INFORMATICA, 1997, Vol. 8, No. 1, 119–138

NOTES ON SOFTWARE AGENTS AND THE MOBILITY ISSUE

Matin ADDIBPOUR

Royal Institute of Technology, Department of Teleinformatics
Electrum 204, 164 40 Kista, Sweden
E-mail: mattin@it.kth.se

Abstract. An overview and comparison of mobile agent systems are presented. The rapidly evolving area of software agents is briefly overviewed. The notion of mobility is analyzed in the context of mobile code languages, and its relation to distributed computing (e.g., client-server model), as well as its possible application areas are studied. Finally the need for a combination of mobility with other features is discussed.

Key words: agent, software agents, mobile agents, distributed computing, client-server model, AI, DAI, KQML.

1. Introduction. Different notions of software agents have been employed in subfields of the emerging agent research community, mainly focusing on separate and unrelated contexts. Mobility is one of the feasible attributes assigned to an agent. Mobility refers to an agent's ability to roam in the network, this notion has been exploited mainly in the distributed computing community. Although mobility is not a necessary attribute for realizing an agent in general, it offers many practical advantages for realizing an agent-based computing environment.

Distributed computing by means of mobile agents has been popularized in recent years. In this field agent-based computing has been considered an extension to remote execution of script programs giving special consideration to the security issue. The goal is to advance the classical client-server computing into mobile agent-based computing. In an agent-based framework applications (executable programs) are sent across the network instead of data. In order to extend this scenario one can consider mobile agents as autonomous agents roaming between agent servers or Agent Execution Environments (AEE) accomplishing their user's goals.

This paper is organized as follows. First a short overview of the rapidly evolving area of software agents is given in Section 2. In Section 3, the main common issues regarding the concepts used in the existing mobile agent systems are discussed. Finally a possible combination of mobility with other aspects of agents is briefly discussed in Section 4.

2. Software agents. What is an *agent*? How is it different from a "computer program", a "node" in a network, or from an "actor"? There have been several attempts to give a definition of "agent". The question of "what is an agent", seems to be as difficult as the question of "what is intelligence". Compare the four following definitions of an agent:

> *"An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors."*
>
> AI agents (Russell and Norvig, 1995).

> *"An entity is an agent iff it communicates correctly in an Agent Communication Language (ACL)."*
>
> Heterogeneous agents (Genesereth and Ketchpel, 1994).

> *"An agent is a computer program that employs AI techniques in order to provide assistance to a user dealing with a particular application."*
>
> Interface agents (Maes, 1994).

> *"An agent is a process that may migrate through computer network in order to satisfy request made by its client."*
>
> Mobile agents (Johansen *et al.*, 1995).

It seems that there is no cross-field consensus about what "agent as a metaphor" refers to. The meaning of the word "agent" is determined by the context in which it is used. In the following three papers, the issue of defining an agent is studied, they all try to solve the problem by using different classifications.

Wooldridge and Jennings (1995) divide an agent's notions into weak and strong ones. *Weak notion* of agency includes: *Autonomy*, a system is autonomous to the extend that its behavior is determined by its own experience. *Social ability*, interaction with other agents (and possibly humans) via some kind of Agent Communication Language (ACL). *Reactivity*, response in a timely fashion to the changes that occur in the environment. *Pro-activness*, exhibiting goal-directed behavior by taking the initiative. *Strong* notion refers to an agent in the context of AI. In addition to the weak notions, an agent is considered to

have *mentalistic notions*. By assigning a mental state to agents, they can be considered as *intentional systems*. An intentional system has two important kinds of attributes: *information attitudes*, which inform the agent about the world, e.g., knowledge and belief, and *pro-attitudes* which guide the agent's actions, e.g., desire, intention, obligation and commitment. An agent should have at least one attribute from each category, other notions are mobility, rationality, believability and so forth.

Frankling and Graesse (1996) give a *natural kinds taxonomy* for autonomous agents and the collection of agent's features is used as a farther classification. After discussing eleven different definitions of agency, they propose the following definition: *"An agent is a system situated within and part of an environment that senses and acts on it, over time, in pursuit of its own agenda and so as to affect what it senses in the future"*.

Nwana (1996) classifies agents based on a typology of them. He argues that no research community owns the term "agent" in the same way that for instance the term "fuzzy logic" is owned by logicians/AI researchers. The word "agent" has become an umbrella term covering a heterogeneous body of research and development.

These classifications and definitions express some common notions of an agent, but they fail to indicate an unified definition. The main difficulties with an unified definition are the following: 1) there are many meaningful choices, some of them are inherently contradictory; 2) a unification would restrict the use of the agent metaphor; 3) the metaphor will be more ambiguous in the future (Burkhard, 1995).

