Mobile Agent Architectures

Mats Persson

January 16, 2001

Abstract

Mobile agent technology is a new research field. The purpose of this report is to take a closer look of this new field, identify its problems, and describe its current state. The theoretical section of this report defines the three properties of agents: mobility, executability and autonomy. The security implications of these properties are analyzed, and some suggestions for protection techniques are given. A hostile mobile agent can act like a virus, infecting the computer and doing serious damage. Many mobile agent systems have been developed in recent years but several of them have been discontinued, because of security issues and an immature research field. This report also suggests that the mobile agents can be seen as a promising new paradigm for distributed programming of future computer networks.

Contents

Summary

Mobile agents are small intelligent programs that travel around in a computer network. They are given instructions by the user or programmer and then wander out in the network to accomplish their assigned task. They can be told to collect information, report problems, perform computations or modify existing programs on other computers in the network. In order to work, the mobile agents need a special infrastructure in the network that handles the execution and transportation of the agents.

Mobile agents are a new technology that has emerged with the increased use of computer networks and the arrival of the Internet and web technologies. One of the first uses of this new technology was searching the Internet for the lowest price of a product by asking several companies for the price of their product and then reporting back the lowest price. Since then, many other new uses have been presented: surveillance systems, information retrieval and mobile services.

In this report mobile agents are seen as both a technology and a new abstract mechanism. Three properties are defined for this mechanism: *mobility*, *executability* and *autonomy*. Mobility is the ability of the code to move from host to host in the network; executability is the ability to execute as program code; and autonomy is the ability to act on its own and make decisions. Some of these properties also exist in computer viruses and worms, which has been a serious threat to computers in recent years.

The negative side of the agents is that they can be very dangerous if no protective countermeasures are taken. Hostile agents can enter computer networks and do serious damage. In this respect they are similar to viruses and worms. Luckily, there are methods of protection, which are described in this report. Briefly, the important methods of protection are: *code signing*, where the signature is checked before executing the code; *path history*, where the trustworthiness of previously visited sites is considered; *proof carrying code*, where the code is proven secure; and *sandboxing*, where the code is executed inside a protective and limited area. There are also methods of protecting the agent from the environment in which it is executing, but these methods are still being researched and some problems are even considered impossible to solve.

The future of the mobile agent systems is not bright in every respect. A large number of agent systems have been developed, many of which have been discontinued, and the agent systems have not yet been a success on the Internet. The reason for this is uncertain, but it is likely to be the security problems and the fact that the technology is still in its infancy. However, the mobile agent technology might be considered in some military systems, where they can be important for maintaining redundant and robust databases. Mobile agents can also be used for active surveillance of either the real battlefield or the network itself.

Computer networks need an effective paradigm for using them, and the mobile agent mechanism is a possible candidate. Computing is no longer localized to a single central computer; instead processing takes place in the

distributed environment of the network. As a new paradigm it would be placed at the same importance level as programming languages or object orientation, although at a higher abstract level.

Intelligent software agents and multi-agent systems are other types of agent systems that are not mobile and cause fewer security-related problems. Also, they are often focused on the artificial intelligence aspect and are therefore not covered in this report.

1 Introduction

This report is a part of the ongoing research into mobile code in the project "Computer Security Specific in the Defence" at the Swedish National Defence Research Establishment. One of the activities in this research is to study the security of mobile code. This is an emerging and important field within computer science, and security is often of military concern.

This report can also be seen as a summary and evaluation of the mobile code field, with a focus on the current trends of mobile agent systems and their major problems with security issues. Such security problems are often similar to the problems with computer viruses that are common today. This report tries to discuss at a more abstract and theoretical level why we have these problems.

In 1998, I wrote a report called "Mobile Code and Safe Execution" [1], where several different languages and systems where studied with respect to their ability to safely execute mobile code. The sandbox models of Java, ActiveX, Perl, SafeTcl and a normal operating system (Unix) were compared and evaluated. One of the conclusions of the report was that it was possible to construct an acceptable and safe execution environment for mobile code, but that the existing implementations still had some problems.

The concept of code that moves from one computer to another is not new. In its most primitive form, you take a computer program stored on some medium, move it to a computer and then execute it there. With the advent of computer networks in the seventies, it became easier to move program code by using the File Transfer Protocol (FTP). Examples of this are remote batch jobs and the use of PostScript to control printers. Nevertheless, the code moved too slowly to give any obvious advantage. Security issues were neither foreseen nor considered.

In recent years, the speed of the networks has increased tremendously, and we now also have the World Wide Web (WWW), which is a new way to transfer information. It became more common to move code via the WWW, but also via email. The developers probably did not understand how powerful mobile code was, and the technology arrived more by natural selection. The more powerful software had some kind of mobile code, which was sent over the network or into itself, modifying its internal structure and state. Processes in an operating system are an example of the latter.

In the early nineties there was a boom in research on mobile agents and many different agent systems were designed and developed. Some of these projects and systems have now disappeared, but the field is still alive. In recent years, several standards have arrived, both for the communication languages (FIPA, [25]) and the mobile agent platforms (MASIF [26]). The programming language Java has almost become a de facto standard for the agents and the system itself.

But there is still no real "killer application" for agents. No application has really shown how powerful agents are or become a huge success. Instead, the mobile code technology has trickled into many applications to enhance their functionality. In a way it has become more of a programming paradigm than a technology.

It should be noted that this report is not about intelligent software agents that are based on the distributed artificial intelligence paradigm, although they have several features in common with mobile agents.

1.1 Overview

This report is divided into several sections describing the theory behind agents and the current state of the art in the mobile agent field. The first section is the introduction, which also contains two problem descriptions. The second section is a more theoretical description of the concept of mobile code. The third section is a brief description of agents and is followed by a section about agent architectures, which contains a fairly large description of one specific architecture. The report ends with Conclusions and a Future Research section. A number of words and abbreviations are defined on page 28.

The security issues are discussed mainly in sections 3.4 and 5.4. Some military uses of mobile agents are mentioned in sections 4.6 and 5.2.2.

2 Background

As a further background, two general problems will be explained to give the reader a better understanding of the ideas in this report.

2.1 The complexity problem

Computers have become more and more complex over the years. Not so much the hardware, like processors which has mostly just become faster, but the computer programs and the network have made the whole area much more complex. This trend is likely to continue. Yet ordinary users want computers that are easier to understand. Once they understand how to use a certain technology or program, it immediately becomes more complex. The likely reason is that once understood there is room for more complexity. You could actually formulate this as a law:

Law: The complexity of computers will continue to increase such that they are always one step ahead of the users ability to handle them.

