if the network of agent daemons is implemented as application on top of the operating system layer and physical network, it is easy to disregard certain system constraints. If, however, agents are to execute as an integral part of the operating system itself, the agent interfaces must be implemented at the system level. Hence, the topological properties and computational characteristics of constituent hosts, routers, and gateways will affect the performance of the agent-based system.

As mentioned above, thus far, most agent-based systems have been developed at the application level. In these systems, the details of the operating system as well as the properties of the underlying communication infrastructure are transparent to the user and the agents that act on their behalf. It is not the agent runtime environment that provides this transparency but a special abstraction layer, often referred to as Middleware. With this abstraction, individual agents and in fact, the entire agent execution environment can be defined on top of a variety of very powerful services (rather than single functions). With the use of these services, one can construct an arbitrary computational infrastructure in which the relationship between constituent nodes is determined by functionality rather than topological and system related properties. At this level, it is irrelevant whether two communicating entities reside on the same host, or are half a world apart, separated by a complex conglomeration of interconnected networks. For the design of a distributed system at this level, the ability to communicate among processes on different nodes through inter-process-communication (IPC) is assumed (Lynch, 1996). The topological properties are built into the distributed system through constraining the ability of processes (or nodes) to communicate. In fact, the computational model may be drastically different from the physical computing and communication infrastructure, as depicted in Fig. 1. Clearly, the mapping from the physical infrastructure to the computational model can effect the attainable system performance. An agent-based software system, implemented at the level of the computational model, will have to work with the abstract constraints, and specific quality of service levels will have to be negotiated across the abstraction layers.

3.1. The need for agent system interoperability

While strictly delineated DACS will generally be based on a single agent implementation, very large infrastructures, and ad-hoc systems in particular may be based upon different implementations of agent systems.

Fig. 1. Computational model embedded in a physical network.
Hence, an important goal in mobile agent technology is interoperability between various agent systems. When the source and destination agent systems are similar, standardization of these actions can result in interoperability. However, when the two agent systems are dramatically different, only minimal interoperability can be achieved.

The following is a common scenario that would benefit from standardization of mobile agent actions. Suppose that AgentX (from Agent System A) wants to communicate with AgentY (from Agent System B). If Agent System A and Agent System B support the same agent profile, the ability to inter-operate is implicit. AgentX can request a transfer to Agent System B, obtain whatever information is required, and return to its original agent system.

However, if Agent System A and Agent System B support different agent profiles, there are two ways for the agents to communicate. In one of the two cases local communication is possible, in the other case it is not.

Case 1: Agent can take advantage of locality. In this scenario, a new agent system is added (Agent System C). Agent System A and Agent System C supports the same agent profile, and Agent System C is local (same host/network) to AgentY. AgentX discovers this fact via a call to Agent System B, then requests a transfer to Agent System C. Once AgentX travels to Agent System C, it can communicate locally with AgentY via IPC, get the information it needs, then return to Agent System A.

Case 2: Agent cannot take advantage of locality. In this scenario, AgentX cannot find an agent system local to Agent System B that supports the same agent profile as Agent System A. It therefore remains on HostA, and communicates with AgentY across the network via RPC. Although it does not have the advantage of locality, communication can still occur.

There are several areas of mobile agent technology that the mobile agent community should consider for early standardization in order to promote interoperability including: agent management, agent transfer, agent and agent system names, agent system types, and location syntax.

4. Agent movement and itineraries

The effects of topological properties on the performance of the agent system are best illustrated through a number of experiments in which agents assimilate properties of the system they are executing in. These results are based on those experiments used to evaluate the performance of agents in a network-mapping task (Minar et al., 1998). While the original experiments aimed at the evaluation of different types of agents, we emphasize on the effects of the underlying topology on the agents’ ability to discover the parameters of the system infrastructure. The fact that we have chosen the network-mapping task as one example of a class of knowledge acquisition problems is secondary. In fact, it should be noted that the discovery of system properties and the representation of the knowledge that was acquired in this process represent fundamental applications for intelligent mobile agents.

