# Experiences and Future Challenges in Mobile Agent Programming

Anand R. Tripathi, Tanvir Ahmed, Neeran M. Karnik

*Abstract*—

**The current research in mobile agent systems has demonstrated the utility of this paradigm in building a wide range of distributed applications and systems. We present in this paper the promising areas for mobile agent based applications. These range from distributed collaborations, to network management, to personal assistants over the Internet. There are many mobile agent programming platforms available today. A large cross-section of these are based on Java. The design of a mobile agent programming platform requires addressing several important problems. In this paper we discuss these design issues, and show how some of the contemporary mobile agent systems have addressed these in their designs. We focus particularly on the Ajanta system, which is a good representative of modern Java-based agent programming platforms. We present here two applications developed using Ajanta, and based on our experiences we present the future challenges for research in this field.**

**Keywords:** *Internet programming, Internet agents, mobile agents, mobile code, mobile objects, distributed computing, security, fault tolerance*

## I. Introduction

A *mobile agent* is a program which represents a user in a network and is capable of migrating autonomously from node to node, performing computations on behalf of that user. It represents an activity whose execution state is preserved on migration, and the agent's execution resumes with this state after it is transported to the destination node.

The main advantages of the mobile agent paradigm lie in its ability to move client code and computation to remote server resources, and in permitting increased asynchrony in client-server interactions[15]. By moving computation close to the needed resources, this paradigm can reduce network communication, thus reducing bandwidth requirements and latency. Agents can be used for information searching, filtering and retrieval, or for electronic commerce on the Web, and for system administration tasks.

The current research interest in mobile agent technology is largely driven by the wide-spread commercial utility of the Internet in our daily life, and the general need for ubiquitous access to computing resources and information services. It has been propelled by the emergence of Java as a universally available mobile code technology[31]. Mobile code technology allows transportation and execution of program code across networks of heterogeneous machines; the program execution is guaranteed to conform to the prescribed semantics.

Department of Computer Science, University of Minnesota, Minneapolis MN 55455

Now with IBM India Research Lab, New Delhi. Email: kneeran@in.ibm.com

In the early 1990s, General Magic developed the Telescript[34] language for mobile agent based network computing. It was followed by research systems like Tacoma[18] and Agent Tcl[13]. These two systems were based on scripting languages such as a Perl and Tcl. The primary focus of these systems was to demonstrate the agent paradigm. They lacked several important and desirable features in regard to security and programming abstractions. The emergence of Java led to the next generation of systems that leveraged its object-orientation and security features. In addition to code mobility, Java also provides a security architecture that helps in constructing suitable solutions for the security problems in mobile agent systems. Aglets[20], Voyager[25], Concordia[24] , Mole[28], Sumatra[27], and Ara[26] are examples of the first-generation Java-based mobile agent systems.

Many of these early mobile agent systems demonstrated the basic utility and functionality of the agent paradigm. Some of them (or their later versions), and other subsequently developed systems have addressed a broader class of issues related to mobility, security, and robustness. However, the field of mobile agent programming is still in its early stages of development due to several challenging obstacles, which need to be addressed in order to realize the full benefits of this paradigm through the deployment of a wide range of agent-based applications. The various research systems and prototypes developed in the recent years have helped the research community in building a better understanding of the solutions needed in practical systems for mobile agent programming.

The focus of this paper is on the current state of experiences in designing and utilizing mobile agent programming platforms, and the future research challenages that pose significant obstacles to widespread use of this paradigm. We first describe, in Section 2, the various potential applications of the mobile agent paradigm. We then describe the salient features of today's mobile agent programming platforms, with specific focus on the capabilities of the Ajanta system. In Section 3, we describe our experiences in using the Ajanta system. The focus of Section 4 is on the future challenges for research and development in this area.

## II. Mobile Agent Based Applications

The mobile agents technology is an alternative implementation choice among distributed object technologies. However, the additional facilities provided by this technology make mobile agents suitable for several application areas. Some promising areas of mobile agent based applications and how mobile agents facilitate them are discussed

below.

Internet-wide collaborative systems, especially workflow systems, are one of the emerging application areas as Internet is becoming the primary computing environment. Mobile agents can provide the programming abstraction for such applications for several reasons. They can asynchronously execute coordination actions. Mobile agent based programming paradigm can be exploited to implement workflow systems in disconnected environments as mobile agents can carry along all the application specific data and code. The mobility of a shared object can be exploited, when implemented as a mobile agent, by moving it from one participant to another at various stages of a workflow. Moreover, if a user's interaction environment in a workflow is implemented as a collection of mobile agents, it is possible for a user to physically move to a different node by simply directing its agents to migrate to that node. We have implemented a mobile agent based collaborative middleware which is used for implementing workflow applications [32]. This system is described in more detail in Section 4.