The consequence is that computers will never become easy to use, unless you are an expert and have at least some basic programming skills. The required level of skill can be seen in the levels of complexity, which can be defined in ascending order as follows:

manipulate	Simple form of interaction with a computer by pointing and clicking, or giving simple commands via the keyboard.
configure	Modifying the internal configuration of the computer, and by this changing its behaviour.
program	Programming a computer.
autonomisize	(New word). Making certain tasks autonomous or automatic. Making program components that are autonomous.

These levels can also be seen as abstraction levels. The last level (autonomize) is a newly introduced concept that I hope will be understood after reading this report. The concept of object-oriented programming (OOP) is somewhere between "program" and "autonomize" above. One of the ideas in this report is to see components, OOP and CORBA, as steps on the road to autonomous agents or some other form of mobile code. In the industry, component programming has become popular, as it is quite easy to program with components [8].

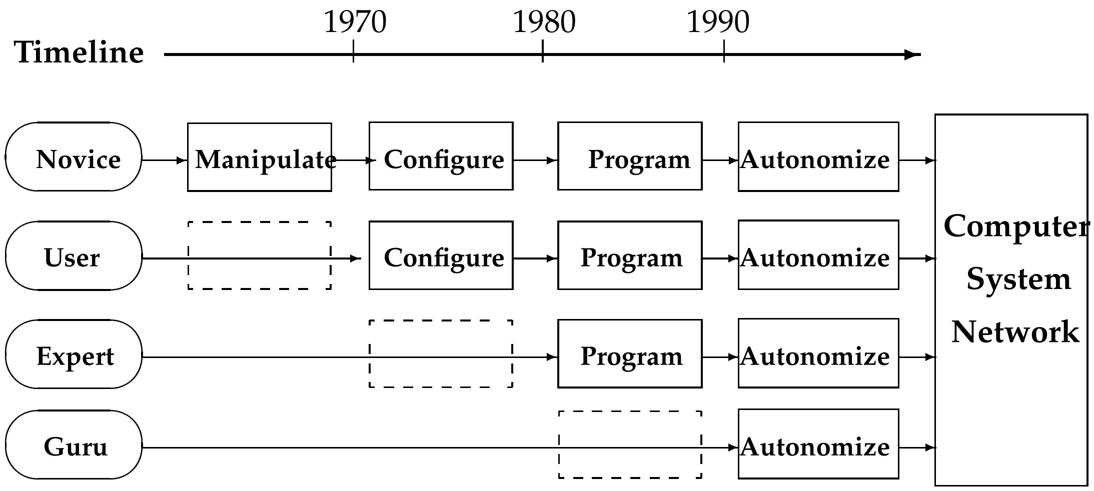

Figure 1: Complexity levels

The picture in figure 1 needs some explanation. It shows how different types of people operate a computer on different abstraction levels. The unskilled novice operates by pointing and clicking (manipulating), which affects the configuration, the programming and its autonomous parts. On the other hand, the skilled guru operates, or rather makes, the autonomous parts. The time line at the top of the picture shows the state of the complexity at that time. The required level of skill of an expert to program a computer in the sixties was almost at the same level as a novice manipulating a computer today, assuming that many people today have basic computer knowledge.

According to the picture above, the future lies in autonomy. That is not the only possibility. The network is also in the future, and autonomous programs wandering around in the network seems to be the most likely prediction. This also seems to be unavoidable, if we want to use the power of the network and computers to their fullest.

Web-browsers like Netscape and Internet Explorer, or programs like Microsoft Outlook, are well-known examples that use this autonomy, and where you can see it most clearly. These programs are also the place where you can see the security problems, which will be discussed in later chapters in this report.

2.2 The unpredictability problem

A computer that gives different results from day to day or just behaves erratically is nearly useless. Correct results are expected. Therefore, computers have been made intentionally deterministic to make them more predictable, but also easier to analyse. Engineers prefer to make synchronous hardware instead of asynchronous, and software engineers want tighter syntax and typed computer languages to make it easier to find bugs. Neither do they want the unpredictability of computers that reprograms themselves.

Likewise, ordinary users of computers often do not like the sometimes very erratic behaviour of computers. They do, however, more easily accept the same behaviour in other humans. Predictable behaviour is expected of computers.

In order to make the point even clearer; let us look at a programming language. Perl is probably one of the least liked programming languages among the hard-core computer scientists. It has quite a dynamic syntax and has some constructions that are context sensitive. This makes the language harder to parse. It can also turn a programming mistake into unpredictable results or behaviour, without giving any warning or error. The "magic" in this language is one manifestation of the unpredictability problem.

When you release a mobile agent in the network, you lose control over it. Even if it is programmed to follow a certain path, the network connections or the computers can go down, causing the agent to get lost. If you let the agent make its own decisions, it can take another path if one path is blocked. Whichever way you do it, the mobile agent has an unpredictable behaviour. You cannot trust it, regardless of any security precautions.

3 Properties of mobile agents

In order to get a better understanding of mobile agents, three properties will be defined and explained in this section. They are mobility, executability and autonomy. A mobile agent can be defined by using these three properties. This definition of three properties is also useful because each property can be analysed regarding its security implications. In an earlier work [16], a similar taxonomy has been made with slightly different properties.

Mobile agents are actually a subset of the larger field of mobile code. A different way to classify mobile code is to divide its behaviour into four distinct design paradigms [2]. These are the well-known "Client-Server" paradigm, where the code does not move at all; "Remote Evaluation", which sends the code to another site where it is executed and the result is sent back; and "Code on Demand", where the code is downloaded from a distant site and executed on the local machine. The fourth paradigm is "Mobile Agents", which are small programs that wander around in the network. This classification system only helps us understand how mobile code works and where it came from. It is not very helpful when studying its security implications.

For more practical implementation issues, you can also look on a different abstraction level, where there are four mechanisms in mobile code: mobility, communication, translation/execution, security [3]. These are not explained in any depth in this report.

Mobile code can perhaps be seen as the new paradigm for solving the complexity problem in section 2.1 above.

3.1 Mobility

The word "mobile" comes from Latin and means something that moves quickly. At the other end of the scale, code moves very slowly when transported on a diskette or tape. It moves somewhat faster when moved by the FTP mechanism. It is not slow because of the network, but because it often involves a manual operation like pointing and clicking or giving a command. If the transportation is without any manual work, it is only dependent on network speed. Note that program code that needs to be compiled before it is executed is moving slower than an interpreted language, because compilation often requires manual intervention. This shows that there are degrees of mobility. The opposite of mobile is stationary.

If something moves faster, it is usually also more effective. Information is more effective the faster it moves and spreads, and the same would apply to code that moves. Anything that moves fast also naturally becomes more dangerous. Stopping a virus attack that spreads from computer to computer within milliseconds is quite hard.

