Security Requirements for Mobile Agents in Electronic Markets

Michael Zapf, Helge Müller, Kurt Geihs

Department of Computer Science
Johann Wolfgang Goethe-Universität Frankfurt/Main
P.O. Box 111932, D-60054 Frankfurt, Germany
Tel: +49-69-798-22363, Fax: +49-69-798-22643
zapf@vsb.informatik.uni-frankfurt.de

Abstract. In this article we identify security threats and requirements for software agent systems in the context of an electronic market. A short description of our own agent system *AMETAS* is given. It provides an infrastructure for a general multi-purpose agent system. We explain which security facilities need to be employed and how some of them were implemented in *AMETAS*.

Keywords: Software agents, mobility, security, electronic market

1 Introduction

Mobile agents extend the capabilities of distributed systems by code mobility. Mobile agents are programs that can wander through a computer network and can contact other agents and agent places to perform their task. An agent place is a kind of "docking station" where mobile agents can perform activities or obtain some service. Viewed from a computational perspective, an agent is an object that contains data which may partially be acquired during its computing activities.

The mobile agent paradigm offers a number of advantages: Mobility and autonomy make permanent connections unnecessary. Thus, agents are well suited for coping with expensive network links or links that may be temporarily unavailable (e.g. for mobile computing). The usage of agents will also be appropriate in scenarios where large volumes of data would have to be shipped over the network while the processing code itself is rather small. In such a case it is worthwhile to consider moving the code to the data. Some network management tasks fall into this category. Another application arena for mobile agents is electronic commerce and electronic information services. Agents can roam the electronic market place in order to search for desired information or to conduct trading activities.

Among the drawbacks of the agent approach is the increased complexity of the security threats. In order to avoid that a mobile agent acts like a virus or a worm there has to be a security infrastructure that provides protection to agents

as well as to agent places. This is particularly important in an open, dynamically changing computing environment. Sending an agent through a potentially unsecure network expresses some form of trust into the network and the agent infrastructure. Especially for electronic market scenarios where money is involved in the service transactions, the security issue is of highest priority and will at the end determine the success of the agent paradigm in commercial transactions.

In this paper we will evaluate the security requirements of mobile agent systems with particular emphasis on agents in electronic markets. In Section 2 we will briefly sketch the required agent terminology and present an overview of our *AMETAS* agent platform. Section 3 analyses specific security threats for agent systems that go beyond the general security issues of distributed systems. Section 4 discusses security measures and methods for coping with these threats. After describing implications to the agent usage in the electronic market in section 5, we conclude our findings.

2 The Agent System *AMETAS*

AMETAS [1] (short for *Asynchronous Message Transfer Agent System*) is an agent system that was developed in our research group. It is completely implemented in Java 1.1, allowing it to run on different architectures. We decided not to use any already existing system because of two main reasons:

- The agent system should serve as a testbed for various agent research topics, requiring maximum flexibility.
- The agent system should support the notion of agenthood that we believe being most accurate.

Numerous research groups have developed their own ways of defining *agenthood* [2] so that there is neither a unique nor a concise definition suitable for proving agenthood formally. The strongest differences appear when comparing definitions from the operating systems research and artificial intelligence research. While the former is concentrated on mechanisms of the communication or of the agent infrastructure (esp. *mobility*), the latter is more concerned about the interaction with the user and the data processing (using an *internal world representation*).

The smallest common divisor that can be found between all different kinds of agenthood is *autonomy*. Agents from the artificial intellence research have to decide on their own how to react on different stimuli, simulating human decision capability. Mobile agents need their own flow of control when they leave the "home" place and wander to a remote place on some other network node.

Starting from the object-oriented view, we define agents as objects that act in a *permanent client role*. As a client, agents can only initiate requests; they are not able to accept and process them like servers in the conventional client-server paradigm. This characterisation implies the autonomy that was just mentioned as a fundamental agent property.

In *AMETAS*, places provide an execution environment for agents by providing standard interfaces and objects that offer methods which can be locally invoked by the agent. These interfaces and objects are part of the basic functionality of every place and are expected to be available. Places can send or receive agents, and agents that have arrived are instantiated and invoked. From this moment on, the agent processes its definition of behavior (represented by Java code) as an individual entity. Each agent gets its separate flow of control by a newly created thread.

