Mobile Software Agents: An Overview

Vu Anh Pham and Ahmed Karmouch
University of Ottawa, Ontario

ABSTRACT The anticipated increase in popular use of the Internet will create more opportunities in distance learning, electronic commerce, and multimedia communication, but it will also create more challenges in organizing information and facilitating its efficient retrieval. From the network perspective, there will be additional challenges and problems in meeting bandwidth requirements and network management. Many researchers believed that the mobile agent paradigm (mobile object) could propose several attractive solutions to deal with such challenges and problems. A number of mobile agent systems have been designed and implemented in academic institutions and commercial firms. However, few applications were found to take advantage of the mobile agent. Among the hurdles facing this emerging paradigm are concerns about security requirements and efficient resource management. This article introduces the core concepts of this emerging paradigm, and attempts to present an account of current research efforts in the context of telecommunications. The goal is to provide the interested reader with a clear background of the opportunities and challenges this emerging paradigm brings about, and a descriptive look at some of the forerunners that are providing experimental technologies supporting this paradigm.

The term *mobile agent* contains two separate and distinct concepts: mobility and agency.[1] Mobile agents refer plainly to self-contained and identifiable computer programs that can move within the network and act on behalf of the user or another entity [1]. The idea of a self-controlled program execution near the data source has been proposed as the next wave to replace the client-server paradigm as a better, more efficient and flexible mode of communication (Fig. 1). Some authors classify mobile agent as a special case of an agent, as in [2]; others separate the agency from mobility, as in [3]. Despite the difference in definition, most research examples of the mobile agent paradigm as reported in the literatures currently have two general goals: reduction of network traffic and asynchronous interaction. The goals of mobility are surprisingly close to those of the well-researched process migration concept, although at a different level of abstraction. These are reduction of network traffic, load balancing, fault resilience, asynchronous interaction, and data access locality. Some authors [4, 5] have suggested that agents can be used to implement network management by delegation [6, 7] and to deliver network services [8]. The mobile agent paradigm proposes to treat the network as multiple agent-"friendly" environments and the agents as programmatic entities that move from location to location, performing tasks for users. Agents can function independent of each other or cooperate to solve problems. The merit of the mobile agent as a viable technology is still a contentious issue among various researchers. However, even if this merit issue is resolved today, researchers have yet to come up with optimal solutions to the issues facing the design and implementation of mobile agent system architecture, which includes (but this list is not exhaustive):

- Agent transfer mechanisms
- Naming, addressing, and locating a mobile agent
- Control of the mobile agent
- Exporting mobile agent states
- Mobile agent data transfer
- Transparent communication
- Security
- Secrecy and privacy
- Coordination
- Communication language, ontology
- Stability, performance
- Scalability
- Portability
- Resource management and discovery
- Authority
- Legality
- Ethics

Some of these issues have roots in the fields of process migration and distributed operating systems; others come from the field of artificial intelligence (AI). The first 12 have technical connotations, while authority (who owns agents and agent resources), legality (who is ultimately responsible for an agent's action), and ethics (in what context should agents be used) are social issues regarding the use of an agent and thus will not be discussed in this article. It is also worth noting that not all of these issues are dealt with in current mobile agent systems. In some areas, such as security, secrecy, privacy, coordination, communication, and control, the current solutions are far from optimal or even adequate. In others, such as stability, scalability, and performance, research efforts have yet to begin.

At the time of this writing, there are 20 academic, private, and organizational projects with a mobile agent system as the central theme, involving well-known institutions such as the University of Ottawa, MIT, University of Stuttgart, UMBC, OGRI, and GMD FOKUS. Private firms large and small also have active mobile agent research projects; they include IBM, General Magic, ObjectSpace, Mitsubishi, Hitachi, British Telecom, and Crystaliz Inc. Also, various standardization efforts are underway [9], one of which is the addition of a mobile agent facility (MAF) to CORBA [10]. The momentous research interest from the academic sector shows that the mobile agent paradigm is conceptually sound, and the interest from private firms shows that the technology is rapidly passing the purely scientific research stage. In fact, several firms have deliverables in the product testing stage (IBM, General Magic, ObjectSpace, and Mitsubishi).

THE MOBILE AGENT IN TELECOMMUNICATIONS

The agent concept is already being used heavily as a part of the service architecture of next-generation networks such as the Telecommunications Information Networking Architecture (TINA) [11]. Moreover, the mobile agent concept is also being considered by the TINA Consortium (TINA-C) in a broadening effort to leverage these new technologies [8, 12]. A number of industry-sponsored projects that investi-

[1] The term agency *is used here to denote "having the characteristics of an agent." Maybe a more appropriate word will be found in the future. In the meantime, this is the meaning of the word agency as used in this article.*

gate the use of mobile agent technology to provide network services and network management include the Hitachi project [13] (sponsored by Hitachi Corp.) and the NEC project [13] (sponsored by NEC). The MAGNA project (GMD FOKUS) aims directly at investigating mobile agent services to be proposed for consideration as part of TINA [13]. With the network playing a pivotal role in present and future information systems, it is easy to understand the growing interest in using mobile agents as part of the solution to implement more flexible and decentralized network architecture. In this section, the potential applications of mobile agents in network services and network management available in the general literature are presented and discussed. The aim is to provide starting points for further research and investigation interests.

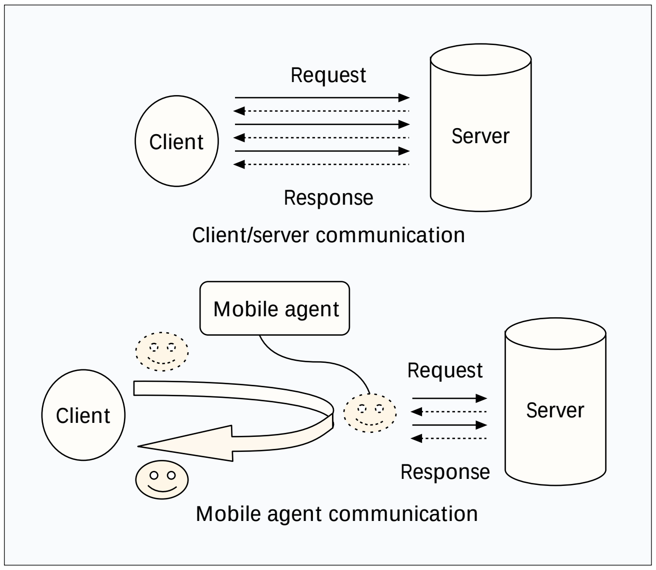

Figure 1. *A mobile agent can optimize network bandwidth usage.*

Mobile Agent in Network Services

The current network environment is based on international standards such as the telecommunication management network (TMN) and intelligent network (IN). IN and TMN rely on the traditional client/server paradigm to provide services via centralized nodes known as service control points (SCP). During execution of a service, the distributed exchanges known as service switching points (SSP) will ask the SCP for control services so that the SSP can carry out the actual processing. SCPs and SSPs communicate via a remote procedure call (RPC)-based protocol (the IN Application Protocol, INAP). To install IN services, specific service management systems (SMS) will download the necessary IN service components into the IN network elements.

A Mobile Agent to Enhance IN Services — A centralized SCP and the necessary usage of INAP represent potential bottlenecks in the case of an increase in the number of IN services. Therefore, the realization of services will have to be distributed as close to the customer premises as possible. The incorporation of the mobile agent concept within the IN for its enhancement has been suggested for the medium term [8]. Mobile agents used in IN would be responsible for the dynamic downloading of customized *service scripts* (i.e., dynamic migration of service intelligence from the SCP to SSP), ultimately allowing the provision of *service intelligence on demand* [4, 8] (Fig. 2).

Magedanz *et al.* [4] identified two general approaches for agent-based service architecture: *smart network* and *smart message*. In the smart network approach, agents are static entities in the network, able to perform tasks autonomously and asynchronously. These agents can also communicate with other agents and be dynamically configured. The issue raised by this kind of architecture is the dynamic downloading and/or exchange of *control scripts*; thus, intelligence resides mostly at the network devices. The control scripts can be simple or complex and represent "lightweight" mobile agent(s). In the smart message approach, agents are mobile entities that travel between computers/systems to perform tasks. Agents are received and executed in an agent execution environment (AEE) (Fig. 2). With this approach, intelligence is partitioned in a balance between the AEE and the agent. The smart message agent can serve as an asynchronous message carrier for its owner (e.g., retrieve e-mail asynchronously and forward to the current location of the owner), or as a broker that requests and sets up all requirements for services (e.g., establishes a real-time connection for media delivery).