For the data to be able to move, it needs a channel. A slow channel mentioned above is FTP. A fairly fast channel is WWW, but it requires that the user is active, except for some smart Java applets or JavaScript that keep the connection alive even when the user is inactive. Email is quite a fast channel because the emails are handled automatically by the mailservers, and one email can travel around the world in a couple of minutes. The fastest channel is of course direct TCP/IP, or UDP, which can be even faster in some cases.

Sometimes in the literature and texts about mobile code you see the two notions, weak and strong mobility. According to another report [2] they are defined as follows: strong mobility is the ability of a system to move both the code and the execution state to a different host, where as weak mobility is the ability to move only the code. That ability should fit better in the next section about executability.

Figure 2: Code moves from site to site

3.2 Executability

If a text or data can be understood or interpreted by a computer, it is said to be executable. A binary program can be executed by a processor and a script can be interpreted by an interpreter, which is a special program designed to execute scripts. There is a duality between code and text. If lines of text can be understood by both computers and humans, it is called a programming language.

There are degrees of executability which can be divided into several levels in ascending executability. The level classification is taken from the previous report [1].

- **Text** contains information that is readable by humans only. An AI program might understand it, but it contains no structured information.

- **Marks** can be instructions inside the text denoting how to interpret it. Usually, this is just a few characters, such as how to write a word in bold text. In LaTeX it is `\textbf{Marks}`.

- **Macros** are instructions that can generate a piece of text. For example, a macro like `<date>` would generate today's date and insert it into the text.

- **Controls** are instructions for how data should be interpreted. They can declare conditional text, define variables, and change them. Basically, they have taken over the control of the interpretation of the data.

- **Scripts** is written in complete interpreting programming languages, which are Turing machine equivalent. To put it more simple, any program can be made in this language. In a script, you can control the execution point with loops and jumps, as well as accessing many system functions.

- **Byte code** is virtual machine code that is interpreted in a virtual processor. At this point the code is really not readable by humans.

- **Machine code** is instructions for the processor in the computer. Machine code is often the result of a compilation. This level is usually the lowest you can go, even though in some processors it is possible to write micro code, which is instructions for the gates and subsystems inside the processor.

The higher executability of the data, the more you can manipulate the computer when it is executing the code. This also means that the lower the executability, the easier it is to protect yourself. The reason is that in machine code it is harder to put wrappers or extra layers over sensitive system functions.

This classification is useful when evaluating the security level in mobile code. You can also compare this classification with the complexity levels in section 2.1 where the ordinary user understands "Text" or maybe the "Marks" executability level. However, it is questionable whether this comparison is meaningful.

3.3 Autonomy

When something is autonomous, it is acting on its own without any outside control. It is responding, reacting, or developing independently of the environment. A piece of program code and data is self-contained when it has encapsulated the code, data and environment into a unit. The environment is the current program state with its variables and bindings to resources. A binding can, for example, be a reference to a database or a local printer.

An autonomous unit must also have a certain level of intelligence, so it can react properly to its environment. This often means some kind of rule-set.

One example of computer programs acting on their own is automatic program updates where the program connects to a server, downloads new code and updates itself. This can, of course, be very practical, as you do not have to worry about the latest updates.

There might be different degrees of autonomy, but no such classification is done in this report. There are differences in the intelligence in autonomous units, and the automatic updates are a simpler form of autonomy. Asynchronity might also fit into this classification.

3.4 Security implications

For each property described above, there are obvious security implications. The higher degree of mobility, executability and autonomy a piece of data has, the bigger threat it is to security. Fast-moving code can more effectively damage a networked computer system. Machine code can get closer to the operating system and kernel by calling system functions, or even replacing some parts of it. An autonomous piece of code can adapt to different environments and find better paths of attacks if the system is well guarded.

If you can identify the threats, you can design proper protections against them. By slowing down the process of dealing with incoming data, you get better protection against an attack, for example requiring authentication before admitting the data. This is an unusual way of looking at the authentication process. An attacker with a piece of hostile code will try to find a faster path of attack if it is met with an authentication blocker.

To protect against the executability property, you can, for example, remove all special characters in emails and only allow the characters A-Z and a-z. It becomes harder, but not impossible, to have a language that can interpret only those characters. Machine code will be completely ruined.

It is harder to defend against the autonomy property as you need hosts that are smarter than the agents. One way is to fool the agents somehow, for example by giving wrong or incomplete data, or special information that only your own agents can understand.

3.4.1 The virus problem

A virus is mobile and executable but not very autonomous because it is dependent on a host program that it attaches itself to. When a virus is independent, it is usually called a worm. In recent years viruses have become a major problem and have caused severe monetary damage in lost data and work time. The normal way of handling viruses is to run virus scanners that search the computer for code with certain signatures that indicate that it is a virus and then delete it. Unfortunately, this scanning is often done after the viruses have damaged the computer.

When an old known virus arrives into the computer, the virus scanner usually catches them before any harm is done. But a new virus is problematic. An old virus is drastically reduced in speed and executability by the virus protection software.

A better protection against all viruses would be to restrict their speed and executability. In section 5.4 in this report, some protection methods will be shown.

Why are there almost no viruses in the Linux operating system while they are thriving in the Windows operating systems? The reason is that in Linux there is no fast channel for the code to enter the computer. Linux does not have automatic updates and has no executable email. Even if a virus arrived it will have trouble with the Linux security levels, which limits its executability.

3.4.2 Break-ins and exploits

Many hacker attacks are done by making code execute where it is not supposed to. A typical exploit is to use a trick called “buffer overrun”. A server accepts incoming data and stores it in a buffer, an area in the memory where the data are stored. The common mistake is to make this buffer too small and without checks for writing outside the buffer. If someone sends too much data at once, the last bytes of the data are placed where they are later executed. These last bytes are where the hacker places the machine code that makes the break-in. You can see this as another form of mobile code.

4 Agents

According to some literature, a software agent is a program that acts on behalf of someone else, or to put it differently, a computing entity that performs user delegated tasks autonomously [9]. The first definition indicates that an agent has the same rights as the user that controls it, and the second that it only performs those tasks when asked to. These two definitions are a bit narrow, because an agent can also be quite independent and do tasks on its own will and by this try to act more intelligently. A better definition is that an agent is a program that has the executability and autonomy properties defined in section 3.

Ordinary software agents are stationary and therefore only executable and autonomous, according to the definition in the previous sections in this report. Background processes or daemons in UNIX can also be seen as stationary agents. It is possible to see the agent as a generic term or even metaphor for a set of programs with certain properties. There are many types of agents that are described in the following sections.