In network management, stationary software agents like SNMP agents are familiar. However, mobile agents based implementation of network management and monitoring systems has several advantages. Mobile agents can update protocols or interfaces of networked components by encapsulating the new protocols or interfaces and migrating to the corresponding nodes. Mobile agents can update administrative policies dynamically and autonomously to reflect the network changes. Moreover, mobile agents can collaborate following an administrative hierarchy among themselves, which provides a stronger integration among networked tools and components. An example of mobile agent based network monitoring application can be intrusion detection system. Mobile agents are suitable for such applications as they can adapt dynamically with network changes like network partitioning and can collaborate with other agents to monitor the network.

Mobile agents can be utilized in different areas of telecommunication networks. Mobile agent are used for network traffic control and resource management [17], where agents are launched in different nodes of a network. These network nodes are classified as service switching points, service control points, and intelligent peripherals, where agents reside and participate with some strategy to control network resources. A mobile agent based framework for intelligent network and how such an architecture can be used for multimedia and other services are discussed in [4].

In the emerging computing environment of thin clients or resource limited devices, also known as ubiquitous or pervasive computing, mobile agents can be deployed as proxies to reduce system resource consumptions. For example, a personalized mobile agent can be launched from a handheld palm to the connecting server, where the agent can filter web traffic, email, or any other application specific data. Mobile agents provide a better solution to program these environments as they can navigate and sense the network, discover resources, and collaborate among themselves. Network service subscription and service configuration in the context of a universal mobile telecommunication system based on mobile agents are discussed in [14].

Research in the area of active networking[30], [1] has also relied on the use of mobile code to dynamically program the functions of network layer components. The problems addressed there are very similar to those in building a mobile agent programming infrastructure for open systems. However, the mobile agent paradigm represents a more general model and facility for distributed applications than active network systems whose functions tend to be mostly confined to network level components. Mobile agent technology provides a rich set of abstractions for the needs in active network research.

In mobile or wireless computing, mobile agents can play an important role as the solution to deal with the difficulties introduced by users' mobility can be easily abstracted in a mobile agent based middleware. Mobile agents can encapsulate the user's computing environment including any networked resources like filesystem. These agents can be redirected when a user moves from a node to another in the network. This migration of code and data provides a flexible computing model for loosely connected or wireless network as network resources are smartly distributed.

Mobile agents are used as personal assistants where agents represent users and can migrate to a large pool of service providers to get the best value. In e-commerce, these personal assistants are used in a range of applications, from buying goods and services online to stock market monitoring. Buyer and seller agents can act and negotiate on behalf of their owners, which accomplishes a better value for the participants. With mobile agent based personal assistants, buyers and seller can easily change their marketing strategy encapsulated in the agents and dispatch them on the web to meet their specific criteria. Moreover, mobile agents based personal assistants can build communities of special interest groups on the web.

As the web is getting larger and more independently maintained, it is difficult to get frequently changing information in a timely manner and information which are not indexed. This is due to the fact that indexing is done periodically. With mobile-agent based approach, agents can be launched to the source of the data to index and report. This will result in a finer grain model of information retrieval than the existing solutions. Agents for distributed information retrievals are investigated by others [5], and we present an example web search application in Section 4.

Another promising area is mobile agents based parallel processing as an alternative to existing metacomputing tools. In a mobile agents based distributed processing environment, cooperating agents can partition tasks and migrate to nodes with unused computing resources. This kind of computing model can also utilize unused personal computing resources. Mobile agents for distributed problem solving of large scale scientific simulation is presented in [9].

## III. Features of Mobile Agent Programming Platforms

The design of a general purpose mobile agent programming infrastructure for open systems such as the Internet requires design choices and solutions to problems in several areas. These include definition of a computation model and related programming primitives, design of a naming scheme and a reliable and secure name service, mechanisms for agent mobility, security mechanisms to protect host resources and agents, inter-agent communication mechanisms, primitives for fault-tolerance and remote agent control, and support for program development and debugging tools [22]. We discuss below the various design options in regard to these aspects and how various mobile agent platforms and Ajanta address them.

Ajanta[1][33], [21] is a Java-based framework for programming *mobile agent* based applications on the Internet. We developed the Ajanta system as a research infrastructure for secure distributed programming using mobile autonomous objects. In Ajanta, the mobile agent implementation is based on the generic concept of a *mobile object*[19].