With these agent-based approaches, services can be provided instantly, customized, and distributed. However, the approaches aim to replace IN components with mobile agents and thus are not consistent with IN's goal of centralizing service control. If IN does move toward this approach, it would evolve into architecture such as TINA.

Mobile Agents in TINA

TINA is the current target architecture for future telecommunications and management services [11]. Considered an evolution from IN and TMN, TINA allows flexible and transparent distribution of computation objects that are supported by distributed processing environments (DPEs). The mobile agent concept is not yet part of TINA; however, the TINA-C is working to expand the specification to accommodate relatively new concepts such as intelligent and mobile agents as extensions to TINA's support for the concept of agents. TINA has identified the following *agent dimensions* [8]:

- Act on behalf of someone
- Persistent
- Adaptive
- Mobile
- Communicating
- Reasoning
- Environmentally aware
- Socially aware

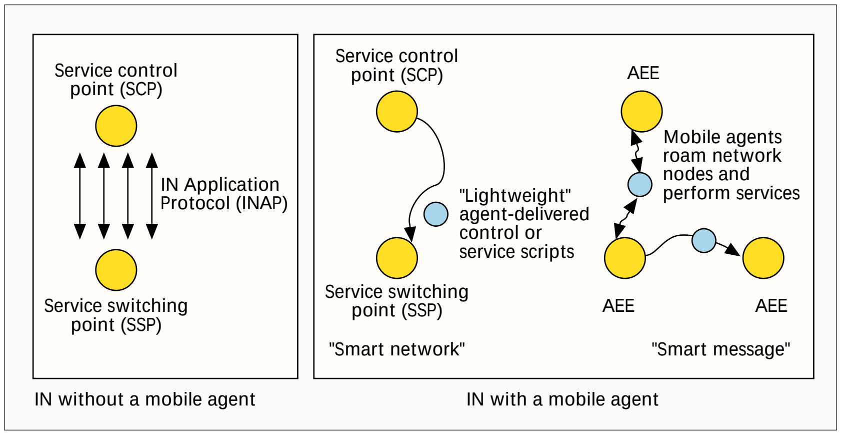
Service control point (SCP)
IN Application Protocol (INAP)
Service switching point (SSP)
IN without a mobile agent
Service control point (SCP)
"Lightweight" agent-delivered control or service scripts
Service switching point (SSP)
"Smart network"
AEE
Mobile agents roam network nodes and perform services
AEE
AEE
"Smart message"
IN with a mobile agent

Figure 2. *How a mobile agent can enhance IN services.*

- Planning
- Negotiating

To support such agent capabilities, considerable work has to be done to extend the DPE to support the AEE in one form or another. The MAGNA project aims to cover some groundwork in this area, through extending and refining GMD FOKUS's mobile agent architecture.

Mobile Agents in Network Management

Proposals and serious efforts have been initiated to use mobile agents to transform current client/server-based network management practices into a distributed and decentralized one [4, 5, 8]. This section elaborates some of the issues in network management and how mobile agents can help solve them.

Network Management Approaches

The most popular approach to manage networks comes from the Internet Engineering Task Force (IETF) and is based on the Simple Network Management Protocol (SNMP). Closely related in structure is the approach based on the Common Management Information Protocol (CMIP) proposed by the International Organization for Standards (ISO) for application within open systems interconnection (OSI) networks.

Both approaches assume the presence of *management stations* (MSs) that interact with *management agents* running on network nodes. The agents in these protocols are computational entities responsible for collecting and storing management information local to the node and responding to requests for this information from the MS via a *management protocol* that specifies the packet format for a set of basic operations. The MS interacts with the agents using client/server architecture. Yemini anticipated that this centralization in network management seriously limits its scalability, leading to poor performance or, worse, an inability to cope with the dimension of the network [14]. Recognizing these limitations, the IETF and ISO have taken steps to decentralize and relieve the bottleneck around the MS. These efforts include complex notification agents (ISO), *proxy agent* (SNMP v2, IETF), and *remote monitoring* (RMON, IETF) (Fig. 3a, b).

A clean design for decentralization is the management by delegation (MBD) approach proposed by Yemini [6] and Meyer [7]. In MBD, the management architecture still includes a management protocol and agents; however, *elastic process runtime support* is assumed to be present at each device. In the traditional way (SNMP, CMIP) the MS does all the computation and sending of the results to the devices via client/server messages. On the other hand, the MS in MBD packs a task (code and data) to agents and sends it to be executed at the devices, thus *delegating* to them the actual execution of the tasks. The executions would be asynchronous, freeing the MS to perform other tasks and thus enabling a higher degree of parallelism in the management architecture. In this way, a large portion of the functionality of the MS would be delegated to the devices (Fig. 3c). As an additional feature, codes are not statically bound to any devices; therefore, the MS can customize and dynamically enhance the services provided by the agents on any device [6].

Mobile-Agent-Based Network Management

The research field of mobile agents in network management is still young. Although projects such as Hitachi, MAGNA, and NEC have posted homepages, little technical information is available. From the general information, all projects have a similar scope of using in-house mobile agent architecture to provide telecommunications services and management. Until now, all projects use the term *agent* to denote a mobile executing unit (EU) [11] or mobile agent. Baldi *et al.* introduce yet another term, *mobile code paradigm*, and proposes that decentralization of network management services can be implemented using one or a combination of three design paradigms: code on demand (COD), remote evaluation (REV), and mobile agent (Fig. 4) [5].

COD is similar in concept to the use of a mobile agent. COD is proposed by Baldi *et al.* to allow dynamic configuration and functionality of network devices (*dynamic, active patching*) [5]. The ISO approach (CMIP) is amenable to this kind of application. However, the authors noted that the management agent in the IETF approach (SNMP) is too rigid to be considered for implementing COD.

In REV, a small code fragment is moved to the devices where it is allowed to invoke other codes to complete the service. The authors noted that this approach subsumes MBD, because MBD has fixed functionality (only the distribution is implemented), whereas REV also provides the benefit of dynamic configuration change obtained with COD. As an