A set of software agents studied and developed in different fields is briefly overviewed in the following subsections (see Fig. 1). Dashed lines in the figure suggest that agent research in some fields are mutually stimulating. The classification given here is related to the field in which the agents are studied. This classification is neither strict nor complete. It is not strict in the meaning that there exist overlaps between the agent's functionality and concepts, and it is not complete because there are certainly other types of agents (Interface Agents, Information Agents and so on) which are not presented here. Mobile agents, which are the main concern of this paper, will be discussed in a separate section.

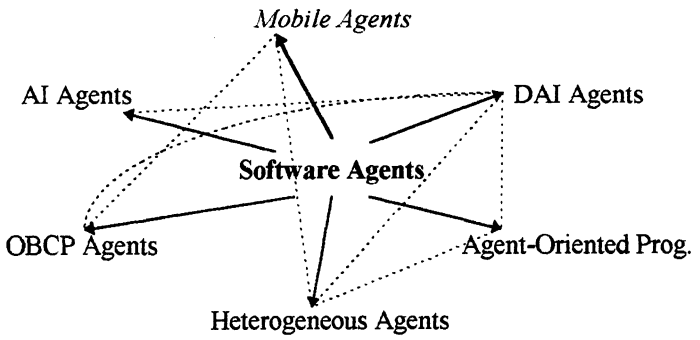

Fig. 1. Software agents discussed in this paper. Dushed lines indicate that some fields are mutualy stimulating the development of agents.

2.1. Agents in AI. The metaphor has its root in the early history of AI. Intelligent computer agents are both the original and the ultimate goal of AI research (Hayes-Roth, 1995). Surprisingly, the issue of agent synthesis has been relatively little considered until a decade ago. Since the construction of an intelligent computer agent is a hard task, the AI community started to attack the problem by a "divide and conquer" strategy. Different subfields of AI studied different parts of the agent, mostly without having any forecast about the final goal, i.e., an integration of the developed methods and concepts into an agent. The diverged development of the agent's components, caused a painful integration and is still a real challenge in AI. In a recent book on AI (Russell and Norvig, 1995), the authors attempt to place almost every research effort in AI in the context of constructing an AI agent. Fig. 2 shows a simplified view of an AI agent, it can be defined as anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors (Russell and Norvig, 1995). Since the environment can be even a software environment, this view is correct for a software agent situated in a software environment (e.g., Internet).

Architectures. Several architectures for constructing the cognition part of a single agent have been studied. Three types which fall in the frame of AI agents can be mentioned, namely:

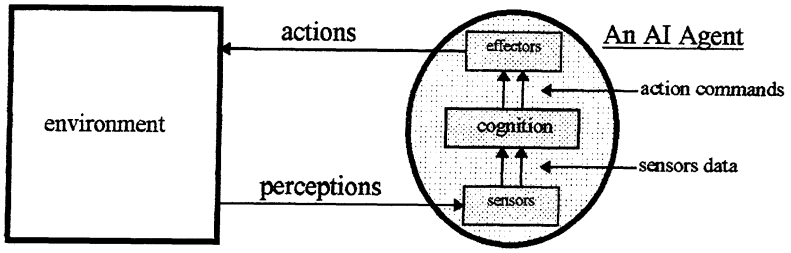

Fig. 2. A simplified view of an AI agent.

- *Deliberative agents*, which contain an explicitly represented, symbolic level of the world, where decisions are made via logical reasoning. The underlying assumption is that cognition functionality can be modularized. Deliberative agents can have the ability to reason about the new situations and other agents' behavior. The limitations are the computational complexity of symbol manipulation in general and slow response time.
- *Reactive agents* are based on a collection of simple behavior schemes (stimulus-response behavior) which react to changes in the environment. The agent should continuously refer to its sensors instead of to an internal world model. There is no symbolic model of the world represented internally. The *subsumption architecture* (Brooks, 1991) and *situated automata* (Rosenschein, 1989) fall into this group. The problems are the agent's hard-wired behavior and the lack of ability to learn, adapt and predict.
- *Hybrid agents* get benefit of both architectures, combining the high-level reasoning with low-level reactive capabilities. The reactive components serve the time critical actions, the deliberative part guides the behavior of the reactive part, for instance by changing the set of the situation rules of the reactive part.

These were architectures of an single agent. In the case of a set of agents, new consideration must be taken into account when building a multi-agent environment. Studying the behavior of a collection of interacting agents is the concern of Distributed Artificial Intelligence (DAI).