Besides agents, *AMETAS* defines objects called *services*. Unlike agents, services are always stationary; they may be considered as an extension of the basic functionality which is common to all places. Each place may offer a different set of services that may provide data gained from the agent system (e.g. which agents are present) or from the underlying system which is inaccessible for agents. In *AMETAS*, agents that are created or re-invoked after a transfer from another place (*migration*) only get a reference to one special object called agent driver which delegates requests to the places with its services. Any request that an agent directs towards the place or a service is at first sent to the *driver* object that is associated with this agent. The driver accesses the mailbox system and deposits the request in the mailbox of the recipient. Every agent and service can be assigned a mailbox; furthermore, group communication is possible via address masks which entail the creation of group mailboxes.

After the agent called the `depositMessage` method on its driver, the communication procedure is over for the agent, and it is free to leave the place. The driver object of the requested service gets the message, forwards it to the service and sends back a reply message if necessary. The reply is stored until the agent retrieves it from its mailbox or until it times out. This ensures a maximum decoupling of the communication peers and preserves the autonomy of the agent. Figure 1 illustrates the message exchange mechanism.

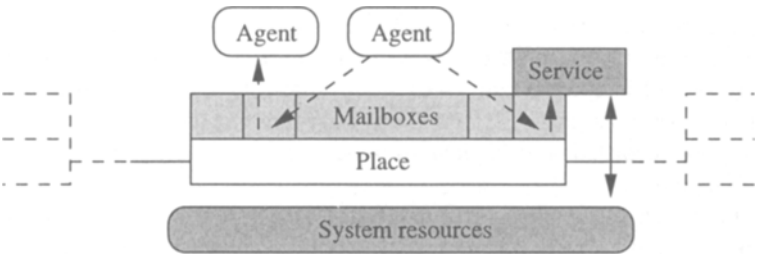

Fig. 1. Agent/system interaction in *AMETAS*

AMETAS makes no assumptions on the actual agent behavior but only provides an infrastructure to support the autonomy of agents. Thus, mobility as well

as intelligence are considered to be additional attributes of autonomous agents in the view of the *AMETAS* agent system.

3 Security Threats

Mobile agents may carry sensitive information and may render execution control (at least partially) to some unknown agent place which might manipulate its state in an unintended, if not malicious way. Just like any distributed system, agents are subject to the following, well-known general security threats:

- unauthorized access to information,
- unauthorized manipulation of information,
- unauthorized access to services,
- vandalism (damaging the system without benefit).

In agent systems, these threats appear in various forms and at different locations. In many cases these threats are amplified through the aspects of mobile code and autonomous activities.

3.1 Threats for Agents

Potential attackers may be other components of the distributed agent system like

- unsecure networks,
- unreliable agent places,
- other agents.

The ensurance of security of agents against malicious places [3] is a severe problem which is usually underestimated at first sight — mainly because this is a rather new phenomenon and the opposite direction is a well-known problem in the context of viruses.

The behavior and state of an agent may be exposed or modified by changing the internal code or data. For example, the owner (i.e. user) of an agent that was sent to bargain for some goods and that carries electronic money in some form may be subject to attempts to manipulate its buying decisions in favor of another, malicious user. Likewise, the communication activities of a buying agent may be recorded and replayed, causing severe liability problems for the owner of the agent.

Agents that have docked onto an agent place transfer control to the place. Their code and data will be readable for the place if no encryption is used. However, the place must have access to the code of the agent. If the agent carried data in encrypted form it would need a decryption key to handle the data by itself; but then the place would also have access to this secret key.

If the place (which must have access to the necessary data) is known in advance the data may be encrypted with the public key of this place. Again, the privacy of agent data cannot be maintained as soon as the data is used at that place.

3.2 Threats for Agent Places

Agent places may be threatened by the execution of malicious agents in several ways [4]:

- Agents may acquire unauthorized access to facilities of the place.
- Agents may gain access to internal data of places and other agents at the place.
- Agents may block the execution of other agents by an intensive utilisation of places and services (*denial of service*).

More specifically, agents may choose the following more or less harmful forms of attack [5]:

- *Spamming*: A place will be impaired or damaged by an artifically high resource utilisation.
- *Spoofing*: An agent tries to adopt a false identity and to gain unauthorized access to data.
- *Viruses, worms, trojan horses*: Agents could be the means of transportation and implantation of malicious code for software attacks.
- *Weed*: Agents that do not do any useful work but consume so little resources that they go undetected.
- *Freeloader*: An agent that uses only services or resources that are free of charge.
- *Flying dutchman*: A freeloader agent that will ensure its own survival by trying to avoid termination or by spawning other agents before termination.