4.1 Mobile agents

Mobile agents have the ability to move from host to host, executing at each place and then keeping the results before moving to the next server. A simple picture of mobile agents in a network can be seen in figure 3. To be able to move, there must be an agent server, sometimes called agency, on each host that handles the incoming agents and executes them. This agency is also responsible for sending messages between agents and does some authentication if necessary.

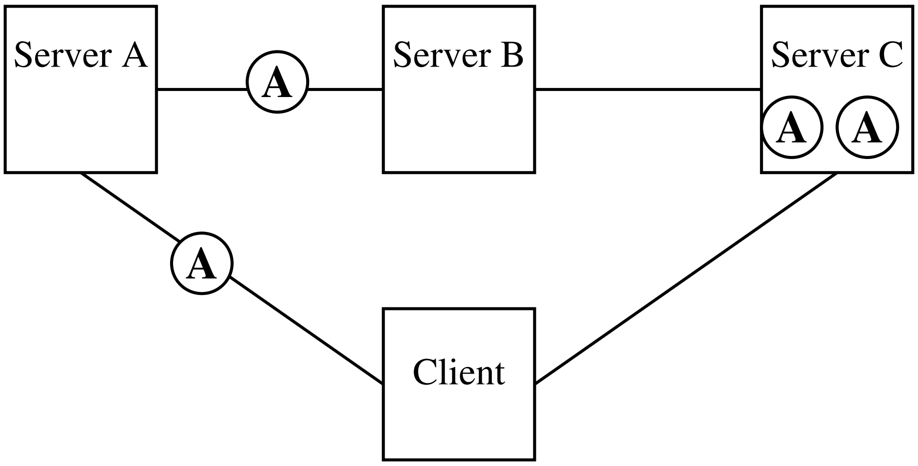

Figure 3: Mobile agents wander around in the network

According to the literature [10], there are many advantages with mobile agents.

- They reduce the network load by sending the code to the data host instead of sending the data over the network.

- They overcome network latency in real-time systems because they do not need a lot of bandwidth.
- They encapsulate protocols by bringing their own protocol code with them.
- They execute asynchronously and autonomously, and are therefore not dependent on a continuously open network connection.
- They adapt dynamically to new environments.
- They are naturally heterogeneous, and work well in a heterogeneous network.
- They are robust and fault tolerant. If a host shuts down, the agents can move to another host. They can also duplicate themselves and execute on several hosts in parallel.

4.2 Intelligent agents

Intelligent agents are supposed to be very strong in the autonomy and intelligence aspect. They are able to interact with other agents, the user, or their environment using a special communication language. They are able to make decisions using their internal knowledge base and some kind of reasoning rule system. They can adapt and react to a changing environment.

The Internet search engines and their search robots are successful examples of this kind of agents. Another example is the program that sorts incoming mail into different folders using user specified rules.

4.3 Distributed and mobile objects

Mobile objects are noted for their relative lack of autonomy, but they can still move their code in the network. They are related to distributed objects that do not move at all, but are spread out in the network. In these objects, a method can be accessed from a computer anywhere else in the network. To make this possible, they need a middleware that can keep track of where they reside in the network and give them proper names. This middleware is described in section 5.5.

4.4 Mobile code versus agents

Often in other literature and reports, the concept of mobile code is seen as a wider area than mobile agents; the agents are seen only as a specialized form of mobile code. According to the properties defined in this report, this is not so.

Mobile code in its pure form has neither autonomy nor any intelligence. It cannot remember what it did at its previous host and cannot make decisions on where to go next. Examples of this form of mobile code are applets on web pages, or any type of code on a web page.

4.5 Agent languages

The common language for writing agents today is Java and has become a de facto standard. Java is a general programming language similar to C++ with object-oriented features. It is not specially designed for agents as was the language Telescript and some other languages [11]. Note that there is a difference between agent programming languages and agent communication languages, where the latter type is not covered in this report.

Due to the dynamic features of agents it would make sense to have a real interpreting language like Tcl, Python or Perl, and not a semi-interpretive language like Java. Also, it would probably be easier to build a more secure environment with interpretive language. On the other hand, the Java language is easier to analyze because of its strong typing, while Tcl, Python and Perl have very weak typing or none at all.

Some agent languages like Telescript have some form of "move" command which gives them the ability to move the agent in the middle of execution and keeps its execution state and "program counter". This ability is usually called strong mobility. Some of the agent systems have languages with special functions for cloning the agent, receiving events or sending messages to other agents.

4.6 Agents in military systems

The robust and fault tolerant nature of agents makes them a viable solution to many of the problems in the military systems, especially with the new ideas in "Revolution in Military Affairs" (RMA) and its new command and control system based on new information technology.

Some of the possible uses of agents in military systems are:

- Reactive sensor networks [22]. By using something called Autonomous Networked Tactical Sentries (ANTS) in an array on the battlefield, it can detect enemy positions and movements.

- Logistics control [23]. Agent technology and heterogeneous distributed database systems cooperate to manage logistics information and materiel.

- Collecting and filtering information or data as a part of the information fusion process, or act as database mediators/agents.

- The agents can avoid network latencies which often occur in radio-based military networks, because they do not need continuous contact with other hosts.

- Redundancy handling in the command and control systems. Sometimes hosts get destroyed in a battle and the agents running on this host can have another redundant host where copies of the agents continue to run.

See also section 5.2 below for other possible application areas.

5 Agent architectures

The software architecture field is more abstract, being the level above the algorithms and data structures fields. Software architecture includes the global control structures, protocols for communication, synchronization, physical distribution, scaling and performance, and selection among design alternatives. It is on this level you see the common architectural styles like pipes and filters, object orientation, event-based models, and table-driven interpreters. Here you also see the distributed network architectures.

One of the alternatives in the design of the software architecture is how to access remote resources or make calls to remote objects; or how to send the program code over the network. Four different paradigms have been identified:

- **Client-Server**. This is the traditional paradigm where a client contacts a server and data are transfered. In this paradigm no code is sent at all.

- **Remote Evaluation**. In this paradigm a connection is made to a remote site. Code is sent for execution at the remote site, and the results are sent back. Resources on the remote host are used.

- **Code on Demand**. This is the opposite of remote evaluation. The code is fetched from a remote site, downloaded and executed locally. This is how many web browsers work when they, for example, download JavaScript.

- **Mobile Agents**. Here the code, including its execution state and some of its resources, is sent to a remote site where it executes. It can continue to another site if needed.