2.2. Agents in DAI. Distributed Artificial Intelligence (DAI) is a subfield of AI which focuses on studying systems where a collection of agents are interacting. DAI is divided into two parts, Distributed Problem Solving (DPS) and Multi-Agent System (MAS). DPS considers task allocation in view of a common goal, the problem is assignment of a global task to a set of co-operative agents. Each agent has a goal of maximizing a global utility function. MAS, on the other hand, refers to several autonomous agents, equipped with their own knowledge, local goals and abilities, sharing a common environment. Agents are concentrating on solving local tasks and maximizing the local utility function. Global situations may emerge, but agents may have competitive goals instead of shared ones. Moving from a single agent case to a society of agents, brings a set of new issues under consideration. DAI is therefore interested in co-ordination, communication, co-operation, negotiation and also in theories of intention and action (Bond and Gasser, 1988).

2.3. Agents in object-based concurrent programming. The notion of an *actor* as proposed in the concurrent actor model fits to the concept of an agent. An actor is "a computational entity which has a mail address and a behavior, and can communicate by message passing and carry its actions concurrently" (Hewitt, 1977). In this context an agent can be seen as an extension to an actor, which in its turn is a self-contained, interactive and concurrently-executing *object. Object-Based Concurrent Programming* (OBCP) is in many respects the ancestor of agent languages. OBCP is based on the notion of active and autonomous objects which are computing concurrently and which interact by exchange of messages. DAI as discussed above is concerned with representation, reasoning and co-ordination in collection of computational agents. There are good reasons for extending OBPC to provide a support to address DAI issues (Gasser and Briot, 1992).

2.4. Heterogeneous agents. Heterogeneous agents refer to agents in the context of agent-based software engineering (Genesereth and Ketchpel, 1994). The main motivation for these agents are the increasing demand for the programs working in isolation to become interoperable. The ability to effectively communicate and exchange knowledge between applications in a networked computing environment, can be realized by heterogeneous agents. An entity is an agent iff it communicates correctly in an *Agent Communication Language*

(ACL). The architecture of a heterogeneous agents system is studied by *Knowledge Sharing Effort* (KSE). The KSE three groups focus on how heterogeneous agents can be interoperable. Agents are heterogeneous if their implementation programming languages, knowledge representations and inference systems are not the same. To realize an agent, KSE defines:

- *Knowledge Query and Manipulation Language* (KQML) as a common Agent Communication Language (ACL). KQML defines a high level protocol for the interaction and communication between agents, based on *speech act theory.*
- *Ontolingua* as a set of tools, for maintenance of vocabularies, to ensure that no confusion can arise when agents with different vocabularies exchange knowledge.
- *Knowledge Interchange Format* (KIF) as a standard knowledge representation language for knowledge interchange between agents.

Each KQML message consists of a communication type and one more KIF expression. Legacy software can be connected to the network of the heterogeneous ACL speaking agents by an agentification of the original system, supporting interoperability between software by decoupling implementation from interface. The semantics of KQML has been recently specified with a farther extension of the language (Labrou, 1996). KQML is currently used in many other areas of agent research as a feasible ACL.

2.5. Agent-oriented programming. Agent-oriented programming (AOP) together with AGENT0 a first prototype of an AOP, are proposed by Shoham (1991). The differences between an object and an agent are the following: an object can have arbitrary attributes, but the attributes of an agent are a fixed set of attributes which constitute the agent's *mental state.* On the other hand the inter-agent communication language has a predefined pattern and is constrained to a set of KQML-like messages in contrast to inter-object communication which is less constrained. The agent percepts the incoming messages, updates its mental state and infers the actions (e.g., outgoing messages) to be taken. New AOP languages based on AGENT0 or the AOP paradigm have been developed.

3. Mobile agents. Mobility in the context of mobile agents means the ability to pick itself up, transfer itself to another location while maintaining the internal order and continue its execution. Generally speaking, a mobile agent refers to

a computer program which is able to migrate from a computer to a remote host and resume its execution.

Technically, a mobile agent can be seen as (Baumann *et.al.*, 1996): *"an object consisting of code, data and execution state that may go beyond protections domains"* or *"a component containing at least one thread of execution, which is able to autonomously migrate to a different site. A site is a component execution environment inside which inter-component communication is less expensive than communication among components residing on different sites"*. More practically, a mobile agent is *"a program:* 1) *that a person or organization vests with its authority;* 2) *that can run unattended for a long time (e.g., a week);* 3) *that can meet and interact with other agents;* 4) *and that can execute on different computer systems at different times of its life"* (White 1996).

Mobile agents are implemented as mobile programs. The underlying programming languages supporting mobility of programs are named *Mobile Code Languages.* In Subsection 3.1, mobile code languages are studied. In 3.2, mobile code paradigms are considered in the framework of distributed computing. Finally mobile agent systems are overviewed in 3.3.

