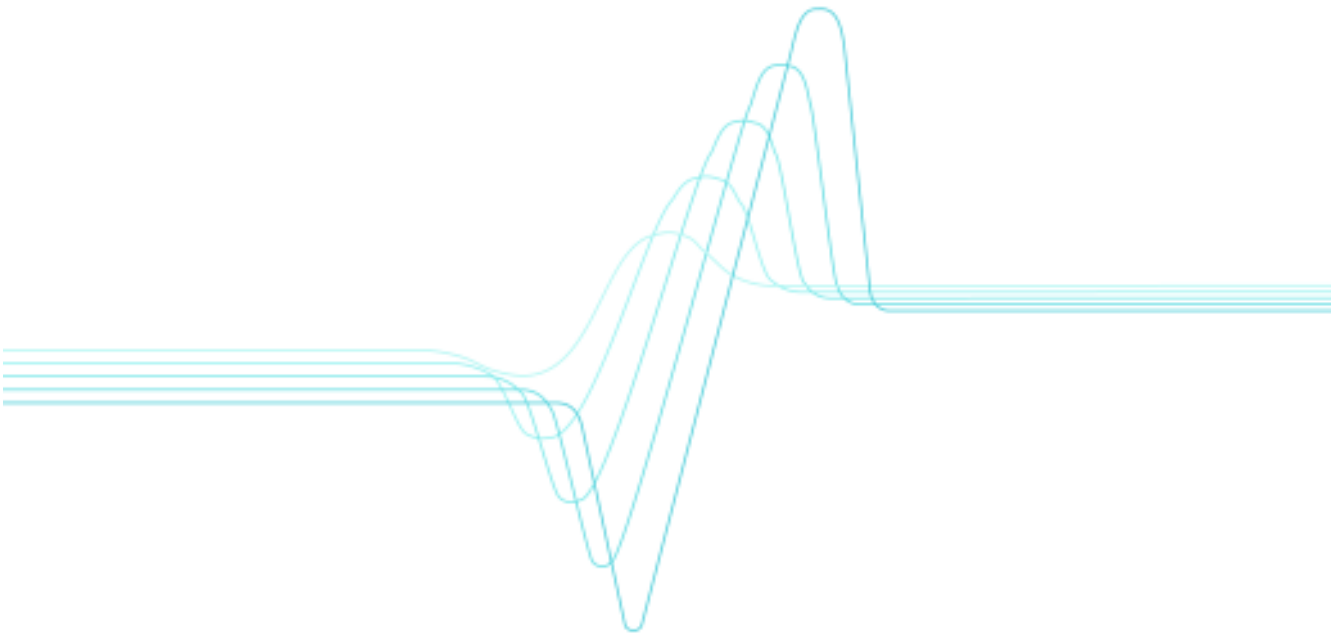
ESPOO 2004 VTT PUBLICATIONS 538

Arto Wallin

Secure auction for mobile agents



VTT PUBLICATIONS 358

Secure auction for mobile agents

Arto Wallin
VTT Electronics

ISBN 951–38–6394–8 (soft back ed.)
ISSN 1235–0621 (soft back ed.)

ISBN 951–38–6395–6 (URL: http://www.vtt.fi/inf/pdf/)
ISSN 1455–0849 (URL: http://www.vtt.fi/inf/pdf/)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 5, PL 2000, 02044 VTT
puh. vaihde (09) 4561, faksi (09) 456 4374

VTT, Bergsmansvägen 5, PB 2000, 02044 VTT
tel. växel (09) 4561, fax (09) 456 4374

VTT Technical Research Centre of Finland, Vuorimiehentie 5, P.O.Box 2000, FIN–02044 VTT, Finland
phone internat. + 358 9 4561, fax + 358 9 456 4374

VTT Elektroniikka, Kaitoväylä 1, PL 1100, 90571 OULU
puh. vaihde (08) 551 2111, faksi (08) 551 2320

VTT Elektronik, Kaitoväylä 1, PB 1100, 90571 ULEÅBORG
tel. växel (08) 551 2111, fax (08) 551 2320

VTT Electronics, Kaitoväylä 1, P.O.Box 1100, FIN–90571 OULU, Finland
phone internat. + 358 8 551 2111, fax + 358 8 551 2320

Technical editing Marja Kettunen

Otamedia Oy, Espoo 2004

Wallin, Arto. Secure auction for mobile agents [Tietoturvallinen huutokauppa liikkuville agenteille]. Espoo 2004. VTT Publications 538. 102 p.

Keywords multi-agent systems, mobile agent systems, security architecture, robustness, security threats, protection

Abstract

In this work, a secure auction place for mobile agents has been developed and implemented. In the implemented auction application, software agents are able to bid on different products independently without user intervention in a secure manner. For implementation, quality requirements for mobile agent systems were studied. By defining the possible threats that a mobile agent system may face, a set of protection mechanisms were selected to build a security architecture that was used in protecting the agent system. The agent platform and the agents were designed based on the security architecture and the implementation was carried out using Java. Finally, in order to provide proof about the robustness of the implemented system, it was tested using a mini-simulation method.

As a result of the implementation, the communication security for transferring a mobile agent between nodes could be provided effectively by using traditional security mechanisms. However, the distribution of keys and certificates is required to be done manually so that the counterparts of the auction events can trust each other, which on the other hand, causes scalability problems. As a difference to most other agent platforms, the type of agent mobility was more restricted and code mobility was not used. For this reason, the security of the agent platform could be provided better. However, restrictions that had to be made to the system were quite big resulting in a decrease in the agent system's ability to adapt dynamically.