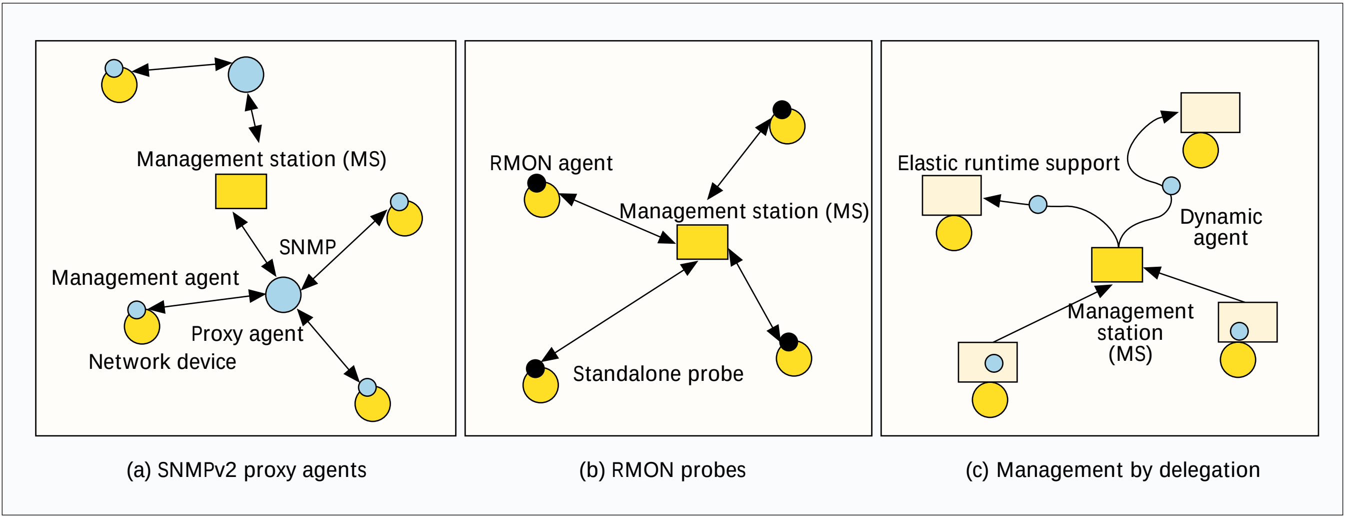

Figure 3. *Approaches to decentralization of network management.*

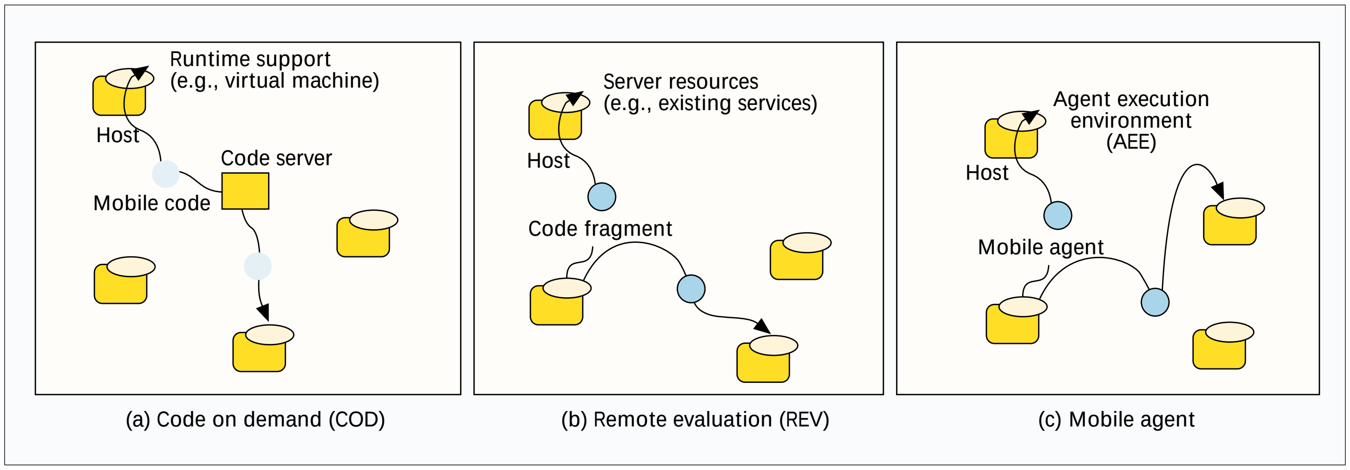

Figure 4. *Mobile code paradigms [5].*

example, the manager can pack a series of commands to be sent by an REV mechanism; these commands then invoke and execute built-in functionalities at the device. One such functionality is the search for routing table entry now being carried out at the MS.

In the mobile agent approach, the authors argue that sufficient intelligence of the agent allows it to travel from node to node to collect information and carry out device control tasks. The authors only cited two drawbacks: how to define an agent's intelligence, and complexity which may increase the agent's size.

To explore the effectiveness of the mobile code paradigm, the authors are currently implementing "near device" network management experiments, where agent-hosting environments are created at machines located near the network devices. Agents can then move among these machines and conduct client/server sessions with network devices using SNMP or CMIP. The implementation system used was Agent Tcl. The primary obstacle to direct device interaction was the lack of embedded support for an agent. Once this is a reality, Baldi *et al.* have plans to test all three design paradigms [5]. Their work is promising but incomplete as the authors concluded. The issue of complexity is only barely touched on; discussion on security and scalability was absent, perhaps delegated to researchers of Agent Tcl. However, the work of these authors represents a strong argument for the potential of mobile code (and mobile agents) in network management.

Research data on the application of mobile agent technology are still appearing, but much more is needed. As such, a survey of some typical current mobile agent systems is needed to generate more interest in experimenting with mobile agents in applications. This is the subject of the rest of this article.

A Survey of Mobile Agent Systems

There are several research activities that are related to mobile agent (MA) technology and many more centering on MA issues, which can be found in [9]. These cannot all be discussed due to the larger scope it would entail. In general, there are three targets for MA system design and implementation: using or creating a specialized language, as operating system (OS) services or extensions, or as application software. In the first approach, language features provide the requirements of MA systems. The second approach implements MA system requirements as OS extensions to take advantage of existing OS features. Lastly, the third approach builds MA systems as specialized application software that runs on top of an OS to provide MA functionalities. Currently, there is not enough information to assess with certainty the most suitable approach. For comparative purposes, this article chooses to limit the discussion to nine current projects:

- Aglet™ from IBM [16]
- Agent Tcl from Dartmouth College [17, 18]
- Agents for Remote Access (ARA) from the University of Kaiserslautern [19, 20]
- Concordia™ from Horizon Systems Laboratory, Mitsubishi Company [21, 22]
- Mole from the Institute for Parallel and Distributed Computer Systems (IPVR) [23]
- Odyssey™ from General Magic [24]
- TACOMA from Cornell University [25, 26]
- Voyager™ from ObjectSpace [27]
- Secure and High Performance Mobile Agent Infrastructure (SHIP-MAI) from the Multimedia and Mobile Agent Research Laboratory, University of Ottawa [28]

These systems are selected because they are representative of strategies currently employed in MA research. Language-based MA systems such as WAVE [29] and Obliq [30] are consciously excluded because these represent a different philosophical approach than MA support by software architecture.

Mobile Agent Models

An agent model is a conceptual view that MA system architects embed in their designs. In the following sections, descriptions of the MA models of Aglet, Agent Tcl, ARA, Concordia, Mole, Odyssey, TACOMA, Voyager, and SHIP-MAI will be presented.

Aglet

Aglet models the MA to closely follow the applet model of Java. It is a simple framework where the programmer overrides predefined methods to add desired functionality. An Aglet is defined as a mobile Java object that visits Aglet-enabled hosts in a computer network. Aglet runs in its own thread of execution after arriving at the host, so it is attributed as *autonomous*. It is also *reactive* because it responds to incoming *messages*. The complete Aglet object model includes additional abstractions such as *context*, *proxy*, *message*, *itinerary*, and *identifier*. These additional abstractions provide Aglet the environment in which it can carry out its tasks. Aglet uses a simple proxy object to relay messages and has a message class to encapsulate message exchange between agents. However, group-oriented communication is not available, and the choice of using a proxy to relay a message may not be a scalable solution in a high-frequency transport situation. Nevertheless, by modeling the MA as a Java object, the designers of Aglet leverage the existing Java infrastructure to take care of platform dependent issues and to use existing mobile code facility of Java.

AGENT TCL

The designers of Agent Tcl do not formally specify an MA model. Instead, MA is understood as a program that can be written in any language and that accesses features that support mobility via a common service package implemented as a server. This server provides MA-specific services such as state capture, transfer facility, and group communication, as well as more traditional services such as disk access, screen access, and CPU cycle. The philosophy was that all functionalities an agent ever wants are available in the server. Agent mobility then only concerns closure, which is the Tcl script (or scripts). There are no additional codes to load (i.e., no external references). In Agent Tcl, the state capture of an agent is handled automatically and transparently to the programmer. However, it was unclear what this state capture includes. Since Tcl is a script language, a frequent example given was that the executing script resumes after the instruction for mobility has been executed. There is also a plan to introduce process-migration-like behavior such that the states of the agent would continue to evolve as it moves from place to place. However, this trend could have adverse effects in areas such as the complexity of the transfer mechanism and cost, adverse effects that are still being dealt with in the more traditional process migration.

ARA

The Agent for Remote Action (ARA) system is similar in concept to Agent Tcl in that it has a core service layer supporting multiple languages through interpreters. ARA aims for seamless incorporation of "mobile programming" to the existing world of programming practice. In ARA, "the mobile agent is a program that is able to move at its own choice and without interfering with its execution, utilizing various established programming languages." The ARA agent moves between and stays at *places* where it uses services provided by the host or other agents. ARA agents are considered normal programs in all other aspects, working with file system, user interface, network interface, and other well-known computing facilities. In addition, ARA also formulates the concept of server agents, the services with which the MA would interact, as static compiled objects located at network nodes. ARA agents run within an interpreter, interfacing with a core language-independent set of services. The core services include resource management, mobility, and security, to name a few. The language-dependent features, such as how state capture is handled and correctness checking, are the responsibility of the interpreter, whereas providing access to the underlying OS services and other MA-specific services is the responsibility of the core. ARA agents are executed in parallel threads, while some of the internal ARA core functions can be executed as separate processes for performance reasons.