3.1. Mobile code languages. There are several ways of providing code mobility. Mobile code languages can be divided into those supporting strong or weak mobility (Carzaniga *et al.*, 1996). *Strong* mobility, means that both the code and the execution state of a program are moving to a new host. Execution of a program is suspended, transmitted to the destination host and resumed there. *Weak* mobility, refers to the ability of a program to be bound dynamically to code coming from a different site, either by downloading from the network (as in Java (Sun, 1995)) or by receiving it from another remote program. Telescript (White, 1994) and Agent Tcl (Gray, 1995) belong to the strong group. Java supports weak code mobility by its class loader which enables downloading of a class from the network, and TACOMA (Johansen, 1995) allows sending both code and data for remote execution.

3.2. Mobile code paradigms. Code mobility can be used in order to achieve different ways of implementation of distributed applications (see Fig. 3). Today, the well-known and widely used client-server (CS) paradigm is used for realizing distributed applications. In the CS model there is no need for code mobility:

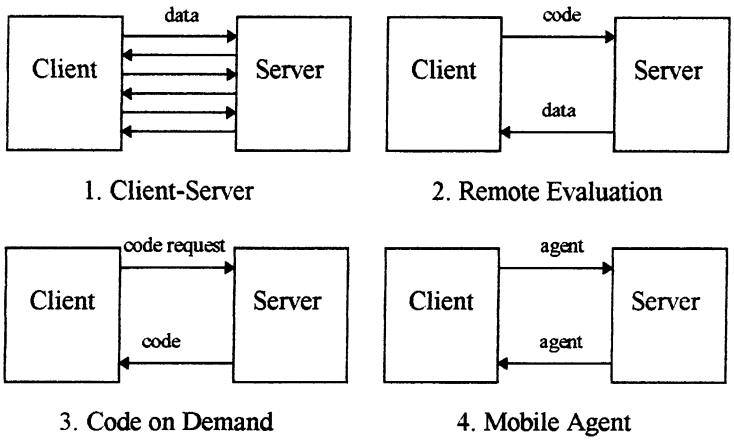

Fig. 3. Four approaches to distributed computing.

- *Client-server.* In the client-server model, a server exports a fixed and non customizable set of services, the client can use services by using a RPC-based communication. The code, required resources and processor are all on the server side.

In this context, three other paradigms using the mobile code approach can be mentioned (Carzaniga, 1996):

- *Remote evaluation.* Assume that the server exports an "execute-code" service which takes as its argument, a code to be run on the server. The client can now send a code and customize server resources. The client can use server resources and computational power in a more flexible way. The code is on the client side, resources and the computational power are on the server side. As an example rsh command in the UNIX world, allows a user to execute script code on a remote host.
- *Code on demand.* If the client has the resources and the computational power, but needs the necessarily code for customization of services, the code can be fetched from the server. In many upcoming Internet applications this paradigm is used. The Network Computers use basically this approach.
- *Mobile agent.* In this approach the code is acutely executed in the local host, but at some point the execution is halted and the executing

program (code plus state) will migrate to a remote host, which may have the resources needed for continuing execution. Note that in the mobile agent paradigm the computation is started before the code migration and is continued afterwards, versus in the two previous approaches, the code is first moved and then executed.

Advantages of these alternative paradigms in comparison with the CS model are:

- *Performance improvement,* to carry out a computation in the CS model, often several network messages will be transferred between the client and server. Sending the code will reduce the frequency of interaction and thus communication. Moving the code near to the data provides the computation to be performed locally and not remotely.
- *Flexibility,* the type of services which the client can obtain are not restricted to the server interface. Sending/receiving code to/from the server offers a service that is programmable with a computationally complete language.
- *Simpler interface on the server side,* since the server does not need to deal with complex interfaces to optimize for low bandwidth connections or many complex procedures for providing extensive functionality.
- *Simpler failure semantics,* local failure recovery provides a more consistent model, since the difficult scenarios where both the client and the server have to roll-back do not occur.
- *Simpler application programming,* once the mechanisms for mobility are solved, since the communication is reduced to a local site issue, the application programmer does not need to deal with complex networking syntax and failure semantics.
- *Semantic routing,* in the case of a mobile agent, it can get information from one server about another server where the mobile agent can accomplish its task.
- *A more intuitive model,* co-locating the service provider and the service requester is a real life metaphor.

Disadvantages of these models are:

- *Security problem on server/client side,* since the server/client has to offer an execution environment for an external code, there exists a risk for a malicious code.

- *Limitation of computational power,* there must exist mechanisms for restricting the incoming code from exhausting the computational resources on the server/client.
- *Network bandwidth,* sending the agent does not always mean less communication overhead, it depends on the size of the agent. Sending a large agent can acutely consume more bandwidth than a couple of data interactions between server and client.

These advantages and disadvantages show that alternative approaches are not generally a better approach than RPC-based CS interaction. It depends on the patterns of interaction between the client and the server and the size of the code. There are certainly cases and applications where the classical model is more adequate. This concludes that the mentioned paradigms should be seen as complementary (Magedanz, 1996).