In summary, there are still a few problems that have to be overcome before the time is ready for large-scale mobile agent auctions. In particular, the security of the mobile agent residing on a remote platform has to be guaranteed without the assumption of trusted platforms.

Wallin, Arto. Secure auction for mobile agents [Tietoturvallinen huutokauppa liikkuville agenteille]. Espoo 2004. VTT Publications 538. 102 s.

Keywords multi-agent systems, mobile agent systems, security architecture, robustness, security threats, protection

Tiivistelmä

Tässä työssä kehitetään ja toteutetaan huutokauppasovellus liikkuville ohjelmisto-agenteille siten, että tietoturvallisuutta painotetaan tärkeimpänä laatuattribuuttina. Toteutettavassa sovelluksessa agentit, jotka pystyvät toimimaan itsenäisesti ilman ulkopuolista käyttäjän apua, käyvät huutokauppaa keskenään. Toteutusta varten määriteltiin liikkuvien agenttien laatuvaatimukset agenttijärjestelmälle. Lisäksi selvitettiin agenttien kohtaamat mahdolliset uhat, jotta voitiin muodostaa tietoturva-arkkitehtuuri, jonka avulla voitaisiin suojella agentteja toteutetussa agenttijärjestelmässä. Agenttijärjestelmän toteutuksessa kehitettiin sekä kaksi erillistä agenttityyppiä (ostaja ja huutokaupanpitäjä) että agenttialusta niille. Järjestelmä toteutettiin Java-ohjelmointikielellä. Lopuksi järjestelmän kyky sietää virheellistä informaatiota testattiin käyttämällä hyväksi minisimulaatiomenetelmää.

Työstä saatujen tulosten perusteella agentin siirtyminen agenttialustojen välillä pystytään turvaamaan hyvin perinteisiä tietoturvamekanismeja käyttäen. Avainten ja sertifikaattien vaihtaminen osapuolten välillä tuottaa kuitenkin ongelmia. Jotta vaihtaminen voitaisiin tehdä täysin luotettavasti, olisi se tehtävä henkilökohtaisesti. Tämä aiheuttaa puolestaan ongelmia järjestelmän skaalattavuutta ajatellen. Monista muista agenttialustoista poiketen tässä työssä käytettiin liikkuvaa koodia yksinkertaisempaa liikkuvuuden tyyppiä. Tämän ansiosta järjestelmän tietoturva, etenkin agenttialustan osalta, saatiin hieman paremmaksi, mutta käytetty liikkuvuuden tyyppi aiheutti vastaavasti agenttien dynaamisuuden heikkenemistä.

Yhteenvetona voidaan todeta, että liikkuvien agenttien järjestelmään liittyy vielä ongelmia, joita ei ole kyetty ratkaisemaan. Erityisesti etäagenttialustalla asustavan liikkuvan agentin tietoturvallisuutta ei kyetä suojelemaan tarpeeksi hyvin. Näin ollen täysin turvallisen liikkuvien agenttien huutokaupan toteuttaminen ei ole vielä toistaiseksi mahdollista.

Preface

This thesis was completed within the VTT Electronics' Software architectures group. Work for this thesis was carried out under three different projects, all of which are somehow connected to each other.

The development of the agent system started in the PLA (Product Line Architectures) programme during the summer of 2002. One of the project tasks was to develop the architecture for intelligent services and apply it in practice. Generic agent architecture for mobile agents was developed, and to demonstrate its applicability in practice, an example of a primitive auction place was implemented. As a result of the case study, new research possibilities for future multi-agent systems were identified, and the research on it was decided to continue.

The second part of the research started in the beginning of 2003 in the OPERA project, which was a follow-up project to PLA. The project contained a work package, which focused on researching multi-agent systems, especially mobile agent security problems. The objective of the work was to first find out the quality requirements of the multi-agent system, then develop security-based architecture for them and finally, refine the previously implemented auction application to meet the detected requirements. The work completed in this project forms the basis for this thesis, although, lots of important background work had already been carried out in the PLA programme.

The final part, the writing of this thesis, was addressed mainly in the project Minttu and was completed in January 2004.

I would like to thank Research Professor Eila Niemelä from VTT Electronics and Professor Juha Röning from the University of Oulu for guiding me through this work. My second supervisor, Professor Janne Heikkilä, also deserves my appreciation. I would also like to express my gratitude to Mr. Rauli Kaksonen from Codenomicon, who provided valuable information about robustness testing and help with the testing process. In addition, Mika Hongisto who helped me especially at the beginning of the work, and Jarkko Holappa who guided me with the security issues, deserves my appreciation. And last, but not least, I would like to thank Katri Kemppainen for love and support during the entirety of this thesis.