In order to support an agent based application, a host provide execution environment for visiting agents. Such facilities are typically called *places* or *agent servers*. An agent server creates a confined execution environment for visiting agents and allows the local resource owner to grant access of its resources to agents in a selective manner. It provides primitives for agents to migrate to other servers, communicate with each other, query their environment, etc.

### A. Agent Mobility

There are two models for supporting agent mobility, In the *weak mobility* model, on migration , the agent state essentially consists of the agent's program-defined data structures, whereas the *strong mobility* model captures the agent's state at the level of the underlying thread or process[12], [3]. With weak mobility, an agent's migration is possible only at specific points in the agent's code, and typically a migration is explicitly requested in the agent's code. The strong mobility model allows an agent to be migrated at any point in its execution. This model is certainly useful if agents need to be moved at unpredictable points in time for fault-tolerance or load-balancing. Several systems – such as Agent Tcl, Sumatra, Ara, Nomads[29] – have supported strong mobility. In the context of Java based systems, this support has generally required development of specialized virtual machines for mobile code execution. In order to remain compatible with the standard distribution of the Java virtual machine, a majority of the current mobile agent programming systems have based their designs on the weak mobility model. Ajanta and several other systems, such as Mole, Aglets, Concordia, Voyager, SOMA [7] fall in this category. Today, it is generally felt that program-controlled migration under weak mobility suffices for majority of the applications.

[1]See http://www.cs.umn.edu/Ajanta for more documentation and information on the availability of a public domain version of this system.

In agent migration, the most difficult problems are open files and network communication channels, as faced in supporting process migration in distributed systems [8]. To keep the design simple and efficient, most agent programming systems do not support resumption of sessions, dealing with open files or communication channels, on migration. This avoids dependencies on remote nodes. Besides, under program-controlled mobility, one can properly close any open sessions before migration, and reopen them after migration.

If an agent is multithreaded, then under the weak mobility model, the programmer needs to take special care when making explicit requests for migration in the agent's code. Problems can arise if one thread requests migration when other threads have not yet completed their tasks. Also, one needs to prevent a situation when two threads issue migration requests to move to different hosts. For Java based systems, one should also keep in mind that the Java runtime implicitly creates threads to support GUI and RMI. Therefore, even when an agent programming system does not explicitly support a multithreaded model for its agents, the programmer must be cognizant of such implicitly created threads. Agents in Ajanta could be multithreaded either implicitly or explicitly. Therefore, when requesting migration, it is the programmer's responsibility to ensure that all other threads have either terminated or reached a state when it is safe to terminate them and migrate the agent.

There are several approaches for transporting an agent's code on its migration to another node. One approach is that all classes required by an agent are transported as a part of the agent transfer protocol. This approach makes agent transport *heavyweight*, since an agent may be composed of a large number of classes, whereas only a few are actually needed for the task to be performed at a particular server. However, this has the advantage that once the agent is transferred, no further remote communication is needed for code transfer during the agent's execution at that server. Another approach, which is adopted in Ajanta, is to obtain the agent's classes *on demand* from a designated code-base server during the agent's execution. It makes agent transfer *lightweight* as it transfers only those classes that are needed during the execution. This was the primary reason for adopting this approach in Ajanta. There are certainly some disadvantages of this approach. This approach makes agent's execution slow, and it is not suitable when an agent have to execute in a disconnected environment.

### B. Naming scheme and Name Service Architecture

A global naming scheme and name service is needed for locating resources, specifying agent servers for agent migration, and establishing inter-agent communication. A naming scheme defines a name-space, which could be hierarchically structured. A location-independent naming scheme is particularly attractive for mobile agent systems, because the name of an agent does not change on migration. This simplifies the programmer's task significantly, because a

program can be written without regard to the current locations of various entities referenced by it during its execution. A name service is needed to support mechanisms for mapping a resource name to its physical address. It could also contain information about the type of the resource represented by a name and the protocol used for accessing it, as in URLs. Alternatively, a system could adopt different naming schemes and name services for different kinds of resources, thus optimizing the name resolution protocols according to the resource types. In the Ajanta design, we have adopted a uniform naming scheme based on the URN model for all entity types; however, subclasses of the base name registry entry distinguish between different kinds of resources for directory management and name resolution.

### C. Security and Protection Issues

Security is one of the most important requirements of an agent programming environment. Service providers need protection mechanisms to enforce the desired security policies for preventing unauthorized or malicious agents from using, destroying or altering the server's resources, or disrupting its normal functioning by mounting "denial of service" attacks.