3.3. Mobile agent systems. The discussed *mobile agent model* can be extended to what will, in continuation, be called mobile agent systems. Mobile agent systems refer to a more general environment for the execution of agents. In mobile agent systems agents are operating on *Agent Execution Environments* (AEE) through out the network. They are able to migrate toward the problem to cooperate with other agents. The mobile agent systems are not only an alternative to a more traditional approach (e.g., CS), but they also have practical value in supporting limited local resources, asynchronous computing, a more natural development environment (e.g., for realizing electronic marketplace), and a more flexible distributed computing architecture.

Today, there exist environments for realizing mobile agent systems, both commercial (e.g., Telescript (White, 1994) and Aglet (Lange and Chang, 1996)) and research prototypes (e.g., TACOMA (Johansen *et al.*, 1995) and MOLE (Strasser *et al.*, 1996)). Most of them are based on a set of common concepts. The concepts and design space for a *hypothetical mobile agent system,* i.e., a distributed system which enables and supports execution of mobile agents will be discussed here. This hypothetical mobile agent system is based on an analyze of the current approaches in realizing the mobile agent systems, by observing a set of concepts which are shared among these systems and the mobile agents running on them.

3.3.1. The mobile agent. The mobile agent can be build from procedural components or from classes and objects. Agents are preferably implemented

in interpreted languages. Although there is a performance penalty for this, it has many advantages in comparison to the machine languages. Interpretable code can more easily operate on heterogeneous platforms, they can contain references which exist in the destination and not in the current host of the agent (late binding) and security is easier to maintain, since language developers can explicitly exclude the access point to the crucial system resources. A mobile agent can consist of:

- the program code (program state),
- the content of the instance variables (data state),
- the stack (execution state).

A simple mobile agent can contain a program code in the form of a script, with data state presented as assignments in the program. For the sake of efficiency, parts of code can also be dynamically fetched on demand.

Mobility. What happens when an agent decides to move? Two scenarios can occur. Agent A can create another agent B for sending to the remote host, after performing the computation in the remote host, B will either return or send back the final results. A can suspend while B is away, or can continue execution at the local host. On the other hand, agent A can stop execution in a point, move to another host and continue execution from the point where it stopped, much alike process migration in Operating System (OS). Usually the language supports a command like *jump, go, move or migrate.* The semantics of these commands are different in different languages, but all cause the program to move to a remote host.

Autonomy. A mobile agent decides to migrate and move to another host by his own choice. In mobile agents autonomy means that the agent's migration is a self-desired migration, and this is not forced by an external entity (e.g., OS). The migrating processes, can not be seen as mobile agents since they are forced to migrate by OS, in respect to better load balancing and improving parallelism.

3.3.2. The agent execution environment. A mobile agent system, provides a set of distributed Agent Execution Environments (AEE). A network of AEEs can be build up by run-time invocation of AEE. AEEs can be seen as computational environments, interconnected by some communication medium, supporting execution of mobile agents. It has also been called Agent Meeting Point (AMP) (Chess *et al.*, 1995), since it is a common place for agents to meet and interact. An AEE contains at least one execution *engine*. An engine sup-

ports computational resources needed for agents execution (e.g., an interpreter or compiler).

AEE defines a set of *locations* (places (White, 1994)) where agents can run. In the simplest case the location and the engine are the same. But locations may be abstract *places* which are transparent from the physical location of the engine. This means that several locations can exist in the same execution engine or one location can be distributed between many engines. Locations themselves may also move between the engines. A set of services is often related to a location. Agents move between locations. Agents' mobility is supported by starting, stopping, and suspending the execution of agents in preparation for moving from one location to another (may be to another engine). An agent migrates to a location in order to get a specific service or to meet another mobile or stationary agents. *Stationary agents* can represent services, an operating system, a database interface, and so forth. Representing all the entities in an AEE as agents has the advantage of having agents as the basic and only parties of interaction.

Interagent communication mechanisms are needed for interaction between mobile agents meeting in a location as well as between the static and mobile ones. Mobility can not be fully exploited without a reasonable inter-agent communication mechanism. Much like OS supporting inter-process communication, AEE must provide some sort of inter-agent communication. Communication can be:

- *synchronous vs. asynchronous*, depending on weather communication primitives are blocking or not,
- based on *message passing or shared memory model,*
- *local* (agents are on the same platform) vs. *remote* (agents are on different locations),
- *direct* (the sender has to know the address of the receiver) vs. *indirect.*

Beside the inter-agent communication mechanisms, the need for an *Agent Communication Language* (ACL) (e.g., KQML) is important in today's open network. It provides interoperability between different mobile agent systems and other agent systems. All the applications which are connected to the AEE can have an agentified interface, at least at the level of handling ACL messages. This gives a uniform interaction pattern for the whole system. KQML-like ACLs are usually asynchronous, based on message passing, for remote communication and direct. A *Linda*-like approach, as proposed in (Lingnau and

Drobnik, 1996), can be synchronous, based on shared memory model, local and indirect.