Oulu, 28th January 2004

Arto Wallin

Contents

Abstract 3

Tiivistelmä 4

Preface 5

Abbreviations 9

1. Introduction 10

2. Agent technology 13
   - 2.1 Introduction and terminology 13
   - 2.2 Status of the mobile agent technology 17
   - 2.3 Towards the standardisation of agent technologies 17
   - 2.4 Mobile agent systems 19
     - 2.4.1 Architectural models of mobile agent systems 19
     - 2.4.2 Mobile agent platforms 23
   - 2.5 Quality requirements for multi-agent systems 25
     - 2.5.1 Interoperability 26
     - 2.5.2 Scalability 27
     - 2.5.3 Mobility 28
     - 2.5.4 Security 29
     - 2.5.5 Robustness 31

3. Security technologies for mobile agents 33
   - 3.1 Cryptographic algorithms 33
   - 3.2 Digital signature 35
   - 3.3 Certificates 37
   - 3.4 Java security 38
     - 3.4.1 Class verifier 40
     - 3.4.2 Security management 40

4. Mobile agent security threats and protection 42
   - 4.1 Threats 42
     - 4.1.1 Agent against agent platform 43
     - 4.1.2 Agent against other agents 44

4.1.3 Agent platform against agent 45
4.1.4 Other entities against agents and agent platforms 46
4.2 Protecting the platform 46
4.2.1 Authentication 47
4.2.2 Authorisation 48
4.2.3 Cryptographic service 49
4.2.4 Execution tracing 49
4.2.5 Proof 50
4.3 Protecting the agent 51
4.3.1 Path histories 52
4.3.2 Partial Result Encryption 52
4.3.3 Proof of agent's identity 53
4.3.4 Co-operating agents 54

5. Case study: auction for mobile agents 55
5.1 Description of the agent system 55
5.2 Agent platform 57
5.2.1 Management 58
5.2.2 Registration 60
5.2.3 Security 60
5.2.4 Communication 60
5.3 Agents 61
5.3.1 Shopper agent 62
5.3.2 Auctioneer agent 63
5.4 Concrete architecture 64
5.4.1 Structural view 64
5.4.2 Behavioural view 65
5.4.3 Deployment view 68
5.5 The security framework of the mobile agent system 70
5.5.1 Platform security 71
5.5.2 Agent security 77
5.6 Results of the implementation 79

6. Robustness testing 82
6.1 Mini-simulation method 83
6.2 Mini-simulation toolkit 83
6.3 Test cases 84

6.4 Results of the tests 88

7. Discussion 90
7.1 Realised quality requirements in the case study 91
7.2 Future of mobile agent systems 96

8. Conclusions 98

References 100

Abbreviations

AES	Advanced Encryption Standard
API	Application Program Interface
CA	Certificate Authority
COTS	Commercial Of The Shelf
CORBA	Common Object Request Broker Architecture
DSA	Digital Signature Algorithm
DES	Data Encryption Standard
FIPA	the Foundation for Intelligent Physical Agents
IETF	Internet Engineering Task Force
IP	Internet Protocol
JVM	Java Virtual Machine
MAS	Multi agent system
MASIF	OMG's Mobile Agent System Interoperability Facility (standard)
MD5	Message Digest, version 5 (Message-digest algorithm)
NIST	National Institute of Standards and Technology
OMG	Object Management Group
PDA	Personal Digital Assistant
PLA	Product Line Architectures
QADA	Quality-driven Architecture Design and quality Analysis method
RSA	Rivest-Shamir-Adleman (encryption algorithm)
SHA	Secure Hash Algorithm
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
W3C	World Wide Web Consortium

1. Introduction

Software agents have been a popular topic in the area of information engineering for several years. Lots of expectations have been laid on them, but large-scale usage of agents is still waiting. One of the main reasons for the immaturity of agent technology is that we have not been able to make it secure enough to meet the strict requirements put on it [1].

The initiative for developing an auction for mobile agents was originated from a project that had developed a model for generic agent architecture but was missing an application that would make use of it. A mobile agent system, where agents could place bids by themselves in a virtual auction without user intervention, was decided to be developed and implemented, and the author of this thesis was assigned to the work.

Implementation of the demonstration can be roughly divided into two different tasks. The first task was to develop an agent platform, which embodied different services for agents (registration, communication and management) and services for the user, for example, creating new agents and managing connections. The second task included the implementation of different agents: a shopper and an auctioneer, which are actual participants of the auction event. Agents had to be designed in such a way that they were able to act independently and reliably also in remote nodes, where an agent owner could not help them. As a result of the demonstration, the developed architecture proved useful, however, there also seemed to be fundamental deficiencies in it. Security of the system or weakness of it caused the greatest concern when thought about applying the researched architecture to commercial applications. The problems that emerged from the demonstration form the basis for this thesis and are considered through the following research questions:

- What are the requirements for a secure mobile agent system?

- What security mechanisms are needed to build an extensive security-based architecture that can be applied in an auction application for mobile agents?

- How can a secure auction place for mobile agents be implemented, and what are the difficulties of implementation and possible solutions of these problems?

- How can quality requirements be met in a real life multi-agent system and what are the conflicts between them?

The field of software agents is still arguing about the right definitions. In this work, we consider software agents to be autonomous problem-solving entities that do not need permanent guidance from the user [2]. More accurately, our agent system is a multi-agent system, which consists of both mobile and stationary agents. In this thesis, mobile agents are not based on mobile code; instead they are considered to be serialised Java objects that contain all of the agent's knowledge packed into the knowledge base. When the mobile agent is transferred to a new platform, an agent is recreated according to the knowledge base. Therefore, the data and knowledge according to which an agent acts is personalised, however, an agent's code on the remote platform originates from the platform itself and only provides functionality predetermined by the platform.

In the security consideration of this thesis, we will concentrate on the security of the mobile agents. However, our purpose is not to go too deep into specific security mechanisms, but to come up with a collection of security mechanisms put dynamically into one. This security framework is then applied to the multi-agent auction system to provide the possibility for mobile agents to participate in a secure virtual auction. We also want to receive feedback about our architecture model and its applicability in practice.