An agent needs to be authenticated to verify the ownership of its user, on whose behalf it is to be granted access to a host's resources. For this, an agent needs to carry a set of credentials, identifying its owner and the privileges delegated to it by the owner. Mechanisms are needed for an agent server to verify an agent's credentials to detect any tampering or replay attacks. Access control policies for server resources are required to identify access rights of an agent, based on its credentials. Such policies need to define access control for application-defined resources as well as system level resources such as files, disk storage, I/O devices, network ports, etc. Ajanta allows specification of policies for an agents' access to files and network resources.

Some systems, such as Aglets, have also incorporated definition of security policies based on authorship of the agent's code. In Ajanta, all access control policies are based solely on the agent's owner and do not take code authorship into consideration. This is to keep the security policies simple and easy to understand.

An agent server may also need mechanisms to enforce policies for resource consumption limits, such as usage of disk storage, CPU time, number of threads, number of windows, network bandwidth, etc. The Nomads system[29] provides an explicit mechanism for this purpose. However, this necessitates a custom-designed Java virtual machine. SOMA [7] has used JVM Profiler Interface (JVMPI) to build APIs for metering resource usage. The Ajanta design has concerned itself mainly with the resource access control mechanisms. For this, it supports a proxy based resource access mechanism.

An agent may need to be protected from malicious servers or intermediate nodes on its travel path, because it may carry sensitive information about the user it represents. Such information could be read or modified by unauthorized servers. For example, agent's credentials could be modified or stolen, or some critical information, such as its itinerary, task plans, preferences, etc. could be altered maliciously. An agent may also need to carry some confidential information intended only for some specific hosts on its travel path. One also needs mechanisms for preserving the integrity or secrecy of the information collected by an agent during its visit to various hosts. In this regard the Ajanta design presents an abstraction for a secure *append-only container*. Most of the other systems, such as Concordia and Grasshopper[23], protect an agent state only during transfer by using secure communication channels and message authentication codes. These systems do not address the problem of secure collection of data by an agent from the servers it visits, or protection of an agent's credentials from theft or tampering.

Security of the name service is essential for secure operations of a mobile agent platform. If the name registry entries are not protected from unauthorized modifications, an attacker can delete a name, or change the contents of a registry entry, such as an entity's location or its public keys. The name-spaces assigned to various principals in the system also need to be protected. A user should not be allowed to create names in the name-spaces of other users, unless properly authorized. Otherwise, a potential attacker could create and register a distinguished name in another user's name-space, thus acting as an impostor. Ajanta's name service is designed with such considerations.

### D. Computation Model and Programming Primitives

A simple event driven model for agent computation has been adopted in large number of mobile agent platforms. Aglets design was mainly influenced by the applet model; handlers are defined for different types of events such as migration, dispatch, arrival at a server, etc. In Aglets and Grasshopper, on arrival at a host, one specific method of the agent is executed as the entry point; this method determines the actions to be executed on the next hop based on the state captured at the previous host. In contrast, in Ajanta any of the public methods of the agent object can be specified as the entry point on arrival at a host. This model is also supported by Concordia and Voyager. This is certainly a more flexible model than the single-entry point approach.

As a higher level programming abstraction, Ajanta supports the itinerary concept. Migration control is abstracted into a single object, rather than scattered at various points in an agent's code. A novel aspect of this facility is that itineraries can be programmed using composable migration patterns. This permits a composition of a complex travel plan from some basic patterns. The notion of itineraries has also been supported by other mobile agent systems – such as Aglets, Mole, and Concordia. The concept of migration patterns has also been used in Aglets [16], [2], but the patterns are described in terms of single hops.

Only a few agent programming system allow an agent to create child agents, which is useful for executing a set of subtasks in parallel. For example, in an information search or data mining application, an agent may create multiple

child agents to visit different sites concurrently. In Ajanta, creation of child agents is supported using a pattern class called *fork*. It also supports synchronization of child agents using a pattern called *fork-join*. There are several issues involved in supporting such a facility. One is to provide mechanisms for such agents to synchronize and coordinate their activities. Security considerations are also important here: an agent should not be allowed to create any arbitrary number of agents at a remote site.

### *E. Inter-agent Communication*

The design challenges for inter-agent communication mechanisms arise due to the mobility of the agents. There are several design choices: connection-oriented communication (such as TCP/IP), connection-less communication (RMI, RPC, or CORBA-IIOP), or indirect communication – not requiring the names of the communication partners – based on event-notification and shared group/meeting objects (as in Concordia and Mole) or global shared tuple-spaces based on the Linda model [6]. Several systems (such as Aglets, Grasshopper, Voyager) have supported synchronous, asynchronous, one-way, and future-based communication models.