Communication requires names for entities in a mobile agent system, including *AEEs, agents, locations, services* and so forth. Naming and addressing can be provided by name servers (e.g., *traders*). In addition they may support run-time discovery and querying of agents by name, by their characteristics, or by their capabilities. Run-time generated random keys (e.g., in MOLE) and prefixing some global name space (e.g., Internet, like in Lingnau and Drobnik (1996)) are two examples of generating namespaces.

Security is an important requirement for a mobile agent system, since the mobile agents have to be evaluated in an open and execution safe environment. Two cases regarding the security issue in a mobile agent system can be pointed out, *AEE security and agents privacy and integrity.*

- AEE security can be treathened by a malicious agent. AEE must protect its resources. AEE security can be accomplished in several layers. The basic layer of security check is at the level of the implementation programming language of the agent. The code can be verified and checked in respect to the security issues, examples are Java bytecode verifier (Sun, 1995), and Safe-Tcl (Gray, 1995). In Safe-Tcl, the interpreter keeps the unauthorized agents away from accessing the Tcl normal interpreter which can handle tools for reading and writing files and process control and memory reconfiguration. The Safe-Tcl can call the interpreter through procedural calls. At the higher levels the agent can be authenticated and identified, by a security manager (Aglet (Lange and Chang, 1996)) or a firewall agent (TACOMA (Johansen *et al.*, 1995)), for authenticated access to AEE resources.
- *Agents privacy and integrity:* there may exist mechanisms to protect agents from a malicious AEE which tries to violate the mobile agent's privacy and integrity, or deny services or execution. The AEE can examine the content of the agent's code and data in order to discover the intentions of the agent. For instance, in the case of a electronic market, AEE can discover the intention of the agent to buy something for X$ and then it can change the offer on the basis of the obtained information. Violation of privacy and integrity is directly connected with controlling read and write access to code and data. Mechanisms for protecting the

agent can be obtained by using cryptography with digital signatures, encryption, and authentication.

A user can communicate with AEE and track her mobile agent through an *User interface agent*, which can be a stationary agent executing on the local host. Some other issues a mobile agent system may address include resource *consumption control* (there must be some mechanism to stop an agent to exhaust the resources in the AEE), *trading, accounting, semantic routing, orphan detection, agent termination, group semantics of agents* and *yellow page services.*

Standardization efforts are going on by the Object Management Group (OMG) under the name Mobile Agent Facilities (MAF) (OMG, 1995). Although OMG's MAF does not cover all the issues mentioned above, it has an intention to enhance the current distributed object technologies (based on distributed stationary agents, interacting through synchronous message passing) with active mobile objects interacting through asynchronous message passing.

3.3.4. Examples of mobile agent systems. Some examples of current mobile agent systems are mentioned in this part.

Telescript (White, 1995) is the first commercial mobile agent programming language. It was initiated originally by Apple for designing electronic marketplaces. It is a pure object oriented language, and runs in a Telescript engine. The basic class in Telescript is a process running in a node, two important subclasses are

- places: provide meeting locations for agents,
- agents: is a process which can migrate between places.

Telescript was the first language which supported agent-based programming for a Personal Digital Assistant (PDA), it was meant to operate on the AT&T personal link, later on it offered even an Internet solution.

Agent Tcl (Gray, 1995) is a transportable agent system. The agents migrate from machine to machine using the *jump* command. In addition to migration Agent-Tcl supports

- message passing and direct communication between agents,
- stationary agents can be written in C/C++.

Mobile agents are encrypted and authenticated using PGP. Safe-Tcl enforces the access restriction to the normal Tcl interpreter.

TACOMA (Johansen, 1995) provides support for agent-based computing. It

is written in C and based on UNIX and TCP. TACOMA uses abstractions like *briefcase, folder, meet.* An agent can pack everything it needs in a briefcase in preparation to move. Each briefcase contains a set of folders for keeping data and programs. Each briefcase contains at least the following folders:

- HOST: where to meet the other agents,
- CODE: filled with the code to be executed in the remote host,
- CONTACT: a compiler agent executing the CODE.

A Scheme compiler agent (CONTACT) in a Sun station (HOST) can execute a Scheme code send from a DEC station. Executing "meet B with *bc*" by an agent will cause the *bc* to be transferred to a remote host specified in the *bc*, the proper compiler will be contacted and after compiling the code, it is possible to interact with agent B. A firewall agent has the responsibility to check the security when a briefcase arrives. TACOMA supports agents written in C, Tcl, Perl, Python, Scheme.