The goal of this thesis is to develop a security-based mobile agent application, which is further defined to be a virtual auction place for mobile agents. In order to be able to develop the agent system successfully, we have to first consider the quality requirements of mobile agent systems, in particular from a security point of view. Then we will develop security-based agent system architecture by refining the architecture, which was initially developed for the auction demonstration, with security mechanisms. This architecture has not been designed from a security perspective and is missing essential components, thus components for security and the glue to fit those into the system architecture

have to be developed. Since the security of the agent system is an inseparable combination of platform and agent security, both issues have to be considered. Furthermore, the mobile agent system will be implemented by refining the earlier work of an author with the security mechanisms. In order to provide proof of the security of the developed agent system, the system's robustness is tested with the help of a robustness testing toolkit provided by Codenomicon [3]. Finally, we analyse the results of the implementation. Difficulties in making a secure agent system are presented as well as a discussion on how the implementation succeeded and what could/should have been done in another way.

2. Agent technology

2.1 Introduction and terminology

The idea about agent, autonomous entities able to solve problems by themselves has been fascinating people for a long time. Nowadays, various kinds of agents can be found around us, we just might not realise all of them. Classification of the agents at the main level can be divided into three categories: biological, robotic and computational agents [4]. Our interest focuses on computational agents, which can be further divided into artificial-life agents and software agents, which we will take a closer look at.

Although the definition of a software agent varies largely between different interest groups, there are some common characteristics that fit with most definitions of software agents. At minimum, a software agent is defined to be an autonomous software entity that is able to interact with its environment. Thus, it is able to react independently without user intervention to stimulus from outside, for example, from humans, platforms or other agents [5]. A few other properties of software agents are listed in Table 1 [4, 5, 6].

Table 1. Agent properties.

Property	**Explanation**
Adaptive	Being able to learn and improve with experience.
Character/ Personality	An agent has personality and emotional state.
Co-operative/ Collaborative	An agent can act together with other agents for a common purpose.
Intelligence	Ability to sense its environment, reflect upon it, and take actions to achieve a goal. State is formalised by knowledge.
Proactive	Agent's actions are goal-orientated. It is able to choose if certain action will bring it closer to its goal and acts accordingly.
Reactive	Responds in a timely fashion to changes in its environment.
Social	Can communicate with other agents or people.
Temporally continuous	Is a continuously running process.

One of the most interesting additional properties of software agents from our point of view is mobility, which means that the agent is able to transport itself from one execution environment to another. This moving mechanism is also referred to as migration. A mobile agent is a specific form of software agent, which is a self-contained and identifiable computer program that can move within a network and act on behalf of the user or another entity [7].

Most mobile agent systems base their agent mobility on code mobility, where both the data and the code of the agent are transferred during migration. This kind of agent mobility can be classified as either strong or weak. The difference between these is that strong mobility allows the code and entire execution state of the agent to be transferred, while weak mobility only makes the code to transfer. Strong code mobility is more complex to achieve, it negatively affects performance and introduces more security risks than weaker mobility, however, it allows an agent to restart execution from the exact point where it was before migration. Weak code mobility also has some unsolved weaknesses in security when applying it to mobile agent systems [8].

Because there are still deficiencies in the security of code mobility, which cannot be fully prevented, we do not use it in this thesis. Instead, we resort to mobile agent technology, which restricts the mobility a little more but correspondingly enhances the security to a new level. Thus, mobile agents in this work are serialised Java objects, which can migrate according to following steps:

1. Execution of the agent is stopped.
2. Knowledge of the agent is packed to the knowledge base.
3. The knowledge base and any possible additional information of a migrating entity are shipped to the destination node.
4. The agent is recreated at the destination node according to information obtained from the knowledge base.
5. Execution of the agent is restarted.

As a result, when the mobile agent migrates between nodes, its "brain" (referring to knowledge base) migrates, but its "body" (referring to the code), which the agent's brain uses in order to be able to perform actions, is not transferred.

Instead, the destination platform has a clone of the body, where the agent's brain is placed and the agent is thus able to act again.

A drawback of the chosen type of mobility is that the dynamics and functionality of the agents are restricted by the platform. Therefore, all agents of a certain type have the same set of operations that they are able to perform and any new operations have to be first updated to the agent platforms. These same things also improve the security of the agent system, since the platform always knows what operations residing agents are able to perform, the set of malicious acts made by agents against the agent platform is narrower, and thus the agent platform is able to trust residing agents better. However, there are many other security-related questions that have still to be answered.

There must also be some reasons that make the use of mobile agent technology worthwhile. One main benefit of a mobile agent system for a traditional system is visualised in Figure 1, where communication between nodes is represented with arrows. Traditionally, a lot of communication is required between nodes in order to handle difficult tasks, such as an auction. However, if mobile agents are used, data needs to be transferred between nodes only when the mobile agent is moved to accomplish its tasks and when it returns home. Thus, mobile agents reduce communication and data transfer between nodes considerably. Other main benefits of mobile agents are asynchronous and autonomous execution, overcoming network latency, robust and fault-tolerant behaviour, operating in heterogeneous environments and adapting dynamically [9].

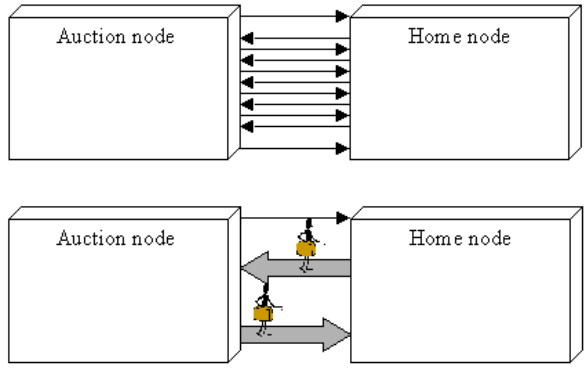


Figure 1. Traditional system vs. agent system.