The place must protect the resources of the system, like the operating system, file system, memory, and access to other programs. In order to ensure the security of the system the place should be able to identify an agent. The authentication of the agent sender is necessary to use an access control mechanism to protect the resources.

3.3 Security Policies

To ensure the safety for both agents and places a set of rules (a *security policy*) must be applied. The security services of the place, the programming language, and the infrastructure are used to enforce the security policies. The owner of an agent and the administrator of the place may determine the policies in use.

Multiple policies may be used depending on the involved agents, agent code, and sender of the agent. A policy contains rules for restricting or granting access to data and services, controlling the consumption of resources and restricting and granting of agent capabilities. The agent system should be able to evaluate the amount of trust in an agent, depending on the sender of the agent, successful authentication, and integrity checks. An agent which wants to move to another host may specify the quality of the network communication like privacy, integrity, and authentication of the destination agent system.

4 Security Measures

Places must take security measures to control both innocent and malicious agent behavior. For example, an agent may spawn a large amount of copies of itself over the network. For the agent system, the behavior of an agent is generally unknown a priory so that it cannot determine if an agent is a virus. Moreover, trusting an agent implies trusting all places that had write access to it before. In order to ensure the safety for the resources an incoming agent should be authenticated before the instantiation. After the successful authentication and integrity check the authorization of the agent can be performed. When the integrity of an agent is proven, it may be considered trustworthy even if it comes from an untrusted place. The place must have a possibility to reject an agent that intends to immigrate if it fails to prove its trustworthiness.

The following objects can be distinguished in a multi agent system [6]:

- author of the agent code
- sender/owner[1] of the agent
- agent code
- agent instance
- places

4.1 Integrity

As objects, agents consist of code and data. In *AMETAS*, the agent author writes the agent code in Java and compiles it. After that the agent is digitally signed by the author. The author may use the DES algorithm or a combination of MD5 hash function and asymmetric encryption (RSA). This allows every place the agent visits to check the integrity of the agent code. The agent which usually consists of several Java classes may be signed by one or more authors.

The mechanism to employ encryption in *AMETAS* is rather straight-forward, as figure 2 shows. Using the stream-oriented Java communication, it is merely necessary to place a *filter* stream object in between the outgoing socket and the main place object. If security is not desired, the transmission may bypass this filter on both the outgoing and incoming connection ends.

Places may grant rights to the agent authors who must therefore be known to the agent system. The authors are entitled to restrict the rights for the agent to a required minimum set. Consequently the authors can minimize the risk that the agent will do something unexpected in their role. The authors may specify users who are allowed or denied to use the agent. The rights are added to the agent and then integrity-protected with a digital signature.

[1] The distinction between author and sender is sensible because not every participant in a open agent system needs to write his own agent but will rather use agents from some other agent author.

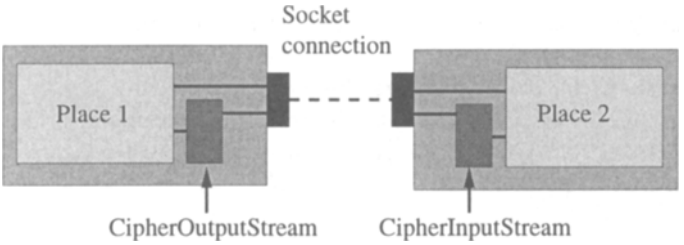

Fig. 2. Embedding of cryptographic extensions in *AMETAS*

4.2 Authentication and Authorisation

We distinguish between the following principals: Human principals like the authors and the sender of an agent are identities who can be identified by their public keys. The places also are identities which own public keys. The keys are certificated by an Certification Authority. This may be a special *AMETAS* place or some other existing CA like VeriSign. The public and private keys of the human participants are stored at their trusted home place. Agents are compound principals and their rights depend on the involved authors and the sender. We assume that agents do not carry private keys because of the missing trust in every place. The identity of an agent is determined by checking the integrity of the code and building a compound principal which depends on the authors and sender of the agent.