CONCORDIA

Concordia is another MA framework built on Java. In Concordia an agent is regarded as a collection of Java objects. A Concordia agent is modeled as a Java program that uses services provided by a collection of server components which would take care of mobility, persistence, security, communication, administration, and resources. These server components would communicate among themselves and can run in one or several Java virtual machines; the collection of these components forms the AEE at a given network node. Once arriving at a node, the Concordia agent accesses regular services available to all Java-based programs such as database access, file system, and graphics, as in Aglet. Like ARA, Concordia specifies a service bridge to provide access to legacy services.

A Concordia agent is considered to have internal states as well as external task states.[2] The internal states are values of the objects' variables, and the external task states are the states of an itinerary object that would be kept external to the agent's code. This itinerary object encapsulates the destination addresses of each Concordia agent and the method that each would have to execute when arriving there. The designers of Concordia claim that this approach allows greater flexibility by offering multiple points of entry to agent execution, as compared to always executing an "after-move" method as in Agent Tcl, Aglet, or ARA. This concept of an externally located itinerary is similarly supported in Odyssey via Odyssey's task object. However, the infrastructure for management of these itinerary objects was not clear from the publicly available literature on Concordia.

MOLE

In Mole, the agent is modeled as a cluster of Java objects, a closure without external references except with the host system. The agent is thus a transitive closure over all the objects to which the main agent object contains a reference. This island concept was chosen by the designers of Mole to allow simple transfer of agent without worrying about dangling references. Each Mole agent has a unique name provided by the agent system, which is used to identify the agent. Also, a Mole agent can only communicate with other agents via defined communication mechanisms, which offer the ability to use different agent programming languages to convert the information transparently when needed.

A Mole agent can only exist in a host environment call *location* that serves as the intermediate layer between the agent and the OS. Mole also supports the concept of *abstract location* to represent the collection of distributed physical machines. One machine can contain several locations, and locations may be moved among machines. Mole limits the abstract location to denote a configuration that would minimize cost due to communication. Thus, a collection of machines in a subnet is an acceptable abstract location, whereas a collection of machines that spans cities is not. As in ARA, Mole directly proposed the concept of a system agent (server agent in ARA), which has full access to the host facilities. It is through interacting with these system agents that a given Mole agent (mobile) achieves tasks. A Mole MA can only communicate with other agents (systems and MAs) and has no direct access to resources.

The uniqueness of the Mole agent model is its requirement for closure of objects, whereas other facilities such as static agent and communication are similar conceptually to other systems. What is unclear is how the Mole system enforces the closure requirement, and whether there are mechanisms to handle closure management automatically. The concept of closure is technically convenient, but without helping tools it can be error-prone and limiting.

[2] *The use of the term "task state" is confusing because these objects keep information about where to go and what to do at a site. This is analogous to where to go and what button to push when one gets there. It does not mean what the agent still has to do until its objective has been achieved, which is the meaning of task.*

ODYSSEY

The Odyssey project shares (or rather inherits) many features from a previous General Magic product: Telescript. However, the amount of open documentation on the Odyssey system is rather terse; therefore, its description will be limited. The Odyssey MA model also centers on a collection of Java objects, more similar in concept to Aglet than to Concordia or Mole. The top-level classes of the Odyssey system are *Agent*, *Worker*, and *Place*. Worker is a subclass of Agent and represents an example of what a developer can do with the Agent class. An Odyssey Place class is an abstraction of where an Odyssey agent exists and performs work. A special facility such as directory service is associated with Place.

Odyssey agents communicate using simple method calls, and Odyssey does not support high-level communication. However, Odyssey agents can form and destroy meeting places to exchange messages. There is also an undocumented feature regarding global communication to a "published" object, but this feature is not officially supported. The distinctive feature of Odyssey is its design to accommodate multiple transport mechanisms. Currently, Odyssey supports Java Remote Method Invocation (RMI), Microsoft Distributed Component Object Model (DCOM), and CORBA Internet Inter-ORB Protocol (IIOP). However, the current release of Odyssey does not add new or distinctive features from its Telescript predecessor, and the MA model is not yet stable.

TACOMA

The TACOMA (Tromsø and Cornell Moving Agents) project represents an early attempt to build an MA system. In TACOMA the agent is modeled as a migrating process that moves through the network to satisfy client requests. The TACOMA project focuses on OS support for MA and how MA can help solve problems traditionally addressed by operating systems.

TACOMA proposes abstractions such as *briefcase*, *folder*, and *file cabinet* as extensions to the OS to provide mechanisms for data transfer and *broker agent* for scheduling agents. *Brokers* are expected to communicate among themselves and with *service providers*; they also serve as matchmakers between agents and service providers. TACOMA also proposes the concept of a *rear guard agent* that serves as a checkpoint in case of failure.

In the first prototype Tcl was used as the implementation language and Horus, a lightweight group communication system, for interagent communication. The first TACOMA agent was a Tcl procedure (script) that used data available in briefcase and folder. The TACOMA MA model suffers from a number of weaknesses, including a weak execution model (no arbitrary entry point), OS specificity, and a weak security mechanism. At the current stage, the runtime system consists of interpreters for Tcl and C that provide access to TACOMA features (similar to fundamental features of ARA and Agent Tcl but much weaker). The programmer also has to take care of state.[3] Overall, it is unclear how far this approach can progress, since TACOMA is the only project that conducts investigation on implementing MA at the OS level.

VOYAGER

It would seem that the general acceptance of Java as the de facto Internet programming language gives rise to accelerated research in MA. Thus far four projects have taken advantage of Java language features to implement MA systems (Aglet, Concordia, Mole, and Odyssey), and more are coming [25, 28, 29]. However, none has yet represented a state of tight integration with Java like Voyager. Proclaimed as an agent-enhanced object request broker (ORB) in Java, Voyager offers several advanced mechanisms that could be used to implement MA systems. The Voyager agent model is also based on the concept of a collection of Java objects; it does have an *Agent* class that developers can subclass to implement Voyager-style MA. However, the product goes one step beyond to provide arbitrary remote object construction, a facility for moving objects (not just agents), and a host of other communication and infrastructure services that can be used to implement arbitrary MA systems.

The Voyager agent is designed to take advantage of the Voyager ORB features which make extensive use of Java's reflection mechanism. A Voyager agent can communicate by calling methods or using Voyager Space™ technology, a group event and message multicast facility that ObjectSpace claims to be more scalable than simple communication mechanisms in Aglet or Concordia. Like Aglet, Concordia, and Odyssey, the Agent class in Voyager also encapsulates a control model. However, Voyager supports a more extensive set of control mechanisms, such as more flexible instructions on how the agent should terminate itself than the fixed or explicit instructions on termination as in Aglet, Concordia, or Odyssey. As in Agent Tcl or ARA, interagent and high-level communication is reserved as application-level features indirectly supported with more primitive mechanisms provided.

The Voyager agent model is essentially the same in concept as in Mole, Aglet, Concordia, and Odyssey systems, which is a collection of Java objects. The designers of Voyager also leave the decision on the complexity of an agent to users, as do other systems that have been discussed. There are no metrics to suggest how the complexity of a mobile program would break the system; instead, the user is led to believe that arbitrary complexity of any degree can be achieved. This is a point worth noting since the user of a system should not have to experiment to find out what the system's limits are.

SHIP-MAI

Work on SHIP-MAI began from using existing MA technology for the transport and distribution of multimedia documents [31]. However, as research and experimentation progressed, it was realized that a number of enhancement requirements to MA systems in terms of high-bandwidth data transport, performance, and security began to emerge [32]. At completion, SHIP-MAI will include framework support for agents and agent task generation, agent task management, agent system management, and deployment. As in Aglet, Voyager, Mole, and Concordia, a SHIP-MAI agent is also modeled as a collection of related Java objects; hence, the SHIP-MAI system also uses Java for portability. However, SHIP-MAI includes two distinct and complementary parts in support system: an agent execution control server (AEC) and an agent execution server (AE) [32]. The AEC focuses on security aspects of establishing a security policy for incoming agents, while the AE focuses on enforcing this security policy and the actual resource allocation for agent execution. The AEC also acts as a command and control center for a collection of related AEs, where it manages the lifecycles of the agents running in this AE collection and provides external agent transport. The designers claim that this partition

[3] State is used here in a general sense and can be a combination of local and process states.