In TCP/IP or RMI based communication, agents need to know each other's names in order to establish communication. Connection-oriented schemes raise the issue of session disruption due to a participating agent's migration. In comparison, RMI based connection-less model throws an exception when a remote invocation fails due to the migration of the server agent; the client agent only need to re-execute the lookup and binding protocol to re-establish communication with the migrated agent at its new location. This approach is taken by the Ajanta design. Support for mutual authentication of mobile agents is a challenging problem, because a *challenge-response* based protocol is difficult to use as agents cannot carry their private keys when executing in foreign domains.

In Aglets, communication takes place through the exchange of *Message* objects – an agent cannot simply invoke an arbitrary public method of another agent. In Concordia, agents do not communicate directly with each other. Communication is mainly supported in the form of event notifications, which are communicated based on a subscription oriented model. Agents can also communicate and collaborate by communicating data indirectly through group objects. Concordia supports encryption of communication using SSL, and authentication of users and groups using symmetric keys.

Security is an important concern in providing remote communication facilities to visiting agents. An agent could copy or transfer information in an unauthorized manner, or it could gain access to protected resources inside a firewall. It can also launch denial of service attacks by creating a large number of ports and using excessive amount of network bandwidth. It is also important to protect an agent from other malicious agents. For this, support for encrypted and authenticated inter-agent communication is important. Such support is provided in varying degrees by the current mobile agent programming systems.

### *F. Fault Tolerance and Agent Control*

A number of mobile agent platforms provide support for persistence (e.g., Aglets, Concordia, Grasshopper); this allows an agent to deactivate itself and store its state in the stable storage. This state can be used for system level recovery from crashes. Beyond this facility, only a couple of systems, Mole and Tacoma, have devised system level mechanisms for failure recovery in mobile agent executions. The Mole project has investigated mechanisms for atomic transactions for mobile agents, and Tacoma has developed an approach based on the *rear-guard* concept, where an agent's state is checkpointed at its previously visited servers and used for failure recovery.

In most systems, there is little support for features that are required for robustness, such as failure detection and recovery. Ajanta provides a unique mechanism for application-level exception handling in mobile agent programs. It presents a *guardian* [33] based mechanism that allows the programmer to perform recovery actions for exceptions that are encountered but not handled by a remote agent. This facility also helps us in debugging an agent program. For remote control of agents, Ajanta provides a secure mechanism for recalling or terminating its remote agents. Aglets also supports recalling of an agent from a remote location. However, it does not enforce any security restrictions in executing a recall operation, which makes an Aglet application vulnerable to attacks.

### *G. Interoperability Standards*

OMG developed Mobile Agent System Interoperability Facility (MASIF), to address the interoperability of different mobile agent platforms, and also to support the integration of this new paradigm with legacy applications using CORBA. Using this facility, mobile agent platforms can communicate with each other using CORBA primitives. Its *MAFFinder* interface provides methods for agent registry and database management functions. The *MAFAgentSystem* provides methods for agent management, transfer, etc. Grasshopper is the only system that has provided MASIF support, which is primarily motivated due to its commercial interests. FIPA (Foundation for Interoperability of Physical Agents) [10], [11] has also developed specifications for external behavior of agents and interoperability with other agents, non-agent software, humans and the physical world. FIPA's reference model includes *Directory Facilitator*, *Agent Management System*, and *Agent Communication Channel*, which are specific types of agents supporting agent management and reside on every agent platform. Grasshopper provides support for FIPA using *add-on* stationary agent objects to support these functionalities by interfacing with its native facilities.

Ajanta does not impose restriction on agents' external behavior and only provides system level support for agent execution. It can interoperate with any agent communication language. For agent naming, Ajanta matches FIPA specification which require unique GUID for agents. In

Ajanta, some of the required Agent Management System functionalities are encapsulated in the generic agent server class. As done by the Grasshopper system, suitable FIPA adapters can be added into the Ajanta framework.

## IV. Experiences with Ajanta System

We have developed a number of agent-based applications using the facilities of the Ajanta system, with the primary goal to test and demonstrate its functional capabilities. We now describe two of these applications.

### A. Mobile Agent based Web Search

Our web search facility in built on top of a middleware implemented using the agent paradigm for implementing a system for sharing files over the Internet. This middleware system allows users to selectively share files across a network with other users. Each user runs a `FileServer`, which is an agent server customized with a `FileSystem` resource. This resource provides visiting agents with access to a user-specified 'root' directory on the local file system (and to all underlying files and directories). Its interface includes basic primitives that an agent can invoke like *fetch*, *deposit*, *transfer*, and *search*. The user can control which agents have access to the files using a simple access control list checked against visiting agent's owner's privileges.