Multi-agent systems (MAS) are systems composed of agents co-ordinated through their relationships with one another. Agents can collaborate either through co-operation, competition or a combination of both, but they cannot solve problems by themselves. Other special characteristics for MAS are [10]:

- system is open and therefore doesn't have global control,
- data is decentralised, and
- computation is asynchronous.

It would be much easier to implement single agent systems, but there are some rationales that speak on behalf of MAS. One large multi-skilled agent could be able to do the same things as many smaller agents, however, in most cases, the agent would be too slow, hard to maintain and probably not even reliable. If we use smaller agents and divide functionality between them, it will improve system modularity, flexibility, modifiability and extensibility [5]. The security of the system can also be improved by giving security-critical functions to strictly controlled agents and letting other more open agents do less risky functions.

Our auction implementation is a good example of a multi-agent system. There exist different roles (shopper and auctioneer) that have the ability to communicate with each other. Agents have different specialised knowledge, they do different tasks that complement each other, and they can also compete against other agents, thus fitting the definition of MAS.

Another term that is often referred to when talking about an agent system containing multiple agents is *mobile agent system*. The difference in the definition of terms is that a mobile agent system, as the name says, always contains at least one agent that is mobile, but there may or may not be other agents in the system. On the other hand, a multi-agent system does not necessary contain any mobile agents, but instead, it always has more than one agent. Thus, the most descriptive term for the agent systems discussed in this work would be mobile multi-agent system. However, the term used is selected based on which characteristics, mobility or multiple agents, is wanted to be emphasised at that time.

2.2 Status of the mobile agent technology

It is evidently clear that there are some deficiencies in agent technology. First of all, the development of multi-agent systems is still in the hands of top experts in the domain. We are missing efficient tools that would make it possible for non-specialists to unlock the power of agents, which would make it easier to spread mobile agent technology. Secondly, among others, there are also some unsolved problems with interoperability, scalability and security. Most of the developed multi-agent systems are built from scratch for specific purposes in an ad hoc manner, thus they are on quite a small scale, without good reusability possibilities, and unable to co-operate with other multi-agent systems [1]. Development towards large-scale multi-agent systems and connecting them to each other thus forming large interconnected networks of multi-agent systems has begun lately, but it has raised complexity issues that we are still not able to cope with [11]. Security, the issue in which this thesis is concentrating on, has also proved to be very complex. Some have even claimed that mobile agents will never be secure enough to be trustworthy, but this is what we will discuss later on.

Despite of its deficiencies, it can easily be shown that multi-agent systems have already been applied successfully in a wide range of application domains. At the head we have e-commerce, which has for a long time shown potential benefits of agents in theory, and lately proven many agent solutions to be functional also in practice. Another creditable participant in the research of mobile agent technology is the telecommunication industry, which has successfully adapted mobile agents in their applications.

2.3 Towards the standardisation of agent technologies

Today there are numerous different research groups and commercial actors researching agent technologies and implementing various agents and platforms for them. Because of the diversity of the agent technology research, a common standardisation organisation was needed. The Foundation for Intelligent Physical Agents (FIPA) was formed in 1996 to fill this gap and since then it has contributed a lot to agent technology. FIPA is a non-profit organisation that produces software standards for heterogeneous and interacting agents and agent-

based systems. In the production of these standards, FIPA requires input and collaboration from its memberships and from the agent's field in general to build specifications that can be used to achieve interoperability between agent-based systems developed by different companies and organisations. Also these days, FIPA is the main driving force aiming to produce standards for the interoperation of heterogeneous software agents [2].

FIPA's core standardisation activities are mainly handled by Technical Committees, which concentrate on creating and maintaining specifications. There are currently many active Technical Committees including, for example, ontology, interaction protocols, semantics and security, which is particularly interesting to us. The Security Technical Committee's objective is to lead the research and development of multi-agent systems. Its area of responsibility covers the understanding of security requirements and properties, defining FIPA-based abstract security specification and to concretise security specifications to form an agent-based security service for deployment in application domains. The Security Technical Committee's work started in the beginning of 2003 and is planned to be ready at the end of 2004 [2]. The objectives of the FIPA's security work seem to be very close to our research and thus it will be interesting to see how its work proceeds in the future.

Another major standardisation organisation and contributor in the field of mobile agent technology is the Object Management Group (OMG), which has been particularly active in making initiatives and standards for interoperability and security. OMG's core infrastructure for mobile agent technology is specified in MASIF (Mobile Agent System Interoperability Facility) [12] with its main objective to enable interoperability between agents systems. Although MASIF addresses security quite extensively, concentrating on authentication and authorisation, it has a few deficiencies and limitations. For example, MASIF does not address multi-hop authentication, thus agents are only allowed to migrate one-hop from the source system. An agent's itinerary is also restricted to only trusted nodes, which limits autonomous agent activity. MASIF's security solutions conform extensively to CORBA security services. Agent communication is also dependent on CORBA standards, since communication goes through the ORB communication channel [13]. Despite the promising standardisation work OMG has carried out mainly in the late 1990's, a number of its contributions seem to have diminished noticeably in recent years. It remains

to be seen how important a part OMG will have in the future of mobile agent technology.