Before migrating, an *AMETAS* agent specifies the network security requirements and informs the place that it intends to migrate. The place on which the agent resides connects to the target place and negotiates the migration protocol. After that the following actions take place if full security is desired:

First both agent places authenticate each other by a protocol similar to the SSL protocol (secure socket layer). Places are identifiable by their public keys. The places prove that they own the corresponding private key by encrypting some random data. In order to preserve the privacy and integrity of the agent it is sent in an encrypted form. We use a hybrid encryption algorithm to prevent too much performance loss. After determining the symmetric algorithm which is used to encrypt the migrating agent, a session key is generated and the following migration uses the encryption algorithm[2] with our cipher streams. The destination place checks the integrity of the code and identifies the agent authors and sender. If the destination place does not have the certificates of the involved principals it will ask the sending place for the certificates. If they are available the agent can be instantiated; otherwise it is up to the place to decide whether to grant minimum rights or to completely reject the agent. After the authentication, the place performs the authorisation of the agent. Depending on the trust in the agent authors and sender the place determines the applicable security policy and the access rights to data and services. If the author or the

[2] Currently only the IDEA algorithm is used.

user is unknown to the place, more restricted policies may be used. We distinguish between individual rights and group rights, and rights may be passed on to other users. If the place considers this being less secure it may decrease the level of trust of the respective users. The resulting compound principal is used by our *access control object* (ACO) in order to grant or restrict access to services and system resources.

4.3 Service Access

AMETAS agents do not have direct access to system resources. An agent must use services which are controlled via access control lists. The access right depends on the granted rights, system utilization, priorities, or payment of the agent. In order to protect the system from agents that waste resources, an accounting service and upper bounds on the consumption can be applied. When an agent tries to use a service, the service will contact the ACO determine if the agents owns the permission to use the service. Permissions may be positive or negative and are granted to individuals or groups. A negative permission overrides a positive permission. In *AMETAS*, only a special sort of agent called `UserAdapter` may create a graphical user interface. Every agent belongs to the group `AGENTS`. As such, it is not allowed to create a GUI, even if the initiating user is allowed to run GUI programs. The system resources like input and output channels of a place are controlled by the `AMETASSecurityManager`, which is derived from the `SecurityManager` of Java. This object cooperates with the ACO in order to allow or disallow an agent the use of system resources.

4.4 Generation of Agents

One capability of agents is to create other agents which may be sent across the network to perform their task. It is extremely important to control the reproduction of an agent — otherwise it may flood one place or the whole network in a short time. However, it is not enough to limit the number of agents that an agent can create: To produce an unbounded number of agents it would be sufficient that an agent which has the permit to generate e.g. two copies of itself passes on this permit to each of its descendants. Thus the information about the generation capabilities must be included in the created agent. Alternatively the agent could be forced to pay for the creation of an agent, making a flooding too *expensive* for it. In *AMETAS*, an accounting model will be used: Agents must pay for the creation of an agent. If the created agent wants to create other agents by itself it must receive some money from the creator. In addition, the lifetime of an agent is dependent on the money of the agent. If an agent runs out of money it should die or return to its home place. This model may be used even without real payment systems in order to control misprogrammed agents.

4.5 Security Measures of Agents

One possibility of an agent to ensure its own security is to visit trusted places only. This restriction may be too strong e.g. in an open electronic market because

the agent could not explore all available possibilities. As mentioned before, a place may read all data an agent has access to. If the agent does not trust the place it could make relevant decisions on a neutral host which is trusted by the all agents that are involved.

Before the migration, the agent may specify its security requirements like mutual authentication of the places and encryption to keep privacy. After the place has granted or restricted the requirements the agent decides whether to use the service or not.

An important problem is the integrity and privacy of collected data. A stateless agent which sends all collected data immediatly to its sender is a bad solution because it diminishes the benefits of the agent-oriented approach. The privacy of data can be protected if the itinerary of the agent is known in advance. The data can then be encrypted with the public keys of the places which are intended to have access to the data. However, this strategy neither allows the agent to read the data by itself nor to modify its itinerary.

Another aspect is the integrity of the collected data. The sending agent platform digitally signs the data, and both places confirm the migration of the data. Although it is not possible in general to prevent data from loss or tampering [7], a manipulation is detectable if both places must confirm the sending or receipt of data. Each place could save some information about the migration of an agent (like the agent ID) and a signed confirmation of the agents receipt which would make it possible for a controlling authority to trace agents.

4.6 Replaying

Another threat to an agent is the *replay* of a migration: A malicious place sends the agent a second time to another place. If the agent was intended to perform a purchase, the agent sender will find himself getting the same object twice. On the other hand, it is possible that one agent actually intends to visit the same place several times. To solve this problem, a so-called state appraisal function can be used [6]: The agent carries a digitally signed function with it. This function describes an invariant of the state of the agent, e.g. the number of products to buy and the number of products which are actually bought. If this invariant is not maintained, the sender of the agent may reject the trade.