MOLE (Strasser *et al.*, 1996) is an agent system based on the Java language. It consists of a set of agent servers. In each server there is an engine based on the top of the Java-interpreter which manages locations executed on an agent system. Each location provides a meeting place for agents. Agents are divided into service agents (static agents) and user agents (mobile agents). Only low level communication facilities and some kind of yellow page service is supported.

Aglet (Lange and Chang, 1996) consists of the aglet (an agile little agent written in the Java programming language), the Java Aglet API (JAAPI) and the Agent Transfer Protocol (ATP) that make the aglet secure and mobile on the Internet. It is a visual environment for building network-based applications that use mobile agents to search for, access and manage corporate data, and other information. The ATP and API form one of the proposals to OMG's MAF.

A HTTP-based mobile agent system (Lingnau and Drobnik, 1996) focuses on communication issues in the mobile agent framework and uses Linda-based communication for agent interaction. The Linda tuples constitute an information space where agents can share information and communicate.

3.3.5. Application areas. At the end of this section, some application areas for Mobile agent systems are mentioned:

- *Network management*, mobile agents representing management scripts enable both temporal and spatial distribution of management activities

in distributed network management.

- *Intelligent Network (IN) services*, can be realized by enhancing the existing Service Control Points and exchanges by AEE in order to enable a more rapid and more customized provision of IN-based services, implemented as mobile agents (Magedanz, 1996).
- *Information retrieval*, can be much more efficiently supported if an agent representing a query can move to a server for accomplishing non anticipated queries.
- *Smart networks*, as proposed and discussed in Harrison (1994), offer new services which are tuned to the subscribers preferences. Services are personalized and this can be achieved by using mobile agents as a component in the agentified structure of a smart network.
- *Electronic commerce*, agents roaming in a network of electronic marketplaces, representing desires of their users, negotiating and offering the cheapest price. This application can be highly exploited if the legal and security aspects of Internet would be solved.
- *Mobile computing*, mobile agents are often mentioned in conjunction with mobile computing devices, like PDAs. These devices have their main limitation in memory and computing capacities and are used most of the time "off-line". Mobile agents are therefor a good solution for enhancing the function of these devices since the agent can represent the user in the network and can accomplish tasks while the device is not connected to the network.
- *CSCW*, mobile agents can be used as migratory applications, which take their user interface and application contexts with them and continue from where they left off, building a good base for cooperative work. They can act as a natural media for people to interact and cooperate.

Almost all of these applications can be handled by stationary programs and traditional techniques and methods like the RPC method. Aside the better performance which mobile agents can offer, they associate a real life metaphor and give a more flexible system for realizing an agent-based computing environment where legacy systems, stationary and mobile agents can communicate and cooperate.

4. Is mobility sufficient for mobile agents? It can be argued that at the technical level, a mobile agent has almost no conceptual attachment with agents

in other fields. However, talking about the applications of the technology, leads us to recognize common features and conjunctions with agents in other fields.

The platform for mobile agents is often an open distributed environment (e.g., Internet). The network has become a single vast environment. Mobile agents must survive and compete in this dynamic and uncertain environment. They should be able to adapt and reason about the unpredictable scenarios. This requires that self-awareness and adaptation must be added to the functionality of mobile agents. Adding *reflective behavior* (Tyugu and Addibpour, 1996) is a possible solution to maintain the needed intelligence in mobile agents. Reflective agents have been studied in both DAI and OBCP. Mobile agent should be able to migrate even between heterogeneous borders of distributed systems. This suggests an integration of mobile and heterogeneous agents.

REFERENCES

Baumann, J., C. Tschudin and J. Vitek (1996). In *Proceeding of the 2nd ECOOP Workshop on Mobile Object Systems.*

Bond, A., and L. Gasser (1988). *Readings in Distributed Artificial Intelligence.* Morgon Kaufmann, Los Angeles, CA.

Brooks, R. (1991). Intelligence without representation. *Artificial Intelligence,* **47**, 139–159.

Burkhard, H.D. (1995). Agent-oriented programming in open systems. In Michael Wooldridge and Nicholas R. Jennings (Eds.), *Intelligent Agents, Theories, Architectures, and Languages,* Vol. 890. Springer-Verlag Lecture Notes in AI.

Carzaniga, A., G.P. Picco and G. Vigna (1996). *Designing Distributed Applications with Mobile Code Paradigms.* Technical report, Politecnico di Milano.

Chess, D., *et al.* (1995). *Itinerant Agents for Mobile Computing.* IBM Research Report.