The web search facility dispatches an agent to a remote user's file access server and perform full-text searches on the files in the user's web directory. The file server first constructs an index for the user's web directory and stores the index in the shared global file system. This allows visiting agents to search the web index of the user with any desired combination of keywords. The client can also program the agent to look for only those files which have more than a specified number of occurrences of the keywords, or the files that have been recently modified. The search agent brings back the URLs of the documents that were found to meet the search specifications. A utility program at the client side suitably formats the search result URLs as hypertext links in an HTML document, which can then be viewed through a browser.

### B. Internet Workflow System

We have conducted some proof-of-concept experiments to investigate the use of mobile agents in implementing workflow environments[32]. Our approach is based on constructing a distributed collaboration system starting from its XML specification in terms of various participants' roles, access rights based on roles, shared objects, operations, and collaboration constraints. The specifications are integrated with an agent-based distributed middleware for collaboration implementation. This middleware enforces coordination and security constraints specified in the XML description of the plan. Moreover, shared objects in the collaboration environment are distributed to the participants according to their role-specific views of the plan.

The overall approach of constructing a collaborative system [32] involves three steps. The first step is to devise a schema in the form of XML Document Type Definition (DTD), which provides constructs for defining roles, shared objects, and operations associated with a workflow task. It also provides constructs for associating privileges with roles and participants, coordination actions and workflow constraints with operations. The semantics of objects and their actual definitions are not described but left to the applications. The second step is the description of a workflow plan, using XML, in conformance with the DTD. The designer of the workflow plan, whom we refer to as the *convener*, is responsible for preparing this description. In the third step, the XML specification is interfaced with a distributed agent-based middleware. This system provides to each user an agent-based coordinator on his system. We refer to this as the *User Coordination Interface (UCI)*. A UCI maintains with it copies of the shared objects that are required as per the user's roles and it provides suitable interfaces to its user.

We have implemented an Internet-wide workflow authoring system based on this generic mobile agent based middleware, where several users participate according to their roles to prepare a document in different stages of a workflow. We are able to leverage the Ajanta system for supporting the security requirements prescribed in the workflow specifications by using Ajanta's security related components like authentication, public key maintenance, access control, host resource protection, and cryptographic services. However, the convener of a workflow system only needs to specify the XML plan without being concerned about underlying agent-based implementation layer. Ajanta framework can also provide simple abstractions for addressing many other challenging problems in workflow systems like exception handling, activity tracking, statistical data collection, and workflow object migration related issues. With our approach, an Internet workflow system can be easily reconfigured through XML manipulation compared to other approaches for building workflow systems.

Based on our experience in developing these and several other applications using Ajanta and considering Ajanta as a good representative of modern mobile agent programming platforms, we can make the following observations on mobile agent programming.

- The mobile agent paradigm can be effectively used as an implementation mechanism, just like RPC and message passing, for building distributed applications.
- Mobile agents provide easy abstraction for code mobility to encapsulate distributed computing. As an additional level of abstraction to existing object technologies, mobile agents are a powerful facility for distributed computing.
- Secure mobile agent programming platforms, such as Ajanta, provide a more secure computing environment than existing RPC technology, where in many cases security is traded for robustness and performance. With mobile agents, after authentication, code and data migrate to the destination host, eliminating the need of maintaining secure channels among distributed objects.
- Our experiments in distributed collaborations demon-

strated that a high level markup language, such as XML, can be effectively used to specify an agent-based distributed computation. This kind of approach in programming agent applications can hide many complexities of mobile agent programming from application programmers.

• The exception handling model introduced in the Ajanta system was found to be very beneficial in debugging during application development. In this model, an agent encountering an unhandled exception is colocated with an application defined exception handler called *guardian*. However, more sophisticated tools are needed for developing and debugging medium to large-scale agent-based applications.

## V. Conclusions and Future Challenges

In this paper we have discussed the various design issues of mobile agent programming platforms. We have discussed how Ajanta and other mobile agent systems address these issues. The large number of mobile agent programming systems developed in the recent years have demonstrated the capabilities of this new paradigm for distributed programming. The spectrum of problems related to the development of a mobile agent programming platform is broad, and various systems have made their unique and novel contributions in different aspects of this spectrum.

The field of mobile agent programming is still in the state of infancy. The full potential of this technology is yet to be realized for real-world applications, as several problems yet remain to be fully addressed.