In *AMETAS*, a place may detect that the same agent visits the place because agents are equipped with a unique identifier. The integration of a state appraisal function is planned for the near future.

5 Electronic Market

Are agents suitable for application in an electronic market? The participants of an electronic market (EM) are buyers and vendors. At first glance one might expect that agents would typically appear as buyers, but a vendor role could be possible as well, e.g. when agents trade goods or services. An agent acts as a delegate on behalf of its user. Therefore it is of major importance to control

the agent behavior in the EM to constrain its possible actions. For instance, an agent must not spend more money than a given maximum amount.

5.1 Payment Systems

In order to discuss the capabilities of agents in an EM we briefly summarize different payment systems. Basically there are three models [8]:

- The *pre-paid* models are comparable to paying in cash. The participant may fetch an amount of money from his bank account and store it on a *SmartCard*-based electronic purse, convert it in electronic money (*E-Cash* from *DigiCash*), or use a check certified by his bank.
- The *pay-now* system differs from this first model by withdrawing the amount of money from the bank account at the time of paying.
- The *pay-later* system is already used with credit cards. The money is withdrawn at some time after the purchase.

From the point of view of the protocol, the latter two models are equivalent because the buyer sends some sort of form (e.g. a check) to the vendor.

The payment may be organized on-line or off-line [9]. The first variant involves a third party (i.e. DigiCash with E-Cash) at the time of paying which is not required with the second one. Payment systems which rely on tamper-resistant hardware are generally off-line (e.g. CAFE). This special hardware can detect and avoid duplicated payment. On-line systems do not need this hardware. At the moment of payment, an on-line broker is engaged that issues *capabilities* for the purchase and performs the actual payment with the bank.

Small amounts of money (less than one dollar) are usually handled by *micropayment* systems. These systems do not require sophisticated security mechanisms.

5.2 Agents and Money

In order to perform a trade, vendors and buyers have to negotiate the procedure of trade, the goods and the payment [10] [11]. This is a good chance to employ autonomous agents. After setting up the contract, both partners sign it using a digital signature which ensures non-repudiation and provability. Furthermore, a third party can independently validate the contract.

For generating the digital signature a private key is required. This is a major problem with mobile agents because the private key must never be lost or compromised. When agents travel to some place, this place is able to inspect the agent behavior and non-encrypted data. Thus if an agent carried a private key, a malicious place could grab it and continue to act as if it were the agent. Without involving the owner of the agent, provability and non-repudiation of trading acts do not seem to be feasible. One possibility could be to inform the agent owner about the ongoing trade (using encrypted e-mail messages) and have him care for the correct procedure. In this case the agent just initiates the

trade. This means, however, to lose important benefits of the agent programming paradigm. Another possibility could be to organize the trade on a neutral host that is trusted by both parties. Finally, the vendor could take a copy of the agent together with its internal state and archive it so that it can prove the behavior of the agent at a later time. However, this would require vast amounts of storage memory.

Micro-payment systems are useful for small amounts of money. They offer two ways of paying: The agent may perform the trade and payment by itself, or the owner of the agent must be involved. In the former case the agent acts on behalf of its owner. Payment is possible because the agent can be authenticated so that the owner can be found. Agents can also make use of payment systems based on credit cards (e.g. CyberCash) with them because they could carry credit card information encrypted with the key of the on-line broker. A drawback of this approach is the lack of a signature.

The employment of electronic cash (like E-Cash) is also feasible; the agent could use some kind of electronic purse. This could also allow it to perform anonymous payments. A possible threat to this kind of agent are malicious places that could rob the agent or terminate it prematurely on purpose or accidentally. If the itinerary of the agent is traceable, those incidents can be discovered. On the other hand, this would break its anonymity because its path could be back-traced to the home place.

Another important aspect is the maximum amount of money that the agent may spend. The agent could be equipped with digitally signed permits to spend a certain amount. Furthermore, the agent has to bring back information about committed money transfers, signed by the respective places. This way the agent owner can prevent the agent to spend too much money because the places are obliged to check the permits.

5.3 Social Control and Consequences

Aside from the implementation of cryptographic facilities some kind of *social control* [12] may enforce the security in an agent system. Participants that are known to be dishonest or cheating could be excluded from the electronic market. For agents, this could be compared to the creditworthiness of a real-life person. If this were to be implemented in an agent system it would require to provide a central facility which could be consulted by the places that are about to sign a buying contract with an agent.