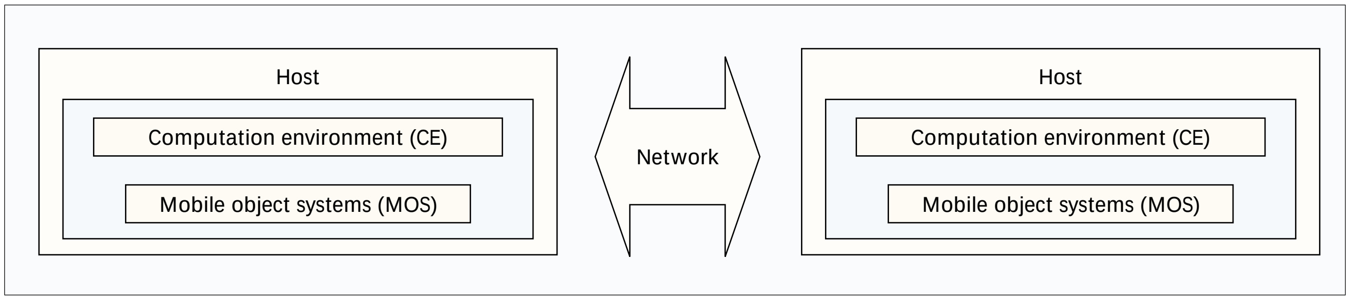
Host
Computation environment (CE)
Mobile object systems (MOS)
Network
Host
Computation environment (CE)
Mobile object systems (MOS)

Figure 5. *Mobile agent system components for security discussion [24].*

strategy allows more efficient use of resources and more flexibility in security configuration. Ongoing work focuses on efficient management of mobile agents to increase agent sharability, system scalability, and manageable fault tolerance. The work is motivated by the lack of an adequate solution for high load transfer of an agent's data; safeguarding an agent's data against possible loss due to system failure or a hostile host; application of MA in a high availability situation; and system features to deal with the issues in MA system management. Concurrent experiments to collect requirements needed to validate SHIP-MAI design are discussed in [33].

Mobile Agent System Requirements and Design Forces

In summary, MA systems consist of either Java class libraries (Aglet, Concordia, Voyager, Odyssey, Mole, and SHIP-MAI), scripting language systems with interpreter and runtime support (ARA, Agent Tcl), or OS services accessible via a scripting language (TACOMA). Aglet, Concordia, Odyssey, and Mole can be qualified as experimental application frameworks. Voyager is a current commercial product that advocates itself as agent-enhanced middleware. ARA and Agent Tcl are called "strongly mobile systems" by some authors [15] because there is tighter integration of mobility as a language feature, compared to external classes such as in systems based on Java. Conceptually, all approaches are similar. MA is considered a special application that requires two parts: the mobile part (MA) and a host part that reside on a computing device at a network node. There is also a service point or location concept that serves as a mediator between the MA and the services offered (called a *static agent* in some cases).

The environment in which an MA must function theoretically can be either uniform or heterogeneous; the latter is the norm. Although an MA system can be built with a single computing platform, such a system would have limited scope and usefulness. Current MA systems assume that the operating environment is heterogeneous, so the first consideration in the design of such systems is how to deal with platform heterogeneity. In the same category is the difference in computing power size and capacity, which seems to be ignored in current designs. The second issue that is absolutely crucial for an MA system is how to guarantee certain security levels so that the agent is protected from the host, the host is protected from the agent, agents are protected from each other, and hosts are protected from each other. Security also dictates that mechanisms exist to account for agents and to ensure that actions can be audited. There are adequate solutions to satisfy the heterogeneity constraint; however, only limited solutions are available to deal with the security constraints. If one assumes that adequate solutions exist for both, the remaining considerations in the design would be how to deal with resource allocation and discovery, how to identify and control agents, how to handle scalability, and so forth. The requirements current MA systems are trying to meet in supporting MA execution generally fall into nine general categories:

- Security
- Portability
- Mobility
- Communication
- Resource management
- Resource discovery
- Identification
- Control
- Data management

These categories are treated from the viewpoint of an MA, which may be different than from the viewpoint of a process. In the following subsections, each of these categories will be discussed as they are handled in some current systems.

Security

Research in agent security is very active due to its crucial importance to the general acceptance of MA systems. However, an extensive and complete discussion on security deserves its own article.[4] Therefore, this section aims instead to provide a comprehensive survey of current security models and solutions, and seeks to establish the initial set of security requirements in an MA system.

The issues and requirements of security in MA systems have been reviewed and discussed [34, 35]. A mobile object system including MA systems consists of four components according to [35]:

- A host—a computer and an OS
- The computation environment (CE) — the runtime system
- Mobile object systems — the computations concurrently running on the CE
- The network or communication subsystem that interconnects CEs located on different hosts (Fig. 5)

An agent is the third item in an MA system, while the AEE is the second. The security problems include how to protect the AEE from malicious agents, how to protect agents from a malicious AEE, how to protect one agent from another, how to protect an AEE from another AEE, how to protect the communication between AEEs, and how to protect the host from the AEE. The last problem is the traditional problem of protecting an OS from misbehaving programs; thus, it is not a new problem and is less important in the context of MA security. Here one is mainly concerned with the first four problems.

Farmer *et al.* identify three basic security principles that an MA system must realize [34]:

- For the most natural applications of MA, the participants cannot be assumed to trust one another.
- Any agent-critical decisions should be made on neutral (trusted) hosts.
- Unchanging components of the state should be sealed cryptographically.

[4] *See the article by Greenberg* et al. *in this issue.*

The authors go on to propose partitioning the network into one or more domains with a protected computer running an interpreter (i.e., an AEE) that is trusted by all agents in that domain. These special interpreters trust each other to various degrees depending on the relationships between the domains. All other interpreters are considered hostile by default and cannot be trusted. Agents in this system require special privileges to collect audit data and respond to attacks, but at the same time must be controlled so that they will not exceed their authority. In order to be effective, the system that dispenses and controls the privilege must be well protected. The security architecture proposed by [34] suffers from problems of securing the trusted interpreter from a rogue or masquerading agent, or a trusted agent that has been tricked into performing malicious tasks. Furthermore, since this architecture proposes a trusted network of hosts, additional problems of maintaining the overall trustworthiness of all participating hosts and ensuring that all hosts implement uniform security measures can be a formidable task in building an infrastructure.

Traditional security mechanisms rely on cryptographic techniques to implement authentication, authorization, and access control. These tools found use in traditional static distributed systems. They are also used as part of the solutions for protecting the AEE from hostile agents or another AEE, but there are no satisfactory solutions to prevent a hostile AEE from doing damage. The hostile AEE can attack by not running the agent code correctly, refusing to transfer the agent, tampering with agent code and data, or listening into interagent communication [34]. The problem of effectively preventing a hostile site from tampering with an agent is a nearly impossible task since the hosting site needs access to all the internal code of an agent in order to execute it [34, 35]. The current direction in dealing with a hostile AEE centers on collecting irrepudiated data for execution tracing, and providing warnings about possible attacks. The solutions proposed to protect an agent include cryptographic tracing of execution, duplication of computation and cross-examination, code obfuscation, and secure domains using a co-processor [35]. Unfortunately, these solutions are either not practical in real situations, due largely to their computing cost, or have limited scope. As for protecting agents from each other, solutions can be reasonably carried out by the design of the AEE, usually by isolating the execution of agents or providing a facility for agents to authenticate each other (Aglet, Mole, ARA, Odyssey), providing a secure object space, or using cryptographic techniques to protect objects from tampering. Protecting the host from malicious agents currently has the most solutions. These include various schemes for access control, various authorizations and authentications using digital signatures and other cryptographic techniques, and proof-carrying code [34, 35]. The most promising areas are achieving flexible access control and a recursive protection domain [35]. With these results, flexible protection solutions can be designed according to the level of trust.