Franklin, S., and A. Graesse (1996). Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents. In M. Wooldridge and N.R. Jennings (Eds.), *Proceeding of the 3rd. ECAIT96 Workshop on "Intelligent Agents, Theories, Architectures, and Languages".*

Gasser, L., and J.P. Briot (1992). Object-based concurrent programming and DAI. In *Distributed AI: Theory and Praxis,* Kluwer Academic Publishers, Boston, MA. pp. 81–108.

Genesereth, M.R., and S. P. Ketchpel (1994). Software agents. *Communication of the ACM,* **37**(7), 48–53.

Gray, R.S. (1995). Agent Tcl: A transportable agent system. In *Proceedings of the CIKMT95 Workshop on Intelligent Information Agent.*

Harrison, C.G. *et al.* (1994). *Mobile Agents: Are they a Good Idea?* IBM Research Report, RC 19887.

Hayes-Roth, B. (1995). Agents on stage: Advancing the state of AI. In *Proceeding of IJCAI'95,* Vol. 1. Montreal. pp. 967–971.

Hewitt, C. (1977). Viewing control structures as pattern of passing messages. *Artificial Intelligence,* **8**(3), 323–364.

Johansen, D., R. van Renesse and F. B. Schneider (1995). Operating system support for mobile agents. In *Proceedings of the 5th IEEE Workshop on Hot Topics in Operating Systems,* Orcas Island, Wa, USA.

Labrou, Y. (1996). *Semantics for an Agent Communication Language.* Ph.D. thesis, CSEE, Univ. of Maryland Graduate School, Baltimore, Maryland.

Lange, D.B., and D. T. Chang (1996). *IBM Aglets Workbench.* IBM White Paper, IBM Corporation.

Lingnau, A., and O. Drobnik (1996). Making mobile agents communicate: a flexible approach. In *Proceeding of the 1st Annual Conference on Emerging Technologies and Applications in Communication.* Portland, Oregon.

Maes, P. (1994). Social interface agents: acquiring competence by learning from users and other agents. In *Working Notes of the 1994 AAAI Spring Symposium on Software Agents.* AAAI Press, Stanford.

Magedanz, T., *et al.* (1996). Intelligent agents: an emerging technology for next generation telecommunication. In *IEEE INFOCOM.* San Francisco, California, USA.

Nwana, H.S. (1996). Software agents: an overview. *The Knowledge Engineering Review,* **11**(3), 1–40.

OMG (1995). *Mobile Agent Facilities.* In OMG Common Facilities, Object Management Group.

Rosenschein, S. (1989). Synthesizing information-tracking automata from environment descriptions. In *Proc. of the KR Conf.* Toronto, Canada.

Russell, S., and P. Norvig (1995). *Artificial Intelligence: a Modern Approach.* Prentice Hall.

Shoham, Y. (1991). AGENT0: A simple agent language and its interpreter. In *Proceedings of the Ninth National Conference on Artificial Intelligence.* Anaheim. pp. 704–709.

Strasser, M., J. Baumann and F. Hohl (1996). MOLE-A Java based mobile agent system. In Baumann *et al. (Eds.), Proceeding of the 2nd ECOOP Workshop on Mobile Object Systems.*

Sun (1995). *The Java Language Specification.* Sun Microsystems.

Tyugu, E., and M. Addibpour (1996). Declarative reflection tools for agent shells. *Future Generation Computer Systems (Special issue on Reflection and Metalevel AI Architectures)*, **12**(2&3), 203–215.

White, J.E. (1994). *Telescript Technology: The Foundation for the Electronic Marketplace.* General Magic Inc., White Paper Vol. 1,2,3. Mountain View, CA.

White, J.E. (1996). A common agent platform, submitted to the joint. In *W3C/OMG Workshop on Distributed Objects and Mobile Code.*,

Wooldridge, M., and N. R. Jennings (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review,* **10**(2), 115–152.

Received November, 1996

M. Addibpour graduated from the Royal Institute of Technology in Sweden in 1995 with a M.Sc. in Computer Systems Engineering from the School of Electrical Engineering. He joined the Software Engineering Lab. at the Department of Teleinformatics in 1994, where he is currently a Ph.D. student. His research interests include reflection and mobility in software agents.

PASTABOS APIE PROGRAMINĖS ĮRANGOS AGENTUS IR MOBILUMĄ

Matin ADDIBPOUR

Straipsnyje trumpai apžvelgta sparčiai vystoma programinės įrangos agentų tematika ir pateikta mobilių agentų sistemų lyginamoji analizė. Mobilumo sąvoka analizuojama mobilaus kodo ir jo ryšių su išskirstytais skaičiavimais kontekste, nagrinėjamos potencialiai galimos šios sąvokos taikymo sritys. Pabaigoje aptariamas poreikis derinti mobilumą su kitomis sistemų savybėmis.