An offence within the EM must certainly entail criminal prosecution. This rises the problem of responsibility of the agent author and agent user: Can an agent user be obliged to verify the correctness of the agent code? This would require in-depth knowledge about the programming of this agent that cannot even be expected from another agent expert, let alone a common user. If the agent user were indeed responsible for all actions of his agent it would be doubtful if agents were ever employed in this application field. On the other hand, if an agent programmer were solely responsible for any action that might be triggered by the employment of his agents, he would find himself half-way to prison from

the moment on when his agents are widely in use. A possible solution could be to limit the liability of the participants. However, this could scare off vendors from the participation in the electronic market if they had to fear to be ruined by a group of misbehaved agents without the prospect of reimbursement.

As it was stated before, places could be a source of danger for agents, too. Unlike everyday shopping where people are directly involved in the purchase acts, agent users do not recognize a cheat fast enough to be able to cancel the act. It will be very likely that among all those places of veritable vendors with highly sophisticated services, there are places in the twilight which could cheat unwary agents. This could happen by forging money amounts in payments or even by modifying the hard-coded behavior of the agent.

It seems that although security of the places against attacks by agents could be ensured, the opposite direction will probably never be satisfactorily solvable. In order to provide a best-possible solution the notion of *reputation* could be introduced. This means that not only agents but also places should be interested in having a well-respected reputation. Analogously, in real life we would avoid those stores where we were cheated once — and we would warn our friends not to go there. In an agent system there could be a facility that informs an agent about the kind of service that it should expect at a remote place. It seems reasonable to believe that the more clients are using their agents to do shopping in the EM, the more vendors will feel obliged to care for a correct handling.

6 Conclusions

In this article we showed that the employment of the novel software agent paradigm does not only offer new ways of participating in an electronic market but also infers new dangers that must be taken care of. We identified different kinds of threats to agent and place security and explained how agent systems can be extended to use security facilities. The agent system *AMETAS* that was designed at our research group has already been equipped with cryptographic support and other security facilities. It should be noted that *AMETAS* is not only intented to investigate security issues for mobile agent systems: The long-term goal of this agent system is to find strategies for defining *type systems* for agents. A type system could assist users in finding the suitable agent for a given task, or provide programmers with basic agents to be specialized.

The considerations above demonstrate that there is still some way to go before mobile agents will carry out of our everyday shopping. There are urgent problems to be solved also in the jurisdictional area to regulate responsibilities and liabilities.

References

1. M. Zapf: Design paradigms in agent-based systems, Proceedings of the working conference on Distributed Applications and Interoperable Systems DAIS'97, Cottbus, Germany; Chapman & Hall 1997

2. M. Genesereth, S. Ketchpel: Software Agents, http://logic.stanford.edu/sharing/papers/agents.ps
3. F. Hohl: An Attempt to Solve the Problem of Malicious Hosts in Mobile Agent Systems; http://www.informatik.uni-stuttgart.de/ipvr/vs/hohlfz/sosp97.ps
4. Crystaliz, General Magic, GMD FOKUS, IBM, The Open Group: Mobile Agent Facility Specification, June 2, 1997
5. D. Chess, IBM Corporation: Things that Go Bump in the Net; http://www.research.ibm.com/massive/bump.html, 1995
6. W. Farmer, Joshua Guttman, Vipin Swarup: Security for Mobile Agents: Authentication and State Appraisal, 1996 Computer Security ESORICS'96, 4th European Symposium on Research in Computer Security
7. D. Chess, B. Groshof *et al.*: Itinerant Agents for Mobile Computing, IBM Research Report, 1995
8. J. Abad Peiro, N. Asokan, M. Waidner: Payment Manager — Overview; SEMPER Activity Paper 96, http://www.semper.org/
9. N. Asokan, Phil Janson *et al.*, IBM Research Division: Electronic Payment Systems; http://www.semper.org/
10. M. Waidner: Development of a Secure Electronic Marketplace for Europe; Computer Security ESORICS'96, LNCS 1146, Springer, Berlin 1996
11. M. Schunter, M. Waidner: Architecture and Design of a Secure Electronic Marketplace, http://www.semper.org/sirene/publ/ScWa_97SEMPER.JENC8.ps.gz
12. L. Rasmusson, S. Jansson: Simulated Social Control for Secure Internet Commerce, 1996; http://www.sics.se/~lra/nsp96/nsp96.html
13. G. Coulouris, J. Dollimore, T. Kindberg: Distributed Systems Concepts and Design Addison-Weslay Publishing Company, 1994