• Security and fault-tolerance remain to be the most challenging problems in this field. Many of the currently available Java-based mobile agent platforms are able to provide adequate support for protecting host resources, including metering and controlling the resource usage levels. However, there are several areas related to security that demand further investigations. The most important of these is the specification and verification of security policies for access control for host resources and delegation of rights to an agent by its owner. Another area that needs further investigation is the development of distributed trust models for mobile agent systems.

• Fault-tolerance mechanisms are required at both the application level and the system level. The system level mechanisms hide the effects of failures of the underlying system components such as node crashes, links failures etc. Application level recovery mechanisms allow an agent application to perform exception handling when some anticipated abnormal conditions arise. The Ajanta system provides a basic model and mechanism for exception handling. More work is needed to develop a methodology, and possibly some design patterns, for handling exceptions in agent applications with multiple agents. In general, there is a lack of experience in this field in regard to the use and evaluation of various approaches to fault-tolerance in agent programs.

• There is a lack of experience with large-scale agent-based applications. Most of the existing mobile agent applications are generally "small" in size, requiring at most a few tens of agents. The lack of good program development and debugging tools has certainly been a reason for this.

• In most of the current mobile agent platforms, support for managing and coordinating agent groups is not present. For developing large-scale agent applications, such as agent-based monitoring of a large network, this kind of support is crucial. The development of any models and mechanisms in this directions needs to undertaken with complete consideration of the underlying security concerns. For example, security policies may prohibit communication between two agents while any of one of them is located at some untrusted host. Also, support is needed for mutual authentication of mobile agents, keeping in mind that agents cannot carry with them any secret keys.

• The agent-to-agent communication is mainly investigated in reference to knowledge exchange, based on KQML or XML. However, a well defined interface for coordination among agents' for administrative tasks is lacking. A high level specification of coordination actions supported by Linda like model is needed for multi-agent systems.

• For an agent's tasks specification, description of the agent's intentions may be desirable than the method level description used in the itinerary. Support for intention-oriented specification of agent tasks using a high level language like XML, and resources-discovery at remote hosts based on such intention specifications is needed for large scale agent based applications.

## References


[1] D. Scott Alexander, William A. Arbaugh, Angelos D. Keromytis, and Jonathan M. Smith. A Secure Active Network Environment Architecture. *IEEE Network*, May 1997.

[2] Yariv Aridor and Danny B. Lange. Agent Design Patterns: Elements of Agent Application Design. In *Second International Conference on Autonomous Agents*, May 1998. Available at http://www.acm.org/~danny/ag.pdf.

[3] Joachim Baumann, Fritz Hohl, Kurt Rothermel, and Markus Straßer. Mole - Concepts of a Mobile Agent System, August 1997. http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole.html.

[4] M. Breugst, L. Faglia, O. Pyrovolakis, I. S. Venieris, and F.Zizza. Towards mobile service agents within an advanced broadband IN environment. *Computer Networks*, 31(19):2037–2052, 1999.

[5] Brian Brewington, Robert Gray, Katsuhiro Moizumi, David Kotz, George Cybenko, and Daniela Rus. Information retrieval. In Matthias Klusch, editor, *Mobile Agents for Distributed Intelligent Information Agents*, chapter 15. Springer-Verlag, 1999.

[6] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli. Mobile-Agent Coordination Models for Internet Applications. *IEEE Computer*, pages 82–89, February 2000.

[7] Antonio Corradi, Marco Cremonini, Rebecca Montanari, and Cesare Stefanelli. Mobile Agents Integrity and Electronic Commerce Applications. *Information Systems*, 24(6):519–533, 1999.

[8] Fred Douglis and John Ousterhout. Transparent Process Migration: Design Alternatives and the Sprite Implementation. *Software – Practice and Experience*, 21(8):757–785, August 1991.

[9] Tzvetan Drashansky, Elias N. Houstis, Naren Ramakrishnan, and John R. Rice. Networked Agents for Scientific Computing. *Communications of the ACM*, 42(3):48–54, 1999.

[10] FIPA (Foundation for Intelligent Physical Agents) 97 Specification . Available at URL http://drogo.cselt.it/fipa/spec/fipa97, 1997.

[11] FIPA (Foundation for Intelligent Physical Agents) 98 specification . Available at URL http://drogo.cselt.it/fipa/spec/fipa98, 1998.

[12] Alfonso Fuggetta, Gian Pietro Picco, and Giovanni Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, May 1998.

[13] Robert S. Gray. Agent Tcl: A flexible and secure mobile-agent system. In *Proceedings of the Fourth Annual Tcl/Tk Workshop (TCL 96)*, July 1996.