Currently, the client-server paradigm is the most common style, and it is even more refined in the distributed object-oriented systems like DCOM and CORBA. The code on demand paradigm is, of course, common on the web. By putting the mobile agent architecture as a separate paradigm above, it is made to look like a completely separate paradigm. This view can be a mistake as it has been shown that mobile agents as a replacement for client-server are not always a good idea. For example, the mobile agents may have to carry around the data in their walk around the network, while the client-server sends the data back immediately, which actually reduces the network load in many situations.

In the following sections the mobile agent architecture will be described and analyzed in more detail.

5.1 Agent machinery

In order to make a mobile agent system work, it is not enough to build the agents themselves. A program at each site is also needed to handle the incoming agents and send out agents. This program is often called an agency. The agency can be built differently depending on which type of agent system is needed, but a general architecture can be seen in picture 4.

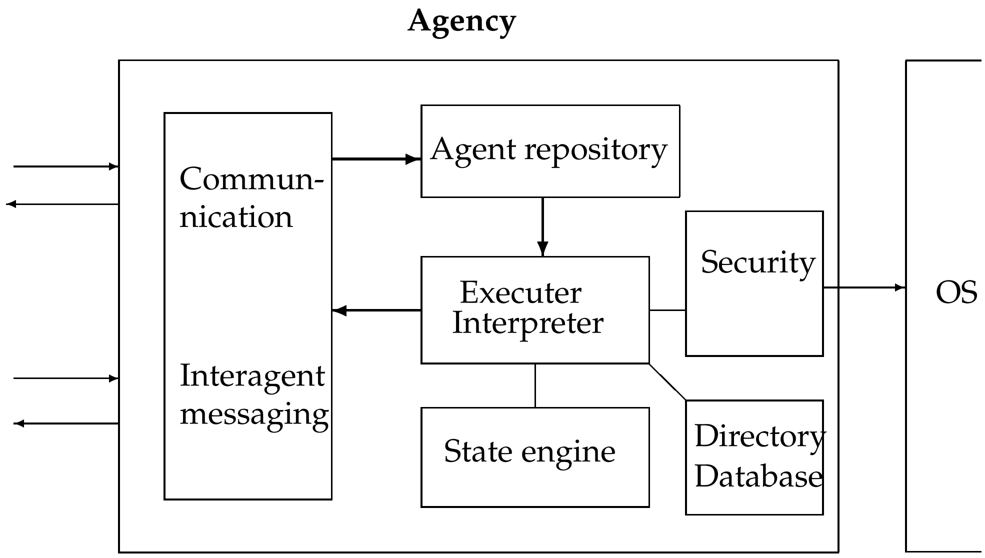

Figure 4: Generic mobile agent system architecture

The generic mobile agent system can have a range of varying components. It needs a communication module that handles incoming and outgoing agents, as well as the messaging between non-local agents. It has a repository that performs authentication, sets priorities and queues up agents for later execution. The executing module has an interpreter and can sometimes run agents written in different languages. The state engine contains the current state of the agency and can have some kind of rule or inference engine that decides what to do with the agents. It also handles local inter-agent communication. There is also some kind of database or directory where data are stored or retrieved by the agents. The security module acts as a kind of sandbox that keeps track of what the agents are allowed to do. It also monitors the agency. There can, of course, be security functionality in the other modules too, such as encryption in the communication module.

5.2 Application areas

One of the first ideas was to use agents for searching through the Internet for the lowest prices of products and services. While the idea was good in theory, few companies wanted other people's agents in their computers. Not only for security reasons, but probably for marketing reasons too. They wanted people to come to their place and keep them there. Agents are impersonal, and you cannot make deals with them or fool them.

In the middle of the nineties there was a slight boom in agent systems, many of which have disappeared now. So there is currently a slight lack of good applications for the agent technology, or at least not any well-known and popular application. The reason could be that the research in the area was not really mature or finished, and the technology was released too early at the start of the web revolution. It is possible that the real boom might not come until it marries with another field in need of a technology.

Although there are many suggestions for possible uses of agents they can generally be divided into four areas. Some of the following areas are not really mobile agents, but often the difference between agents and mobile code is not very large.

5.2.1 Information retrieval

One important activity many people want to do is to search for information on the web [18]. Collecting, sorting and moving information is probably the most important usage of the Internet, or any other network. The agents can act like search robots, wandering from site to site filtering and collecting information. There is also electronic commerce, where the agents locate products and services, compare prices, and negotiate price.

5.2.2 Remote control

Remote control applications are intended to control or reprogram remote computers, devices or unmanned vehicles by sending agents with new commands or program updates. These updates can be done very quickly, making it very good for applying security patches. Agents can also be used for monitoring devices and reporting back when status changes or problems occur, or can even be used for intrusion detection and active defence of computer systems [19].

A similar example of remote control is an abstraction called Mobile Streams [14]. Using that system, a distributed, event-driven application can be scripted from a single point of control and dynamically extended and reconfigured during execution.

5.2.3 Programmable Networks

The third application area for mobile agents or simply mobile code is to dynamically program the networks themselves in order to make them more flexible, customized and give them higher performance. At the lower level, the network devices like routers and switches can be remotely programmed [24] by sending mobile code which can change the topology and routing. In telecommunication networks, service components can be distributed with the help of mobile code [7]. Instead of being passive, the networks become more active by taking certain part in the computations or filtering the data [12]. This can be achieved by adding code fragments to the data packets.

5.2.4 Distributed systems

Instead of doing all the processing and computations on a central computer they can be distributed to several computers in a network. It is somewhat similar to process migration, but the difference is that processes usually migrate within a tightly coupled unit with several synchronized processors. The code is distributed to the remote computer to do the filtering and processing locally. This often reduces the network traffic and is also a way to balance the load of computers with different capacities. It can also be more redundant when several computers do the same processing and the results can be compared.

5.3 Examples of agent systems

About one hundred agent systems have been made to date [28], and there are probably many unofficial systems. About half of them have been made in academic environments and the other half in commercial companies. There are two interesting things to note here: the agent systems have been commercially developed at the same time, as it has been a hot and new research area. Also, most agent systems have been developed on Unix systems.

Many of the agent systems have disappeared by now, and the research and development on them have closed down, while others have found a small application niche and still survive.

Examples of agent systems include Aglets, D'Agents, ARA, Concordia, Java Beans, Mole, Odyssey, TACOMA, Voyager, Telescript and SHIP-MAI. More descriptions of these systems, and several others, and how they work can be found in [13]. Three systems will be shown or examined in more detail in this report.

5.3.1 Jumping Beans

The Java language has more or less become the de facto standard within the agent technology area. Most new agent systems are based on Java, and one of them will serve as an example on these. Jumping Beans is a commercial product from Ad Astra Engineering [27]. It is based on Java Beans which jump from computer to computer during execution. The beans are actually components or objects.