In most systems the AEE and host are assumed to be trustworthy and would carry out operations such as helping the agent to protect the privacy of its results. In most systems implemented in Java, the designers choose to rely on the Java security model and/or provide customized extensions to this model. Aglet has its own implementation of the security manager and employs the sandbox Java applet model. Mole and Odyssey do not yet have any security model other than the basic Java facility. Concordia took steps to provide its own security manager and implements a secure transport mechanism with the secure socket layer protocol (v. 3). Voyager also has its own security manager plus a customizable sockets interface that developers can use to implement arbitrary socket-level security. The problem with these Java implementations is that any weaknesses in Java also permeate to them, and the Java model was proven in some cases to easily be broken [35]. Unlike Java, which is scrutinized openly for security compromises and keeps being improved, the security mechanisms in ARA and Agent Tcl are not yet ready and could not be widely tested; hence, they could be weaker than that of Java. Security is an ongoing research issue and should be regarded as an ongoing feature. What can perhaps be done now is to prepare the MA architecture such that new and better security mechanisms can easily be incorporated as they became available.

PORTABILITY

Platform heterogeneity is currently the norm; therefore, all current MA systems must deal with porting agent code and data to work on multiple platforms. Before Java, systems such as Agent Tcl and ARA depended on an OS-specific core and language-specific interpreters for agent code and data. The problems of these approaches are performance and scalability. After Java, the Java virtual machine (JVM) represents a better compromise because platform-neutral byte code can be compiled just in time to adequately alleviate the performance problems.[5] It is also a matter of scale to port the supporting structure to the various JVMs. Therefore, it is not surprising that many MA systems choose Java as an implementation language.

The Java approach does have its advantages and disadvantages. The advantages include widely accepted platform neutrality, ease of network programming constructs, and a constantly evolving security model. The disadvantages include the ongoing maturation of the language and its lack of mature development tools, plus a common weak link if a flaw in the language is found. However, the last weakness could be said about the core of either ARA or Agent Tcl systems, or any agent system in general.

MOBILITY

Agent mobility mechanisms include remote invocation (also known as remote execution), cloning, programming language support, middleware, and COD [11]. Most systems discussed in this article use application protocols on top of TCP for transport of agent codes and states. Systems based on Java make use of the programming language features extensively, such as support of a form of RPC (remote method invocation, RMI), object serialization, and reflection. Often the agent states and codes are transformed into an intermediate format to be transported and restarted at the other end. The mechanisms used are some variations of remote execution (all systems use it, since agents from all systems are restarted), programming language support (all systems that are based on Java[6]), and middleware.

Aglet has its own Agent Transfer Protocol (ATP), an application-level protocol that communicates via a TCP socket. Concordia also uses TCP sockets and Java object

[5] *A note of caution: this improvement is still mostly theoretical. The effort to improve performance always involves risks of reducing the portability of code. Code that is optimized for Sun's Java environment may not even work on another Java environment.*

[6] *These systems have to add features to Java to support migration via RMI or middleware (IIOP, DCOM), since Java does not have built-in features for code and state migration and reception.*

serialization for transport mechanisms. Odyssey is the only system that isolates the transport layer from the rest of the system and can use RMI, Internet Inter-ORB Protocol (IIOP), or the distributed component object model (DCOM). Voyager uses Java object serialization and reflection extensively in its transport mechanism. Mole initiates a replacement for the Java code loader model by proposing a code server and is expected to follow the same path as other systems based on Java. SHIP-MAI uses Java object serialization and manages transport at three levels: AEC–AEC, AEC–AE, and AE–AE. Both ARA and Agent Tcl hide the transport details from the developer in the form of a single move or go command and may use a number of protocols to transfer agent code and data. The core systems of ARA and Agent Tcl are supposed to handle decoding back to a language-specific interpreter format. Agent Tcl currently uses a TCP socket, while ARA foresees Simple Mail Management Protocol (SMMP), Hypertext Transfer Protocol (HTTP), and File Transfer Protocol (FTP) as possible routes. Note that Aglet, Concordia, Odyssey, Voyager, and Mole also support a transparent agent code and state transfer via a single command or method.

In the systems surveyed, execution of the agent is stopped and all local-resource-dependent activities have to be completed before it can be moved. In Agent Tcl and ARA, the system guarantees that execution resumes at the next statement after the move statement with the local variable restored. In Java-based systems the entry point (or points) in the form of method calls are executed at the next destination. All Java-based systems except Concordia have a single entry point after transfer of an agent occurs. In Concordia, an arbitrary entry point can be called at the destination.

COMMUNICATION

A communication model is needed so that agents can communicate with each other and for the system to control agents, and this has been discussed in the context of an agent model for each system in previous sections. In this section the strengths and weaknesses of each approach will be presented.

Systems based on Java mostly support event, message, and/or RPC-based communication, while ARA supports only client/server-style message exchange at a predefined service point. Agent Tcl plans to support event-based communication [21], but the description does not clearly define what this event means. Interagent communication in any form is not discussed in available literature on ARA. Agent Tcl has recently added support for RPC-style communication between agents and a message-passing model via byte streams [37].

Among the systems discussed so far, ARA and Agent Tcl have the most traditional communication framework. Both systems rely on a rigidly defined interface of possible procedure calls to establish communication. ARA still does not allow interagent communication, but Agent Tcl has introduced an Agent Interface Definition Language (AIDL) to allow agents in different languages to communicate [33]. The AIDL approach is similar to CORBA IDL and is necessary given the multiple languages that Agent Tcl supports. This style has the advantage of interfacing matching communication that facilitates client/server bindings. However, since it is based on RPC it also suffers from not being able to handle RPCs automatically, so interfaces have to be set up before proper connection is established. It also requires extra processing steps and extra objects. Therefore, agent communication in this style is not very straightforward. However, this approach may have benefits in establishing an exchange interface to facilitate a higher-level exchange [37].

In stark contrast to ARA and Agent Tcl, systems based on Java make use of the homogeneous language environment to bypass the interface incompatibility issue. Most systems implement a distributed event communication and/or a message-passing mechanism using normal Java objects. Voyager goes one step further to make use of the Java reflection mechanism to create agents remotely without resorting to an RPC-based system. It also introduces a multicast group event distribution called VoyagerSpace that it claims to be scalable to enable high-traffic agent communication. With VoyagerSpace, distributed events and published/subscribed styles of communication are possible. VoyagerSpace is also nestable and allows arbitrary disconnection. This is in contrast to AgentGroup from Concordia, where a group can be formed, but it is not possible to join a group arbitrarily. Of course, the disadvantage of the above approaches is language dependency. It would be advantageous to have a similar architecture available in language- and application-neutral form. The various CORBA services seem to fulfill this need.

RESOURCE MANAGEMENT

Agents are executing programs that may require access to low-level system resources such as CPU cycle, disk, memory, graphic subsystem and network. They may require higher-level access such as persistence service (i.e., database), thread, and services from static entities (i.e., back-end directory server, SQL server). Equitable distribution of this access to resources among requesting agents is dealt with by resource management.

Resource management is not clearly discussed in most systems mentioned in this article. Concordia is the only exception, explicitly providing a queue manager abstraction, but limiting its services to managing access to transport service. Agent Tcl and ARA seemed to vaguely specify that resource management would be carried out by the core layer; however, both systems did not specify where or how it could be carried out. Aglet, Odyssey, and Voyager, as other Java-based systems, left the low-level resource management to the JVM (by using the Java thread facility) and provided no other facility.

RESOURCE DISCOVERY

An agent is far more efficient if it can dynamically discover the resource it needs to accomplish its tasks than if it is hardwired to do work. Resource discovery covers an area complementary to resource management. When an agent arrives at a site, it should be able to discover the services offered at that site or things it could do. To be more efficient, this information needs to be available even before the agent decides to go to a site. Service discovery and trading is the fundamental premise of MA in [38]. Some authors argue that this is the application level of an MA system [17, 19]. As in resource management, resource discovery is virtually absent in current systems. Although the concept of static entities servicing an agent's requests is there (ARA, Mole), such systems assume that the agent is aware of these services before coming to a site. It would be better if schemes were designed as part of the overall architecture to allow both prior and just-in-time discovery of services. Furthermore, more research efforts are needed, particularly those aimed at determining the optimal approach that permits flexible applications to be built that offer both high performance and scalability.