Our experiments are based on the simulation of agents in an arbitrary network topology modeled by a digraph \( G = (V, E) \). As an agent traverses the graph, either by random walk or conscientious movement, it acquires information about the topology of \( G \). Upon visiting a vertex \( v_i \), an agent assimilates knowledge of the existence of all edges \( e_{ij} \), selects the next vertex on the tour, and makes the transition. Other parameters, such as available bandwidth, current flow, or fault information may be collected in a more practical scenario. Under the assumption that all agents can move between vertices in unit time, the increase of topological knowledge in each agent over time is recorded. The fact that an agent has acquired additional topological information is represented as a change of uncertainty, i.e., the ratio of an agent’s knowledge to the total topological information (in the number of edges \( e_{ij} \) in \( G \)). The graph \( G \) used in our experiments consists of the two fully connected sub-graphs \( G_1 \) and \( G_2 \), which are connected by a single edge \( e_{12} \) to form \( G \) (see Fig. 2).

Representative results of our experiments are shown in Fig. 3. All agent execution traces shown in Fig. 3 contain sections in which there is no improvement of the topological information acquired by an agent. These plateaus occur whenever an agent traverses an area of the graph, which had been visited previously. The size (or length) of a plateau depends, among other things, on the agents itinerary, the agent’s starting point, and most noteworthy, the topology itself. In order to assimilate information from new regions, an agent may have to traverse areas already contained in the agent’s knowledge base. Since we assume that agents have no a priori information about the makeup of the environment, a plateau is an insufficient condition for the termination of the knowledge acquisition process. In fact, certain characteristics of the infrastructure may prohibit certain agent traversal to ever lead to convergence, that is, certain areas may remain unvisited.

A conscientious agent, as defined in the original experiment (Minar et al., 1998), traverses the graph by...
selecting the next vertex to be visited according to the number of previous visitations to that vertex. The vertex with the least number of visitations is then selected. With this strategy, an agent is attracted by new areas, which add new topological information. This can drastically reduce the time spent in an already known area of the graph. The preference order over the reachable vertices may be an effective heuristics, however, it does not represent an admissible heuristics (Pearl, 1984) for weakly connected topologies as the one shown in Fig. 2. While the sample graph may appear contrived, we believe that agents in a distributed environment may encounter such structures due to inconsistent routing table entries, bandwidth constraints, system faults, or ad-hoc collaborative infrastructures.

For a strongly connected graph, as shown in Fig. 4, the preference ordering deployed in conscientious agent does represent an admissible heuristics. That is, the goal to acquire knowledge from all vertices in the graph, thereby guaranteeing the agent’s knowledge base to converge will be achieved. Nevertheless, an optimal traversal of the graph cannot be guaranteed under the assumption that the agent does not have any a priori information. The graph depicted in Fig. 5 clearly illustrates the superior traversal by conscientious agents, yet also indicates the variation of performance. For agent applications, such as system monitoring, resource discovery, traffic management, and fault management, the performance goal is, however, to obtain complete and reliable system state information in the least amount of time possible.

In the above example, network-mapping agents had to assimilate knowledge of a static network topology. Consequently, the information in the agent’s knowledge base increased monotonically with time. In more realistic applications, the agents may have to monitor dynamically changing system parameters, such as network flow, available bandwidth, buffer space, system load, or quality of service commitments. As before, an agent may traverse the distributed infrastructure and assimilate the corresponding parameters from constituent entities. The information is then made available to a decision process, which may be a system administrator, a special process, or the agent itself. The quality or effectiveness of actions taken as a consequence of the information obtained by the agent will clearly depend on the precision and completeness of this information. The reliability dynamic system information does not only depend on the underlying topology, but also the system’s dynamics. In this scenario, the agent’s knowledge base changes non-monotonically, and the strategy previously employed by conscientious agents is no more admissible, as the objective has changed to reflect the intricacies of a dynamic environment. Hence, agents must follow an itinerary of system traversal, which optimizes the reliability of information. Since system topology and system dynamics are seldom known a priori, agents may have to derive an
optimal itinerary over time and adapt to changing system parameters. In order to meet the above performance objectives and to tackle some of the issues implicit to a large, dynamic distributed infrastructure, ongoing research in the area of intelligent mobile agents and distributed computing aims at the deployment of multiple collaborative agents.