Jumping Beans is a framework, or API, which the developers can embed in their applications to add mobility to their projects. By embedding Jumping Beans, the applications can jump from computer to computer while they are executing, even if the application has never been installed on the target machine.

Each host computer that wishes to participate in the framework must have the Jumping Beans "daemon" software running on it, which listens to the TCP/IP port to receive mobile applications. In addition it performs many other tasks, including:

- Receiving incoming mobile applications.
- Enforcing security while mobile applications are executing.
- Dispatching, deactivating, and reactivating mobile applications.

- Providing Jumping Beans services to mobile applications and to the resident host software.
- Crash recovery when needed.

Jumping Beans has several uses. It can be a replacement for the SNMP (Simple Network Management Protocol), or provide automatic response for network events and load new modules when needed. It can also be used for remote network management by dispatching commands to remote devices, or for intelligent software distribution.

The Jumping Beans system aims for a high security level and has many of the usual security features of the traditional sandbox. Jumping Beans security is based on ACLs, user log-ons, certificates, public/private keys, digital signatures, and central management. While the mobile application is executing, security is enforced by the Jumping Beans client using Java security capabilities. During transport, security is enforced jointly by the central server and the receiving client. Having a central server that manages security is an easy way to solve many security problems, but as a critique of this product, this is not very flexible, scalable or robust.

This product has a thorough treatment of the security problems of mobile agents and it seems as if a security expert designed it, which was the main reason it is described in this report.

5.3.2 D'Agents

D'Agents is a mobile agent system from Dartmouth College in New Hampshire, USA, where it has been an ongoing research project for several years. It is developed on Unix, and the agents can be written in Tcl, Java or Scheme. To be able to run the agent system, you must have an agent server listening to a certain port on each computer. The agent server consists of an interpreter and a number of help agents that handle policies.

In 1999 a short paper was written [21] on middleware, where the D'Agents system was studied and evaluated in more detail. The following is a translation of some of that text.

Implementation

The programming languages in D'Agents have been extended with some extra functions to facilitate moving, cloning and message-passing between agents. The more important ones are shown in Table 1 below. There are also functions for handling events, but these are rather primitive, and the agent must handle most of the mechanisms itself. Figure 5 shows an example of an agent written in Tcl that uses some of these functions.

Security

D'Agents has several security mechanisms. It has authentication of the agents with PGP and a sandbox based on Safe-Tcl for the Tcl interpreter. The sandbox can be configured to allow only a certain number of hops

agent_submit	Starts a new agent on this or some other computer.
agent_jump	Moves this agent to another computer.
agent_fork	Clones a copy of this agent.
agent_send	Sends a message to another agent.
agent_receive	Receives a message.

Table 1: New functions

```
proc hello {} {
  global agent
  main create -name Hello -display $agent(actual-server):0
  button .button -text "Hello, World!" -command {set done 1}
  pack .button
  tkwait variable done
}
agent_begin
puts "Enter a nonnegative integer: "
gets stdin number
agent_submit $machine -vars number -procs factorial \
  -script {factorial $number}
agent_receive code result -blocking
puts "$number! is equal to $result\n"
agent_end
```

Figure 5: Example of an agent written in Tcl code

for the agents, how many clones they can make, how much time they can use, and several other configurations. Every agent that arrives at the agent server is first authenticated with PGP. Then its programming language is identified and the code is sent to the right interpreter and executed. When required the credentials of the agents are checked with another stationary agent called *resource_manager*, which searches for the name in a policy file. If the *resource_manager* finds the name, it answers "yes" or "no" depending on whether the operation is permitted or not. An operation can be, for example, `agent_submit` which starts a new agent, or an operation can provide access to the file system.

D'Agents uses the underlying operating system for certain parts of the security. Each agent runs as a separate process and can then be limited in time and memory by using some system calls of the operating system.

Security deficiencies

There are several deficiencies in D'Agents if you use the list of security requirements of mobile code in [1].

- *No logging of agents:* There should be some kind of log of incoming and outgoing agents, as well as executed agents and newly created agents.

- *No access control for files:* There is no system for handling access control for the files, so if the agent gets access to the file system, it can write and read everywhere the agent system has access. This way it can modify its own rights.
- *Unrestricted memory allocation:* There is currently no memory limit for the agents. It is easy to make an agent that consumes a lot of memory and that way makes a Denial-of-service attack. An example of this code can be seen in figure 6. It would be possible to use `ulimit` in Unix to set the memory limit.
- *Allows `eval` in code:* If someone with many privileges writes an agent that contains an `eval`, it can become a potential security hole. Luckily, the current implementation did not have this hole.

```
proc hog {} {
  set x 0;
  while {1} {
    incr x;
    set a($x) "test";
  };
}

agent_submit perseus -procs hog -script hog
```

Figure 6: Memory consumer

Design

Even though the D'Agents system is an experimental academic research system, there are some design issues that can lead to further potential security problems. It is written in C++, and there is not much documentation on it, which makes it harder to analyze or even get an overview of the system. The communication protocol between the agents has faults, and the errors are hard to find because neither the protocol nor the program code is properly documented. The implementation forks too often when it executes an agent, which creates unnecessary processes. Lastly, few error messages are given to the user, and it is hard to generate them in general.

5.3.3 Ant model

The idea of the ant model is to have a much simpler programming language for the agents, let them wander around more randomly and freely, and let them have a non-deterministic behaviour similar to ants. This contrasts with ordinary agents, which can have quite complex code both for execution and moving scheme. Moreover, the people who program them naturally want them deterministic. The ant model in this section is my own, but similar robot models or swarm theories exist elsewhere in research.

These ant agents should consist of three parts: a name, a piece of simple code, and a container for data. The ant agents could wander around picking up some data at one agency and dropping it off somewhere else. There could also be killer agents that destroy other agents, generator agents that create new agents, or clone agents that copy other agents.

In each agency there would be one heap where all the data are dropped or picked up. There would also be several functions that the agents could use for more advanced processing, calculations or system calls.

There would not be much security, except limits on the size of the agents, the size of the heap, or available system functions. Other agents would protect the agency by killing off unwanted agents.

The ant agents would not operate as single units but as a collective. Several agents working together could make a calculation. This would require quite different programming methods and a more distributed thinking by the programmer.

5.4 Security for mobile agents and other problems

The greatest challenge in mobile agent technology is probably the security problems [6]. The concept of letting your own agents enter other people's computers or letting other foreign agents enter your computer without proper protection, can be very dangerous. It would be quite easy to write an agent virus if there was no form of protection.