IDENTIFICATION

Agents must be identified uniquely in the environment in which they operate. Proper identification allows control, communication, cooperation, and coordination of agents to

Mobile agent system	Security	Portability	Mobility	Communication	Resource management
Aglet	Limited, sandbox model	Java	Aglet Transfer Protocol	Event, message object	Java
Agent Tcl	Limited, sandbox model	Support multiple language interpreters	Multiple protocol	RPC	Yes
ARA	Limited, sandbox model	Support multiple language interpreters	Multiple protocol	RPC	Yes
Concordia	Limited, sandbox model and secure channel	Java	Socket and Java serialization	Event, group	Yes, via the queue server
Mole	Basic Java	Java	Enhanced Java model with code server	Event	Java
Odyssey	Basic Java	Java	Java RMI, CORBA IIOP, DCOM	Event	Java
TACOMA	Limited, uses firewall agent	None	TCP	Folder object	Operating system
SHIP-MAI	Sandbox model, secure channel, policy, access control	Java	Java object serialization	Event, group, room object, Java syntax for method call	Planned
Voyager	Limited, sandbox model, secure channel	Java	Java object serialization, reflection	Distributed event (VoyagerSpace), Java syntax for method call	Java

Table 1. *A summary of moble agent system features.*

take place. All schemes used in the current systems are variations of generating a unique number sequence designating the agent created. A globally unique identifier is also used in identifying Microsoft COM components and CORBA objects. The difference in an MA system is that it must allow easy access for human (programmer) use, since remembering a unique sequence of numbers is not straightforward to human. Thus, the purpose of an identification scheme is to generate unique identifiers and establish some infrastructure to allow convenient and accurate access to the agents that carry them.

Given the importance of uniquely identified agents, most Java-based systems have at least a facility to generate unique identifiers. Surprisingly, this issue was not discussed at all for ARA or Agent Tcl. To facilitate access to the agent using these identifiers, Mole suggests using DNS to associate name to number, and Voyager has an alias facility that a programmer can use to refer to an agent. Voyager also supports a federated naming service that facilitates linking of directory services to form a large logical directory (DNS style) which Voyager claims offers a superior way to locate an agent. In contrast, Aglet and Odyssey use a simple table lookup to associate name string to URL, while Concordia uses a directory manager to manage the naming service for its agents.

It is clear that the approach of Aglet and Odyssey is far from sufficient, and ARA and Agent Tcl do not discuss solutions at all. Moreover, it is not clear how the Voyager or Concordia approach fares in real situations where directories are constantly updated, a common scenario with agents moving about. SHIP-MAI currently requires the agent to report to the AEC at the local level and then uses a hierarchical arrangement of AECs at the global level. Some groups, such as Mole and the Open Software Foundation (OSF) Mobile Agent (MOA) project [36], choose to take advantage of the current DNS infrastructure; however, it is unclear whether such an approach is adequate.

CONTROL

The basic goal of control is simple: to provide ways in which the agent may be created, started, stopped, duplicated, or instructed to self-terminate. Control also covers subissues such as how the activities during the MA lifecycle can be coordinated. The common approach has been providing these abilities in the root MA class and then allows the programmer to tailor the MA responses to control-oriented events or messages (e.g., Aglet, Odyssey, Mole, Concordia, SHIP-MAI, and Voyager). The ARA and Agent Tcl control models were not clearly discussed, although the available information suggested that an arbitrary method or procedure could be relayed and invoked on an MA. The ability to respond to events has been proposed for the future version of Agent Tcl [33], but the solution may be a customized one instead of a de facto standard as in Java. On the other hand, Voyager allows the most flexibility and gives a lot of freedom in terms of control. Any methods can be called remotely, and it also has an event multicast facility to facilitate group control.

The control model is far from optimal in current systems, particularly when dealing with a group of many MAs. Most current solutions seem to work well with a small number of isolated MAs; however, there is little convincing proof that these strategies would work when dealing with a large number of MAs. The choice of reuse by inheritance is also somewhat limiting. A better choice would be reuse by composition, which gives the programmer flexibility in defining how the MA should be controlled. What is also missing is a control framework of how different low-level facilities such as VoyagerSpace and Aglet Message should be used to achieve scalable control of the MA.

DATA MANAGEMENT

An MA can carry with it data it needs to do work. It also needs to store itself in a persistent form for fault tolerance or other purposes. These services are provided by the AEE and

Mobile agent	Resource discovery system	Identification	Control	Data management	Case study in telecommunication
Aglet	None, user-implemented	Yes, via globally unique number sequence	Yes‹	None, user-implemented	None so far
Agent Tcl	Limited	N/A	N/A	Yes, in core	Yes
ARA	Limited	N/A	N/A	Yes, in core	None so far
Concordia	None, user-implemented	N/A	Yes	Yes, but limited	None so far
Mole	None, user-implemented	DNS	Yes	N/A	None so far
Odyssey	None, user-implemented	N/A	Yes	None, user-implemented	None so far
TACOMA	None, user-implemented	N/A	Yes	None, user-implemented	None so far
SHIP-MAI	Planned	Yes, use globally unique number sequence	Yes	Yes	Yes, in mobility management and information delivery
Voyager	None, user-implemented	Yes, use globally unique number sequence, alias, federated naming directory service	Yes	Persistent interface	Yes, but details not publicly available

Table 1(continued). *Summary of mobile agent systems features.*

are anticipated by all nine systems. However, only Voyager and Concordia have implemented persistent mechanisms, whereas other systems do not discuss how data management will be carried out at all. Furthermore, only Voyager provides ways in which persistence can be customized, although these still leave a lot to be desired. ARA talks about *checkpoint*, a snapshot of the MA that would be stored permanently, but it does not specify how the interface would occur or how it would support customary solutions. SHIP-MAI provides a repository for an agent to store data that is centrally managed and a distributed agent cache mechanism at the AE server; however, the details are not yet available.

Data management is important to allow an MA system to scale and also affects performance, but not all the systems discussed have satisfactory solutions. Some researchers take a different viewpoint and apply results in persistent language or programming systems as a solution, along with how to implement MA systems with these tools [39].

CONCLUSION

In the traditional distributed application environment (e.g., client/server architecture), specialized programs are designed to accommodate as many clients as possible. The client processes usually run on remote machines and communicate with server processes to perform work. This approach can generate a high level of network traffic and, depending on the network design, can be susceptible to congestion delay. The mobile agent paradigm proposes bringing the requesting client closer to the source and hence reducing the necessary traffic.

This article has presented an overview of the emerging mobile agent technology in the context of telecommunications. It has elaborated on some fundamental issues encountered in mobile agent system design and the solutions offered in nine representative systems. However, as Table 1 shows, the representative systems seem too generic in scope, with few specific experiments involving issues of interest to telecommunications. Demonstrations of how these systems propose to solve network management and services issues are not yet available. It is expected that these systems will have to be either specialized or used to build advanced systems. These steps are necessary to deal with a particular application domain such as network management and services. Nevertheless, the trend toward decentralization of network design to cope with growth in demand for services and the pressure for more efficient network management seems to be well suited to the emergence of this technology. MA is considered by many to be lightweight process migration that does not depend on operating system semantics, thus, it is less complicated to implement than process migration. However, the other aspect of mobile agents, agency, is relatively less well developed. Nevertheless, this limitation is only a temporary barrier, since objects can be mobile and perform useful work now without possessing all the required characteristics that define agency.[7]

ACKNOWLEDGMENTS

This work was supported partly by the Telecommunication Research Institute of Ontario and partly by the Natural Sciences and Engineering Reseach Council of Canada. We would like to also express our thanks to the anonymous reviewers for their insightful and helpful comments.

REFERENCES

[1] K. Rothermel and R. Popescu-Zeletin, Eds., *Mobile Agents*, Lecture Notes in Comp. Sci. Series, vol. 1219, Springer, 1997.