In addition to FIPA and OMG, there are also a few other standardisation organisations that are working towards initiations and standards for mobile agent technology. IETF (Internet Engineering Task Force) has paid special interest to Internet-agent technologies. Another standard body, W3C has addressed Web languages and ontology for mobile agents. Some other research groups and formal forums that have promoted mobile agent technology are the Agent Society, NIST and the Agent InteropWorking Group [13].

Standardisation is always very important from interoperability and reusability points of view, having thus also an important effect on extended use of any new technology. On the other hand, standardisation is a compromise, and the standardisation process itself can be very slow. Therefore, there is also a possibility that the standardisation process itself prevents the broad use of new technology. In the area of multi-agent systems, the main problem of standardisation is the lack of consistent theory and definitions, although agent communities are working all the time to bridge these problems.

2.4 Mobile agent systems

This section takes a brief look at a few earlier developed architectural models for mobile agents, which have influenced the agent system introduced later in this thesis. In addition, two mobile agent platforms that have many similar aspects to the one presented in this thesis are described in order that the developed agent system can be compared to already existing implementations. When examining these agent systems, special attention is paid to the mobility and security issues provided by the platforms.

2.4.1 Architectural models of mobile agent systems

Many agent standards, such as FIPA and MASIF, define architectural models for mobile agent systems. However, these architectural models are, in most cases, at a quite abstract level, which leaves free hands for agent system designers to

apply and develop concrete architecture for the agent system. A visualisation of the FIPA abstract architecture is presented in Figure 2 and detailed information about it can be found in [2].

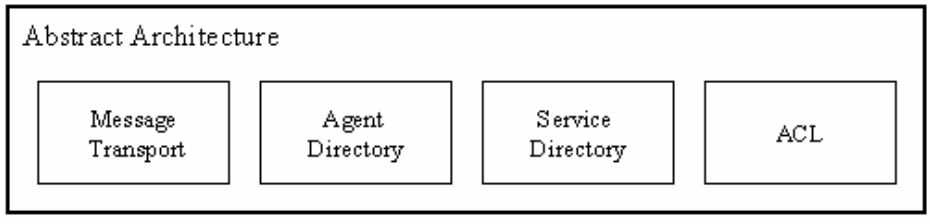


Figure 2. FIPA abstract architecture.

Additionally, these architectural models usually only concern agent platform architecture, thereby ignoring the architecture of mobile agents. However, the architecture of mobile agents and their co-operation is a very important part in the case study presented in this thesis. Therefore, two agent co-operation architecture types and the internal architecture model of a mobile agent are examined briefly in the following.

Centralised vs. distributed agent co-operation

Agent systems have two main co-operation architectures, which distinguish from each other clearly. The first, presented in Figure 3, is centralised co-operation hierarchy. This architecture contains multiple agents, but only one works as a central agent in the server platform, while the other agents are clients. All inter-agent communication, presented in the picture with lines connecting agents, goes through the central agent. Thus, other agents are subordinate to external control by the central agent.

The scheme of agent co-operation architecture where a single agent acts as a global manager might be workable on a small scale, however, the scheme becomes impractical when the number of agents in the system increases. For this reason, centralised architecture is applicable for a limited amount of agents, but should not be used in large-scale distributed systems [6, p. 13].

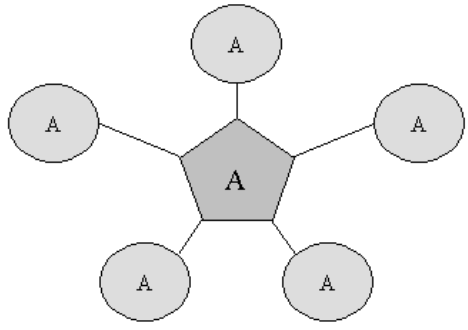


Figure 3. Centralised architecture.

Another approach for the agent co-operation hierarchy is to provide agents with the ability to control their communication by themselves without external control. This requires more advanced intelligent co-operation from the agents, and the number of possible inter-agent communication links is much more numerous. An example of the agent hierarchy model without external control referred to here as distributed architecture is presented in Figure 4 [6, p. 13]. This model is particularly challenging for mobile agents, since under these circumstances they have to be aware of their locations and also know where co-operating agents currently are, although the agent platform also has responsibility for providing the required services.

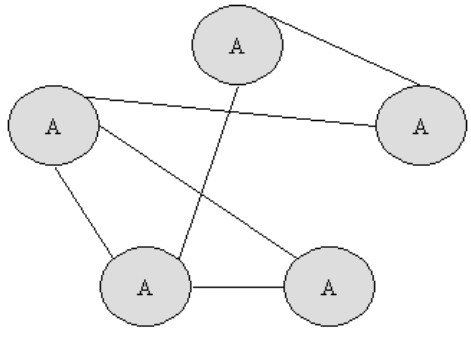


Figure 4. Distributed architecture.

The agent system introduced in this thesis uses co-operation architecture, which is a combination of both architectures presented previously. It can act both ways, but usually consists of a few central agents called auctioneer agents and several

other agents called shoppers that act in certain events under the auctioneer's supervision. In contrast to centralised architecture, there can be many central agents at the same time working on the same problem field, thus reducing burden from the shoulders of other agents. Additionally, non-central agents (shoppers) can also interact with each other without control from the central agent. Thereby, architectural bottlenecks that are faced by centralised architecture can be avoided.

Layered agent architecture

Generic agent architecture introduced in [14] has had greatest influence on the mobile agent architecture further developed in this thesis. It is a conceptual view of an agent, based on vertically layered architectural style. According to it, an agent is composed of three layers: a definition layer, an organisation layer and a co-operation/co-ordination layer. The agent architecture and its three agent layers are depicted in Figure 5. The two other layers are included in the figure for completeness sake; the communication layer provides low-level details for inter-agent communication and the application programmer's interface (API) layer links the agent to its physical resources. Further information about Ndumu's agent architecture, and also complete explanation of the presented three layers, can be found from [14].