The security problems for agents can be divided into four types: protecting the host from attacks by the agent, protecting the agent from tampering by the host it is executing on, agent versus agent, and attacks from other entities on the agent. However, the last three types can be combined into one type of protecting the agent. The methods of attacks, which are commonly known within the security field, are disclosure, denial of service, corruption, and interference.

There are many useful solutions for protecting the agent host. Fortunately, many techniques from the traditional areas of trusted systems and communication security can be applied in an analogous way. Several of the following solutions come from an earlier work [15].

- *Signed code:* The agent is authenticated before it is allowed to enter the agency. This is done by letting the agent carry a digital signature of itself.

- *Path histories:* The idea behind path histories is to have a record of previously visited sites. By looking at this history, the agency can decide whether to trust the agent or not. Before sending away the agent, a new digital signature by the current host is added to the record.

- *Proof carrying code:* The producer of the code in the agent must supply a proof that the code is safe. The proof is verified before executing the code. In Java, a simpler version of this method is used, where the code is verified but the agent carries no proof, and instead relies on type safety.

- *Executing code in a sandbox:* The code in the agent is put in a constrained executing environment where it is limited in time, memory, range and duplication. The access to system functions is also quite limited or filtered through a protecting layer.

- *Logging:* If the incoming and outgoing agents are logged, it is much easier to detect and trace attacks from hostile agents. The logging is not really protection as it cannot prevent damage, but it improves the security level.

A much more difficult problem is to protect the agent from either of the other agents, the host or other entities, because there are no traditional techniques to be used as the problem of protecting an application from the operating system has never been considered. If there are no security countermeasures, a malicious agent host can easily examine the code in the agent, clone it or modify its code or data. A number of different techniques have been proposed, which can generally be divided into four types.

- *Data encapsulation:* The results of the agent's actions can be protected by the means of encryption and digital signatures. A trusted third party can also be involved by giving a timestamp. This encapsulation does not give any real protection but tampering with the data can be detected, and previous results stored in the agents cannot be read by the current host when the data are encrypted. However, it can be deleted or removed. One other problem with this method is that it requires quite large keys and signatures to be carried with the agent, but by using a technique with sliding encryption this load can be reduced.

- *Multiple agents:* In this technique one or more extra agent is created for every original agent that is created. One extra agent can act as a watch guard and keep track of the original agent's actions and path. If several agents are created, they act as redundant copies, and if the original agent is destroyed or tampered with, at least some of the extras may survive and continue execution.

- *Blackbox execution:* The code of the agent is encrypted and executed in such a way that no information of its algorithm is revealed. This technique is very powerful but does not protect against agent destruction or denial of service. The agent host can still fool the agent. Currently only some theoretical work has been done in this area with encrypted functions. Some work has also been done with obfuscated code that is very hard to read for the human eye, but the main problem of blackbox execution is still unsolved.

- *Tracing:* The agent can keep a trace of which actions or operations it has done on that host. Usually this information is hashed or encrypted. The original host can then examine the agent when it returns and verify that it has done the correct operations. If the original host detects tampering, it can figure out which host in the network is malicious and avoid sending agents to it.

There are also a few other problems in the mobile agent area that are not related to security. While the size of the agent is supposed to be relatively small if it is going to reduce the network overhead, the size of the data it carries can sometimes grow very large if it visits a lot of hosts and collects information at each place. Sometime it can be better to use the traditional client-server model or even let the agent send the data back to its originating host.

This leads to the problem of sending messages to agents. If the agent is jumping quickly from host to host, how can the message packet catch the agent and deliver its message to the agent? One solution to this problem is to send out several messages in a large wave across the network, and the agent cannot escape. Unfortunately, this creates a lot of overhead in the network load.

Objects in a distributed system need distinguishing names. They also need to be fairly short; otherwise it would create too much overhead. Agents can be created at a very high rate and sent out all over the network. This can result in some naming problems even if there are central servers that keep track of their names. The obvious solution is to give each agency one name that it prepends to each agent, but the problem here is when agents are cloned at other hosts or new ones created at virtual agencies on foreign hosts.

5.5 Distributed object systems

An object is a separate program component with a set of data and methods (functions). They are often made in an object oriented programming language. Distributed objects are objects that can reside on any computer in a network, and any other object can use a method in this object from any other place in the network. To be able to do this a special system is needed, which keeps track of where the objects are in the network and which interface they have. This system also handles synchronising, error handling, security and naming of new objects. There is often also a "glue" for cooperating with older software that is not object-oriented.

Examples of distributed object systems are CORBA and DCOM. These systems are not made for sending mobile code, as the objects are not moving in the system. However, the distributed systems can act as a backbone for an agent system, giving them an infrastructure, the means for communication and naming. For example, ActiveX controls are already based on DCOM.

6 Conclusions

This report has shown that mobile agents have three properties: mobility, executability, and autonomy. It has also tried to show that these three properties are quite powerful as a new programming paradigm, but also that the current state of the mobile agent field is perhaps not on the right track when it is searching for a good application.

Many projects on mobile agents have failed. The reason is not certain, but it can be because of a lack of good applications, too many security problems or a lack of trust in this new technology. It looks like the mobile agent technology is at a crossroads and is choosing between going even more "engineered" and deterministic or going into the more "biological" and chaotic path. One observation is that agents are not as much a technology that is a paradigm, a way of programming like object orientation, or even at the same abstract level as "programming language". It is probably a mistake to make agent systems an entirely separate technology or method for solutions in a distributed environment like the Internet.

One major obstacle on the way is the security issues. The agent host must be protected from the agents, and the agents must be protected from each other and the host. There are some viable solutions for host protection, but there is still a need of research into the protection of agents.

7 The future

Research in mobile agents is quite young and there is considerable research in progress in this area. Many problems with security and distributed systems need to be solved.

7.1 Research topics

The current research topics can be divided into the three areas of applications, techniques, and protection. In the area of applications there is work being done in mobile code for remote control and autonomous sensor networks. There is also a possibility to use mobile agents as a form of advanced IT-weapons.

New techniques for agent communication, agency architecture, topologies, naming, and integration with other object technologies are also needed.

The protection of the agent is an important problem. In section 5.4, several protection techniques have been outlined and in many cases need more research. The most interesting and perhaps the most difficult problem is black box execution. Other topics within agent security are listed below.

- *Flexible access control:* A better system for access control is needed [17]. The agent could carry its credentials with it and be given different access depending on how trusted it is. Many existing agent systems have too simple access control.

- *Recursive evaluation:* This phenomenon occurs when an interpreter starts running an interpreter inside the program being executed. Thereby it is possible to execute data that was not intended for execution. This may result in unexpected security holes.