5. Multi-agent systems

The area of multi-agent systems has its roots in the field of distributed AI. The primary research emphasis has been on distributed problem solving and coordination of computational constituent modules in order to find efficient solutions (Jenning et al., 1998). In recent years, agent and multi-agent systems have experienced some degree of anthropomorphism, in that agents have been attributed with implicit intellect allowing computer scientists to study sociological aspects of multi-agent systems. In the context of DACS, we shall take a more problem oriented approach by investigating the intricacies of the application and how agents may exploit the intrinsic properties of the solution space and the environment they act in. Our emphasis shall be the performance of agents in a specific environment applied to a particular problem. Rather than making the agent’s autonomy the central point of study, the goal is to develop an analytical framework in support of the design and analysis of agent-based systems. Although the constituent agent’s in a DACS may act autonomously, they coordinate their effort by sharing information, thereby implicitly synchronizing their actions. The advantage of multiple agents in a DACS lies therefore in the division of labor, similar to a parallel processing system. The following example illustrates the advantages of multiple cooperating intelligent mobile agents in a resource-monitoring scenario.

5.1. Cooperating agents for resource monitoring

In any distributed environment, a number of IMAs with active interaction will lead to a high degree of information exchange across the entire domain. Here, uncertainty is the degree of measure of the maximum error associated with the quantified information carried by the agents. An agent retrieves information from a resource \( R \) at a time instant \( t \). Some other resource \( R' \) may request for that specific information from the agent. Assuming that the reply reaches resource \( R' \) at time instant \( t + t' \), we define uncertainty of information as the maximum phase lag of information reported at \( R' \) when compared to the actual value at this time instant \( t + t' \). For simplification, we assume a homogeneous distributed system of \( n \) resources. The system is further assumed to be topologically independent (i.e., the underlying topology is not taken into consideration), and characterized by a uniform stochastic process. The network topological constants are proportionality constants in a given topology, related to inter-node distances in the network. In what follows, we use multi-agents for optimization of global information exchange throughout the distributed system. The problem is to compute an upper bound on the degree of uncertainty of the information carried by these agents.

Fig. 5. Random versus conscientious graph traversal.
We define the operators in Table 1 as follows:

First, we consider a single IMA, which traverses a system of \( n \) resources. Based on a sequential itinerary, the agent visits each of the system’s resources in sequence. Using this itinerary, the agent assimilates a knowledge base, which contains the values \( KB(t) \) with the following values at time \( t \):

\[
KB(t) = [V_1(t) - (n - 1)u, V_2(t) - (n - 2)u, \ldots, V_n(t)].
\]

However, the actual values maintained by each of the resources are: \([V_1(t), V_2(t), \ldots, V_n(t)]\). We compute the corresponding uncertainty of information, \( \gamma \), for this scenario as a measure of the time lags.

\[
\gamma = k[(n - 1)u + (n - 2)u + \cdots + u] = ku[(n - 1)n/2]. \tag{1}
\]

In Fig. 6 intuitively, one would expect the sequential itinerary to yield a knowledge base with the least uncertainty, under the assumption that the agent monitors a set of \( n \) independent, homogeneous resources implemented by the same stochastic process. To show that this conjecture is indeed true, we devise slightly from the sequential itinerary. Without loss of generality, we change the agent’s traversal such that instead of visiting resource \( R_n \) at time \( t \), the agent visits resource \( R_1 \). With this modified resource traversal, the agent’s knowledge base can be described as:

\[
KB(t) = [V_1(t), V_2(t) - (n - 2)u, \ldots, V_n(t - u')]
\]

where \( u' \geq nu \).

The corresponding uncertainty, \( \gamma \), is computed as:

\[
\gamma = k[(n - 2)u + (n - 3)u + \cdots + u + u']
\leq ku[(n - 2) + (n - 3) + \cdots + 1 + 1]
\leq ku[(n - 1) + (n - 2) + \cdots + 1 + 1]
\leq ku[1 + (n - 1)n/2]. \tag{2}
\]

Clearly, the uncertainty in the later case is greater than the uncertainty computed in Eq. (1). Hence, any deviation from the sequential itinerary will result in a larger uncertainty when considering a single agent in a homogeneous set of \( n \) resources. Hence, the sequential itinerary yields the least uncertainty under the stated conditions.