[7] *On the same note, there is a growing consensus among researchers in mobile agent systems such as those discussed in this article to replace the term “agent” by the term “object,” to eliminate the confusion with the artificial intelligence (AI) agent.*

[2] H. S. Nwana and N. Azarmi, Eds., *Software Agents and Soft Computing: Towards Enhancing Machine Intelligence*, Lecture Notes in AI Series, vol. 1198, Springer 1997.
[3] J. Vitek and C. Tschudin, Eds., *Mobile Object Systems: Towards the Programmable Internet*, Lecture Notes in Comp. Sci. Series, vol. 1222, Springer, 1997.
[4] T. Magedanz, K. Rothermel, and S. Krause, "Intelligent Agents: An Emerging Technology for Next Generation Telecommunications?" *Proc. INFOCOM'96*, San Francisco, CA, 1996.
[5] M. Baldi, S. Gai, and G. P. Picco, "Exploiting Code Mobility in Decentralized and Flexible Network Management", K. Rothermel and R. Popescu-Zeletin, Eds., *Mobile Agents*, Lecture Notes in Comp. Sci. Series, vol. 1219, pp. 13–26, Springer, 1997.
[6] Y. Yemini, "Network Management by Delegation," *Integrated Network Management II*, Krishnan and Zimmer, Eds., Elsevier, 1991.
[7] M. Meyer, "Decentralizing Control and Intelligence in Network Management," *Integrated Network Management IV*, Sethi *et al.*, Eds., Chapman and Hall, 1995.
[8] S. Krause and T. Magedanz, "Mobile Service Agents enabling Intelligence on Demand in Telecommunications", *Proc. IEEE GLOBCOM '96*, 1996.
[9] http://www.agent.org
[10] http://www.omg.org/library/schedule/Technology_Adoptions.htm#Mobile_Agents_Facility
[11] http://www.tinac.com
[12] M. Rizzo and I. A. Utting, "An Agent-Based Model for the Provision of Advanced Telecommunication Services," *Proc. 5th Telecommun. Info. Networking Architecture (TINA) Wksp.*, Melbourne, Australia, pp. 205–18, Feb. 1995.
[13] http://www.fokus.gmd.de/ima/projects.html
[14] Y. Yemini, "The OSI Network Management Model," *IEEE Commun. Mag.*, May 1993, pp. 20–29.
[15] G. Cugola et al., "Analyzing Mobile Code Languages," J. Vitek and C. Tschudin, Eds., *Mobile Object Systems: Towards the Programmable Internet*, Lecture Notes in Comp. Sci. Series, 1222, Springer, 1997, pp. 93–110.
[16] http://www.trl.ibm.com/aglets
[17] R. S. Gray, "Agent Tcl: A transportable agent system," *Proc. CIKM Wksp. Intelligent Info. Agents*, J. Mayfield and T. Finnin, Eds., 1995.
[18] http://www.cs.dartmouth.edu/~agent
[19] H. Peine and T. Stolpmann, "The architecture of the ARA platform for mobile agents," K. Rothermel and R. Popescu-Zeletin, Eds., *Mobile Agents*, Lecture Notes in Comp. Sci. Series, 1219, Springer 1997, pp. 50–61.
[20] http://www.uni-kl.de/Ag-Nehmer/Projekte/Ara/index_e.html
[21] D. Wong *et al.*, "Concordia: An Infrastructure for Collaborating Mobile Agents," K. Rothermel and R. Popescu-Zeletin, Eds., *Mobile Agents*, Lecture Notes in Comp. Sci. Series, 1219, Springer, 1997, pp. 86–97.
[22] http://www.meitca.com/HSL/Projects/Concordia
[23] http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole.html
[24] http://www.generalmagic.com/technology/odyssey.html
[25] D. Johansen, R. van Renesse, and F. B. Schneider, "Operating system support for mobile agents," *Proc. 5th IEEE Wksp. Hot Topics in Op. Sys., IEEE Comp.*, New York, 1995.
[26] http://www.cs.uit.no/DOS/Tacoma/index.html
[27] http://www.objectspace.com/voyager
[28] http://deneb.genie.uottawa.ca
[29] P. Sapaty, "Mobile WAVE technology for distributed knowledge processing in open networks," *Proc. CIKM '95 Wksp. New Paradigms in Info. Visualization and Manipulation*, Baltimore, MD, Dec. 2, 1995.
[30] http://www.research.digital.com/SRC/personal/Luca_Cardelli/Obliq/Obliq.html
[31] B. Falchuk and A. Karmouch, "AgentSys, A Mobile Agent System for Digital Media Access and Interaction on the Internet," *Proc. IEEE GLOBECOM '97*, Phoenix, AZ, Nov. 1997.
[32] A. Karmouch and V. A. Pham, "Study of a Mobile Agent-based Architecture in a Networking Environment," Tech. rep. TR-03, Univ. Ottawa, Aug. 1997.
[33] A. Hooda, A. Karmouch, and S. Abu-Hakima, "Nomadic Support Using Agent-Level Communication," to appear, *Proc. 4th Symp. Internetworking*, Canada, July 1998.
[34] W. M. Farmer, J. D. Guttman, and V. Swarup, "Security for Mobile Agents: Issues and Requirements," *Proc. 19th Nat'l. Info. Sys. Security Conf.*, Baltimore, MD, Oct., 1996, pp. 591–97.
[35] J. Vitek et al., "Security and Communication in Mobile Objects Systems" J. Vitek and C. Tschudin, Eds., *Mobile Object Systems: Towards the Programmable Internet*, Lecture Notes in Comp. Sci. Series, vol. 1222, Springer, 1997.
[36] D. S. Milojicic, S. Guday, and R. Wheeler, "Old Wine in New Bottles: Applying OS Process Migration Technology to Mobile Agents," Open Group Res. Inst., 1997.
[37] D. Kotz *et al.*, "Agent Tcl: Targeting the Needs of Mobile Computers," *IEEE Internet Comp.*, vol. 4, no. 1, Aug. 1997, pp. 58–67.
[38] B. Schulze, "Contracting and Moving Agents in Distributed Applications Based on a Service-Oriented Architecture," K. Rothermel and R. Popescu-Zeletin, Eds., *Mobile Agents*, Lecture Notes in Comp. Sci. Series, vol. 1219, Springer 1997.
[39] M. da Silva and A. R. da Silva, "Insisting on Persistent Mobile Agent Systems," K. Rothermel and R. Popescu-Zeletin, Eds., *Mobile Agents*, Lecture Notes in Comp. Sci. Series, vol. 1219, Springer 1997.

ADDITIONAL READING

[1] http://www.opengroup.org/RI/java/moa/index.htm
[2] http://www.sce.carleton.ca/netmanage/perpetum.shtml

BIOGRAPHIES

VU ANH PHAM is a research associate with the Multimedia and Mobile Agent Research Laboratory, University of Ottawa, Ontario, Canada. His primary interest is in the security, scalability, and performance issues of mobile agent system architecture. He received his M.Sc. in system science from the University of Ottawa in 1998.

AHMED KARMOUCH (karmouch@site.uottawa.ca) is a professor of electrical and computer engineering and computer science at the School of Information Technology and Engineering, University of Ottawa, Ontario. He holds an Industrial Research Chair from the Ottawa-Carleton Research Institute and the Natural Sciences and Engineering Research Council. He is also director of the Ottawa Carleton Institute for Electrical and Computer Engineering. Prior to joining the University of Ottawa in 1988, Karmouch was senior manager at Bull S.A, Paris, France, were he was responsible for the Multimedia Information System group. Karmouch has made significant contributions in the areas of multimedia communications, multimedia databases, and mobile software agents. He is director of the Multimedia and Mobile Agent Research Laboratory. He leads several projects on multimedia and mobile agent research in collaboration with Telecommunications Research Institute of Ontario, Communication and Information Technology of Ontario, Canadian Institute for Telecommunications Research, Nortel, Bell Canada, Mitel, National Research Council Canada, Centre National de Recherche Scientique in France, and TeleLearning National Center of Excellence. He received M.S. and Ph.D. degrees in computer science from the University of Paul Sabatier, Toulouse, France, in 1976 and 1979, respectively. He is a member of several committees and editorial boards. His current research interests are in interactive multimedia information systems, home architecture and services, telelearning, and mobile software agents for telecommunications.