- *Protection against viruses and superworms:* Although the agent systems can be used as intrusion detection, the security techniques from the agent technology can be used for protection against viruses and superworms in the operating system. A superworm is basically an agent that does not need an agency to work, and instead uses faults in the operating system for execution and mobility.

7.2 Predictions

Lastly in this report, there is room for more speculative predictions about the future. Some ideas about an ant model have been described in section 5.3.3 above. Many other analogies can be made with similar phenomenons in nature, like swarm theories where the agents acts like a large swarm, or the agents try to simulate the human brain like a neural network. Will the global network like the Internet become a global brain [5]? Other ideas, like evolutionary programming and genetic programming, also have their roots in nature.

Distributed programming might become the new paradigm. Mobile code will be an important part of this paradigm. New primitives for distributed programming will be needed. Other work in this area is the more formal pi-calculus [20], which is a process algebra where processes, or perhaps even agents, can interact by sending communication links to each other.

As mentioned at the end of section 2.1, the future probably lies in autonomy and the distributed autonomous network.

Definitions

ACL	Access Control List. A list of people/objects which have access to a file or other entity.
ActiveX	A technology with protocols and interfaces for downloading distributed objects. Based on DCOM.
API	Application Programming Interface. Description of how to interact with a program.
C++	Object-oriented programming language based on C.
CORBA	Common Object Request Broker. A system for letting distributed objects communicate with each other.
DCOM	Distributed Common Object Model. Similar to CORBA but developed by Microsoft.
FIPA	Foundation of Intelligent Physical Agents.
FTP	File Transfer Protocol.
Java	Object-oriented programming language similar to C++. Has a strong type system and is often interpreted.
LaTeX	A typesetting language.
MASIF	Mobile Agent System Interoperability Facility.
OOP	Object Oriented Programming.
PGP	Pretty Good Privacy. A non-hierarchal public/private key system.
Perl	Originally a text processing and system administration programming language. Interpreted.
Python	Interpreted programming language with good object orientation. Dynamic.
SNMP	Simple Network Management Protocol. Used for controlling networks.
Tcl	Simple interpreted programming language.
TCP/IP	Transmission Control Protocol/Internet Protocol.
Telescript	A programming language for mobile agents developed by General Magic. No longer exists.
WWW	World Wide Web.
UDP	User Datagram Protocol. Has no packet synchronization.

References

[1] Mats Persson, *Mobile Code and Safe Execution*, User Report, FOA-R–98-00807-503–SE, April 1998

[2] Alfonso Fuggetta, Gian Pietro Picco, Giovanni Vigna. *Understanding Code Mobility*, IEEE Transactions of Software Engineering, Vol 24, No 5, May 1998

[3] Gian Pietro Picco, *Tutorial: Understanding Code Mobility*, Tutorial handouts on conference "Agent Systems and Applications/Mobile Agents" (ASA/MA '99)

[4] Jan Vitek, Christian D. Jensen (Ed), *Secure Internet Programming*, ISBN 3-540-66130-1, Springer Verlag, 1999

[5] Michael Brooks, *Global Brain*, New Scientist, 24 June 2000, pp 22-27

[6] Detlef Schoder and Torsten Eymann, *The Real Challenges of Mobile Agents*, Communications of the ACM, June 2000, Vol 43. No. 6, pp 111-112

[7] Lars Hagen, Markus Breugst, Thomas Magedanz, *Impacts of Mobile Agent Technology on Mobile Communication System Evolution*, IEEE Personal Communications, Aug 1998, pp 56-69

[8] Peter M. Maurer, *Components: What If They Gave a Revolution and Nobody Cares?*, IEEE Computer, June 2000, pp 28-34

[9] Alper Caglayan, Colin Harrison, *Agent Sourcebook*, ISBN-0-471-15327-3, Wiley 1997

[10] Danny B. Lange, Mitsuru Oshima, *Seven Good Reasons for Mobile Agents*, Communications of the ACM, March 1999, Vol 42 No 3, pp 88-89

[11] Tommy Thorn, *Programming Languages for Mobile Code*, ACM Computing Surveys, September 1997

[12] Markus Breugst, Thomas Magedanz, *Mobile Agents - Enabling Technology for Active Intelligent Network Implementation*, IEEE Network Magazine, May/June 1998, pp 53-60

[13] Vu Anh Pham, Ahmed Karmouch, *Mobile Software Agents: An Overview*, IEEE Communications Magazine, July 1998, pp 26-37

[14] M. Ranganathan, Virgnie Schaal, Virginie Galtier, Doug Montgomery, *Mobile Streams: A Middleware for Reconfigurable Distributed Scripting*, ISBN 0-7695-0340-3, Proceedings ASA/MA 99, pp. 162-175

[15] Wayne A Jensen, *Countermeasures for Mobile Agent Security*, Computer Communications, Special Issue on Advances in Research and Application of Network Security, Elsevier Science BV, Summer 2000

[16] Stan Franklin, Art Graesser, *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents*, Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996

[17] Trent Jaeger, Aviel D. Rubin, Atul Prakash, *Building Systems that Flexibly Control Downloaded Executable Content*, Proceedings of Sixth USENIX Security Symposium 1996

[18] B. Brewington, R. Gray, K. Moizumi, D. Kotz, G Cybenko, D. Rus, *Mobile Agents in distributed information retrieval*, In Matthias Klusch, editor, Intelligent Information Agents, Springer-Verlag, 1999

[19] Mark Crosbie, Gene Spafford, *Active Defence of a Computer System using Autonomous Agents*, Technical Report No. 95-008, COAST, Purdue University

[20] Joachim Parrow, *An introduction to the π-calculus*, chapter in Handbook of Process Algebra (not published yet), Elsevier 2000, `http://www.it.kth.se/~joachim/intro.ps`

[21] Mats Persson, *Mellanskiktsprogram för Mobil Kod, Agentsystemet D'Agents*, FOA internal PM, Reg. nr. 99-1669/L

[22] *Reactive Sensor Networks*, `http://strange.arl.psu.edu/RSN/`

[23] *Logistics Information Access via Cooperating Agents*, `http://www.engr.sc.edu/research/CIT/projects/ALP.html`

[24] *NetScript: A Language and Environment for Programmable Networks*, `http://www.cs.columbia.edu/dcc/asn/`

[25] Foundation of Intelligent Physical Agents, `http://www.fipa.com`

[26] *Mobile Agent System Interoperability Facility*, Object Management Group, `http://www.omg.org`

[27] *Jumping Beans*, Ad Astra Engineering, Sunnyvale CA, `http://www.jumpingbeans.com`

[28] *The Mobile Agent List*, `http://www.informatik.uni-stuttgart.de/ /ipvr/vs/projekte/mole/mal/mal.html`