[14] L. Hagen, J. Mauersberger, and C. Weckerle. Mobile agent based service subscription and customization using the UMTS virtual home environment. *Computer Networks*, 31(19):2063–2078, 1999.

[15] Colin G. Harrison, David M. Chess, and Aaron Kershenbaum. Mobile Agents: Are they a good idea? Technical report, IBM Research Division, T.J.Watson Research Center, March 1995. Available at URL http://www.research.ibm.com/massdist/mobag.ps.

[16] IBM. JMT (Java-based Moderator Templates) Specification - Alpha3. Available at URL http://www.trl.ibm.co.jp/aglets/jmt/index.html, 1998.

[17] Brendan Jennings, Rob Brennan, Rune Gustavsson, Robert Feldt, Jeremy Pitt, Konstantinos Prouskas, and Joachim Quantz. FIPA-compliant agents for real-time control of Intelligent Network traffic. *Computer Networks*, 31(19):2017–2036, 1999.

[18] Dag Johansen, Robbert van Renesse, and Fred B. Schneider. Operating System Support for Mobile Agents. In *Proceedings of the fifth IEEE Workshop on Hot Topics in Operating Systems (HotOS-V)*, pages 42–45, May 1995.

[19] Eric Jul, Henry Levy, Norman Hutchinson, and Andrew Black. Fine-Grained Mobility in the Emerald System. *ACM Transactions on Computer Systems*, 6(1):109–133, February 1988.

[20] Gunter Karjoth, Danny Lange, and Mitsuru Oshima. A Security Model for Aglets. *IEEE Internet Computing*, pages 68–77, July-August 1997.

[21] Neeran Karnik and Anand Tripathi. A Security Architecture for Mobile Agents in Ajanta. In *Proceedings of the 20th IEEE International Conference on Distributed Computing Systems*, pages 402–409, April 2000.

[22] Neeran M. Karnik and Anand R. Tripathi. Design Issues in Mobile Agent Programming Systems. *IEEE Concurrency*, 6(6):52–61, July–September 1998.

[23] Thomas Magedanz, Christoph Baumer, Markus Breugst, and Sang Choy. Grasshopper – A Universal Agent Platform Based on OSF MASIF and FIPA Standards. http://www.ikv.de/products/grasshopper.

[24] Mitsubishi Electric. Concordia: An Infrastructure for Collaborating Mobile Agents. In *Proceedings of the 1st International Workshop on Mobile Agents (MA '97)*, April 1997.

[25] ObjectSpace. ObjectSpace Voyager Core Package Technical Overview. Technical report, ObjectSpace, Inc., July 1997. Available at http://www.objectspace.com/.

[26] Holger Peine and Torsten Stolpmann. The Architecture of the Ara Platform for Mobile Agents. In *Proceedings of the First International Workshop on Mobile Agents (MA'97)*, Berlin, Germany, April 1997. Springer Verlag, LNCS #1219.

[27] M. Ranganathan, Anurag Acharya, Shamik Sharma, and Joel Saltz. Network-aware Mobile Programs. In *Proceedings of USENIX '97*, Winter 1997.

[28] Markus Straßer, Joachim Baumann, and Fritz Hohl. Mole - A Java Based Mobile Agent System. In *Proceedings of the 2nd ECOOP Workshop on Mobile Object Systems*, 1996.

[29] Niranjan Suri, Jeffrey M. Bradshaw, Maggie R. Breedy, Paul T. Groth, and Gregory A. Hill. Strong Mobility and Fine-Grained Resource Control in NOMADS. In *to appear in the Proceedings of the Second International Symposium on Agent Systems and Applications and the Third International Symposium on Mobile Agent Systems (ASA/MA'2000)*, September 2000.

[30] David Tennenhouse, Jonathan M. Smith, W. David Sincoskie, and David J. Wetherall. A Survey of Active Network Research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.

[31] Tommy Thorn. Programming Languages for Mobile Code. *ACM Computing Surveys*, 29(3):213–239, September 1997.

[32] Anand Tripathi, Tanvir Ahmed, Vineet Kakani, and Shrematie Jaman. Distributed Collaborations using Network Mobile Agents . In *2nd International Symposium on Agent Systems and Applcations/ 4th International Symposium on Mobile Agents*, September 2000.

[33] Anand Tripathi, Neeran Karnik, Manish Vora, Tanvir Ahmed, and Ram Singh. Mobile Agent Programming in Ajanta. In *Proceedings of the 19th International Conference on Distributed Computing Systems*, May 1999.

[34] James E. White. Mobile Agents. Technical report, General Magic, October 1995.