Under the same assumptions as before, we now incorporate \( m \) mobile agents. We are considering a hypothetical model wherein there is no overhead attached to the synchronization for the meeting of the agents. Each of the \( m \) agents is responsible for \( n/m \) resources. We assume that each of the \( m \) agents exchanges information with other agents after it traverses its corresponding resource partition. In what follows, we will show that an increase in the number of IMAs results in the diminishing of the difference between the best and the worst case uncertainties. We shall consider the scenario of the worst case knowledge base of the agent.

Each of the \( m \) agents is responsible for \( n/m \) resources. Based on a sequential itinerary, the associated uncertainty \( \gamma_m \) decreases as the number of agents increases. In practice, however, inter-agent communication for the exchange of information does constitute a significant overhead. This suggests the existence of an optimal number of agents for a given environment. Autonomous control of the agent population as a function of topology and system dynamics is one of the challenging research issues in the area of intelligent mobile agents. A detailed discussion of possible approaches is beyond the scope of this paper and will be presented elsewhere. Nevertheless, the above example illustrates the effects of multiple cooperating agents. In general, the multi-agent paradigm represents a powerful and scalable solution to many problems in DACS.

6. Agent security

It has been pointed out that one of the major research areas on software agents (Farmer et al., 1996; Berkovits
et al., 1998) and active networks (Ortiz, 1998) should be on agent security issues, including agent authentication and authorization before multi-agent systems can be fully deployed. We clearly advocate ubiquitous computing by providing an open computing infrastructure, which promotes the sharing of resources and services. Nevertheless, such an environment constitutes a significant investment that must be protected against destruction as well as corruption. In the context of agent based systems, the problem of providing the necessary system security is rather complex. Not only do we need to provide the necessary mechanisms to protect information that is transmitted between constituent hosts, we must develop a security infrastructure which can protect the entire system from the very mechanisms upon which the system is based.

The problem of providing the necessary level of security is exacerbated by the introduction of mobile code into the system. In addition to conventional information security, constituent hosts must be protected from fraudulent agents, agents must be protected from fraudulent hosts, and agents must be protected from other agents. The latter is particularly important in a multi-agent environment in which agents may alter their behavior as a function of information received from other agents.

Jamba is a secure framework for distributed computing systems (Havaldar and Wong, 1998). It provides features for digital certificates, access control, audit control, and system administration. In the following section, we will provide a brief overview of Jamba and highlight specific approaches for providing a secure framework for multi-agent systems.

7. System architecture of Jamba

The Jamba system is the set of components that provide the security services. The Jamba environment consists of the Jamba system, the Certifier and the application components, which is composed of producer and consumer agents.

In Fig. 7, the various types of interactions among the Jamba components are:
1. Request for security services (by both the consumer agent and the provider agent).
2. Logical communication path between the consumer agent and the provider agent.
3. Certificate distribution by the certifier to Jamba, consumer agent and provider agent.

In general, the Jamba system is a set of components that work together to provide a number of security services. One of the components, the secure facilitator, acts as a gateway for the other components. The secure facilitator is the only component that is ‘publicly’ accessible, and all security services are provided through this facilitator. When an application component wants to interact with Jamba, it registers with the secure facilitator. De-registration mechanism is also provided. The secure facilitator obtains the digital certificate from the certifier, and manages it on behalf of the Jamba system. The public interface of the secure facilitator also enables request and use of security services of the Jamba system. The only Jamba component with which it interacts directly is the Core Manager. The Core Manager interacts with and manages the rest of the components of the system (as shown in Fig. 8).

The Core Manager is the most crucial component of the Jamba system. It creates and manages all the core components. The core components include Principal authenticator, access decision, audit decision, domain organizer, vault. Each of these components provides certain specific functionality to the Jamba system. The core manager manages the relationship among them. It is
also the only Jamba component that is in direct contact with the secure facilitator. So, the secure facilitator accesses all the services through a well-defined interface of the core manager.