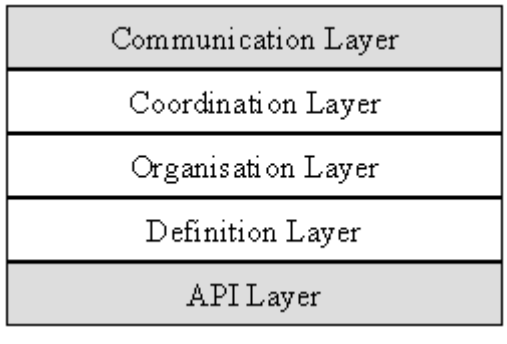


Figure 5. Layered architecture for agents.

2.4.2 Mobile agent platforms

A few years ago, when the success of mobile agent technology was at the peak of its wave, new mobile agent systems and platforms were built by numerous groups around the world. A summary of mobile agent systems and technologies used in building them is represented in [5]. However, now the dust has settled, and it can be seen that many of those implementations have been forgotten and only the best mobile agent systems have survived. Research on agent systems has still proceeded and recently some agent platform research groups and companies have already released second generations of their mobile agent systems, such as Grasshopper 2 and Ajanta, which will be studied in more detail in the following.

Grasshopper 2

As said, Grasshopper 2 is already the second generation of the agent platform developed by the IKV++ Technologies AG. It is based on specifications from FIPA and OMG MASIF and has gained worldwide success since the release of its first version in 1998.

Grasshopper's core is a Java-based mobile agent platform that provides the possibility to develop and run agent applications. The agent environment is built from one region registry and several agencies, which can be compared to agent platforms in our agent system. The agency contains several services, which are security, registration, persistence, management, transport, and communication. The main purposes of these services are similar to our agent platform services, except the persistence service, which is used for saving agent-related data in case of system crashes. In this way, agents can continue their task after the restarting the agency. Agent mobility in the Grasshopper is based on weak code mobility, and thus an agent's state is restarted each time it migrates to a new location [15].

The agency's security service uses many traditional security mechanisms to provide external security: Secure Socket Layer (SSL) protocol protects agents during the migration, certificates are used in authentication purposes, and cryptographic algorithms for encrypting the data packets under transmission.

Internal security of the agent system is strongly dependent on the security mechanisms in Java 2, where access control forms the core of protecting the agency against agents. Surprisingly, none of the agent specific security mechanisms are reported to be used in the protection of the agent system, which raises the question to be discussed: can the security of the agent system be provided just with conventional security mechanisms?

Ajanta - Mobile Agent Research Project

Ajanta is a mobile agent programming system developed at the University of Minnesota. The first evaluation version of Ajanta was made publicly available in 1999 and few years later in 2003, Ajanta's second commercial version was released. The reason for its introduction here is that it is one of the most interesting ongoing mobile agent research projects related to our work, since its main focus is on the design of secure and robust execution of mobile agents in an open environment [16].

The Ajanta agent system is implemented using Java language and is built from agent platforms called agent servers and mobile agents. The agent server makes the host's resources available to agents in a controlled manner. It also provides some basic primitives to agents, such as communication and migration. Mobile agents are mobile Java objects based on weak code mobility. Mobility of the objects is implemented using Java's serialisation facility, which allows the capture of the objects state, transmitting it to some other agent server and recreation of objects on that server.

Security of the Ajanta is mostly based on Java's security model. It contains many conventional security mechanisms, for example, tamperproof certificates called credentials, cryptographic mechanisms, and access control mechanisms provided by the Java virtual machine. Furthermore, some special mobile agent protection mechanisms are used, such as itinerary recording and mechanism for protecting the agent’s state at the remote hosts [17].

2.5 Quality requirements for multi-agent systems

Quality requirements for a multi-agent system depend strongly on the purpose of the system. Therefore, exact quality requirements for all multi-agent systems cannot be defined. However, general guidelines can be drawn up to help designers choose which quality requirements are particularly important for their applications.

In certain cases, multi-agent systems are required to be adaptive. It means that agents are able to learn and improve with experience [10]. Such agents may be required to adapt to modifications in their environment. They may include changes to the component's communication protocol or possibly the dynamic introduction of a new kind of component previously unknown or the manipulation of existing agents [18].

Another quality requirement that is often associated with multi-agent systems is modularity, which increases the efficiency of task execution, reduces communication overhead and usually enables high flexibility and better reusability. On the other hand, it implies constraints on inter-module communication [18].

There are also many other quality requirements concerning multi-agent systems such as maintainability, portability, flexibility and integrability, which partly overlap with others introduced here. However, our purpose is not to go too deep into the requirements, but we try to concentrate on the essential qualities. Therefore, we take security, which is a very important quality requirement in many multi-agent systems including the one in our case study, and pay special attention to it in the following review. Other requirements, which are also considered to be important from the viewpoint of the case study presented in this thesis, are interoperability, scalability, mobility and robustness. Reasons why these quality requirements are selected and information about those is discussed separately in the following subsections.

2.5.1 Interoperability

FIPA defines interoperability in multi-agent systems to be the ability of agents from different agent platforms to be able to communicate with each other. In other words, agents should be able to interact and share information, knowledge and tasks to achieve their goals [10]. This is one of the most essential quality requirements involved, since without it, an agent system could not even be considered as a multi-agent system.