When the Jamba system initializes at startup, it requests the certifier for digital certificates. These certificates are used in interaction with consumer agent and provider agent components, and vouch for the validity of the services. The key pairs and the certificates are managed by the Jamba system itself. In fact, the management of key pairs, certificates and related security information is totally dependent on the component that owns them. The Jamba environment does not specify any specific mechanism in this regard, resulting in better flexibility.

The certifier plays the central role of a certificate authority. It generates and maintains certificates and related information. This utility component generates certificates for any component that requests for certificates to be generated and verified. All the components in the Jamba environment require and obtain a certificate (during initialization) before proceeding with any other interactions (especially with the Jamba system). During the registration of a component, attributes (privilege and identity) are assigned, credentials created and a current object corresponding to the registered component is stored in the vault. The creation of the attributes and the credentials is done by the principal authenticator component. The core manager oversees this series of actions, and the result is returned to the secure facilitator. In a similar fashion, all other services undergo a pattern of steps managed by the core manager.

The X.509 standard is used for the certificates. The Digital Signature Algorithm (DSA), which is a public key cryptosystem, is used to sign the certificates. This vouches for the validity of the certificate. The public key of the certifier is publicly available, and a public interface is exposed for communication with the certifier. The components obtain certificates from the certifier and use them for accessing services and communication. These components have a degree of trust in the certifier, which acts as an authority on its certificates. Any component can use the Jamba system for various security services. It has to interact with the certifier to obtain a certificate. This certificate needs to be submitted to the Jamba system for registration. Once registered, it can request services. A component has a Jamba-specific part and an task-specific part. The Jamba specific part is small and is provided as a set of utility classes. This eases system design and development. The task components in Jamba can act either as a provider or consumer of task-specific services. This approach is similar to that used by Kerberos (Treese, 1988) except that public key cryptosystem is used instead of user name and password. This is particularly important to use the public key technology as there are many administrative units in a large distributed autonomous cooperative systems.

The principal authenticator is responsible for generation of privileges and identities for a registered application component. The credentials, which are generated as part of the registration process, are stored in the vault (by the core manager). The principal authenticator uses information stored persistently in a database as a basis for assigning of privileges and identities. It communicates with the database processor with regard to accessing related information.

One of the most important services provided by the Jamba system is access control. When a consumer agent attempts to access certain task-specific services from a provider component, the provider agent requests the Jamba system (through secure facilitator) to validate the access rights of the consumer. The secure facilitator passes this request to the core manager, which communicates with the access decision component to make a decision. The access decision component makes the decision on allowing access based on the component identities of both the consumer agent and provider agent.

Last but not least, the vault is a passive storage component. It stores information such as component references, privileges, credentials, current objects and secure associations. This organization simplifies accessing information regarding a particular application component for operations such as access control. The vault itself contains a set of components. These components internally organize and manage different objects based on the component identity of the task component. The vault can be easily extended to store new objects, and is a flexible storage mechanism. All other core components of the Jamba system use the vault at various different stages.

The security framework provided by Jamba allows the system developers to build security components into the multi-agent systems, which include access control, audit control, authentication and system administration.

8. Summary

In this paper we have presented the role and the functions of Intelligent Mobile Agents (IMAs) in large distributed autonomous cooperative systems. Our discussion focused on the use of software agents and multi-agent systems as a new programming paradigm for the design, analysis and implementation of system level software as an integral part of a large autonomous distributed system. The field of autonomous software agent and multi-agent systems is a fast expanding area of research and development in recent years. It represents a melting pot of research results from related areas in computer science such as distributed computing,
artificial intelligence, software engineering, distributed databases, and programming languages. This new approach will provide a diverse range of software solutions to distributed computing at the system-level as well as at the application-level (Jenning et al., 1998).

There are several potential problems that must be overcome before the software agents and multi-agent systems can be fully deployed and used in an effective and efficient manner. There are a number of fundamental research and development issues that remain to be solved. These issues include, among others, the control and management of agent population, the representation of information in agents, efficient agent inter-communication and agent security.

While the multi-agent system represents a powerful and scalable paradigm for the design and implementation of DACS, new mechanisms will need to be developed that allow the system to control and manage the agent population as a function of the underlying topology and the system dynamics. To do so, research must emphasize on the study of basic principles and find efficient ways to deal with the creation and termination of autonomous agents in large distributed autonomous cooperative systems. For the deployment of multi-agent system as a fundamental paradigm in large DACS, the complexity of knowledge representation, communication between agents, and the representation of agents themselves becomes a critical issue. Hence, research efforts must encompass the study of interaction among agents in terms of the communication language used, mutual authentication, and low-level agent infrastructures. As the transfer of agents through the distributed system infrastructures may represent a significant resource overhead, strategies for compact knowledge representation must be developed.

Although there has been significant emphasis on the development of basic security mechanisms for the design and implementation of a secure agent infrastructure, the problem of system level security remains to be in need of further investigation. Particularly for ad-hoc distributed systems, for which there may not be any a priori membership in a particular public key infrastructure, the issues of authentication between agents, between agents and systems, and mutual trust deserve further investigation.

One of the most noteworthy advantages of the multi-agent paradigm is its ability to scale and adapt to the properties of the DACS. Through dynamic changes in behavior of individual agents or the agent population at large, an agent-based DACS will be able to sustain constant performance under a variety of possibly adverse conditions. Among other approaches, agents may react to changes in the system through adaptive itineraries, which alter the agent’s traversal of system components, adaptive heuristics, which alter the preference ordering in the decision sub-system, or adaptive inter-agent communication. The effects of system characteristics on the performance of multi-agent systems must be carefully investigated through experimental system modeling and performance analysis. Only when robust and scalable solutions to these problems are found will the full potential of intelligent mobile agent systems for large distributed autonomous cooperative systems be realized.

References


Dr. Johnny Wong is an Associate Professor of the Computer Science Department, Iowa State University at Ames, Iowa, USA. Before he came to Iowa State University, he was on the faculty of the Computer Science Department at the University of Sydney. His research interests include Operating Systems, Distributed Systems, Telecommunication Networks, Broadband-Integrated Services Digital Networks (B-ISDN), Concurrency Control and Recovery, Multimedia and Hypermedia Systems, Intelligent Multi-Agents Systems, Intrusion Detection.

He has been an investigator for research contracts with Telecom Australia from 1983 to 1986, studying the performance of network protocols of the ISDN. During this period, he has contributed to the study and evaluation of the communication architecture and protocols of ISDN. From 1989 to 1990, he was the Principal Investigator for a research contract with Microware Systems Corporation at Des Moines, Iowa. This involved the study of Coordinated Multimedia Communication in ISDN. In Summers 1991 and 1992, Dr. Wong was supported by IBM corporation in Rochester. While at IBM, he worked on the Distributed Computing Environment (DCE) for the Application Systems. This involved the integration of communication protocols and distributed database concepts. Dr. Wong is also involved in the Coordinated Multimedia System (COMS) in Courseware Matrix Software Project, funded by NSF Synthesis Coalition Project to enhance engineering education. From 1993 to 1996, he was working on a research project on a knowledge-based system for energy conservation education using multimedia communication technology, funded by the Iowa Energy Center. From 1995 to 1996, he was supported by the Ames Laboratory of the DOE, working in Middleware for Multidatabases systems.

Currently, he is working on Intelligent Multi-Agents for Intrusion Detection and Countermeasures funded by the office of INFOSEC Research and Technology of the National Security Agency (NSA) and Database Generating and X-Ray Displaying on the World Wide Web Applications funded by Mayo Foundation.

Dr. Armin R. Mikler received his Diploma in Informatik from fachhochschule Darmstadt, Germany in 1988. He received his M.S. and Ph.D. (Computer Science) from Iowa State University in 1990 and 1995, respectively. From 1995 to 1997, he worked as a postdoctoral research associate in the Scalable Computing Laboratory at the Ames Laboratory, USDOE. Since 1997, Dr. Mikler has been an assistant professor in the Department of Computer Sciences at the University of North Texas. His research interests include: Intelligent Traffic Management, Coordination of Intelligent Mobile Agents, Decision-making in Distributed Environments, and Cluster Computing.