Interoperability between agents can be achieved with help of three key elements [10]:

- common agent communication language and protocol,
- common format of context of communication, and
- shared ontology.

In our case study, the well-defined structure of messages plays an important part in interoperability. It enables agents to understand messages from other agents and thus forms the basis for co-operation between agents. Also, communication protocols such as an auction protocol, plays a key role when considering interoperability issues.

As we noticed earlier, a multi-agent system consists of various agents that are not able to reach their goals by themselves but need to work together and co-operate in order to achieve their shared objectives. However, agents inside the MAS do not always work together towards the same goals. Sometimes agents' goals may be mutually exclusive, and thereby, if one agent achieves its goal, another no longer can. These agents are competitive. However, this is not in conflict with interoperability between agents, because although agents are competing against each other, they may work towards the common goal of the whole system.

In addition to interoperability between agents, there also exists interoperability between multi-agent systems. Standardisation, and thus developing common interfaces and communication methods, is a common technique to attain interoperability between agents and agent systems. A well-known example is the

effort of FIPA to provide a framework for interoperability between different agent systems.

2.5.2 Scalability

Generally, scalability refers to how well the capacity of a system to do useful work increases as the size of the system increases [19]. In multi-agent systems this is a great challenge, because the size can vary largely from tight interaction between two agents to a community of thousands of agents working together. In spite of this, systems should be able to operate both on a small- and a large-scale without problems.

Scalability can be divided into two different views - macro and micro. A micro view considers issues from the viewpoint of a single agent in the society: How this particular agent is affected when the size of the overall system or the number of agents increases. If it is more difficult or slower for the agent to get services or access to computational resources, then scalability is poor. Also, the need to interact with more agents when the size of the system increases puts stress on an agent and can cause degradation of its performance.

From the macro point of view, the main question is how the system reacts when the number of agents increases. A system has several options: It can increase its performance thanks to new agents' computational capacity, it may be unaffected, or if the scalability is poor, system performance will be degraded. In the worst case, even a system failure may occur, which can cause the whole system to shut down.

When we look at scalability from the viewpoint of our auction implementation, its importance can clearly be seen. Possible scalability problems do not come up in the development phase, when just a few agents are involved in the system. However, when considering the real purpose of the use of the application, where numerous agents and agent platforms would be connected to the agent system, the importance of scalability cannot be underestimated. Without good scalability, functionality of the auction application can be questioned since its real advantages come up only when it is used by a large number of agents from different owners.

2.5.3 Mobility

The main reason why agents should be mobile is the improved performance that is achieved by moving agents closer to the services available on the new host [5]. However, mobility is not suitable for all agents and thus it should be carefully considered, which transactions are performed by mobile agents. As a rule of thumb, it can be said that when an agent's transaction gets more sensitive, correspondingly mobility should be decreased [9]. It is also good practice to divide responsibilities in a multi-agent system so that stationary agents handle security-critical tasks such as money transactions, and smaller mobile agents are assigned to specific tasks that are not so sensitive, for example, information gathering.

The requirement for the multi-agent system to support agent mobility sets also additional requirements on the system. Agent management has to be considered carefully and many questions related to it arises [5]:

- Does mobile agents need possibility to be controlled remotely from the home platform and how can it be done?
- What if a network failure occurs and the agent dies? If the user does not get information about this, he waits unnecessarily for the mobile agent to return.
- Mobility also introduces a new complexity of security issues that are not faced if agents are stationary.
- And finally, what distinguishes an unreasonably roaming mobile agent from a virus?

Wireless computing creates a challenging environment for agent mobility. Firstly, wireless transmission of data is still much more limited because of the lack of bandwidth. That gives an advantage to mobile agents, because they reduce communication over the network, but data transmission still has to be optimised. This can be achieved with the help of packing data and agents for the migration. Secondly, connection to the services should be independent of the user’s location. This can be implemented to some degree, but nevertheless, the wireless connection can get cut of in some situations. Thus, the systems ability to reconnect should be as good as possible.

2.5.4 Security

Security requirements for agents and mobile agent frameworks are mainly similar to those in traditional networked computer systems. These requirements can be divided into four main categories, which are confidentiality, integrity, accountability and availability [9]. These main security requirements are reviewed next, one by one, and their special characteristics are described in detail.

Confidentiality

A general definition for confidentiality can be stated as follows: it ensures that the information in a computer system and transmitted information are accessible for reading only by authorised parties [20, p. 5]. This definition seems to fit quite well with multi-agent systems, where a lot of different communication that should be kept confidential takes place. A platform sends messages and controls information to agents, agents reply to the platform and agents send messages between each other. Every time data is transferred across the network without the proper security action, it is vulnerable to malicious principle eavesdropping. Even if transferred data is encrypted and its content thus cannot be revealed, an eavesdropper can gather information based on the message flow between communication participants. Consequently, any private data transferred between principals of the multi-agent system must be kept confidential.

When a mobile agent migrates to a remote platform, the requirement for confidentiality is put to the real test. Private data inside the mobile agent should be kept confidential even if the agent platform turns out to be malicious. However, providing confidentiality for mobile agents inside the malicious platform is extremely difficult to attain.

The agent platform also contains private data that must be carefully protected and remain confidential. Such sensitive information is, for example, found in audit logs, which are records for storing agent activities on the platform. Key stores and especially private keys must also be protected carefully, because there would be serious security problems if some malicious principle would get access to keys and thus to all data that is encrypted with those keys.

Another issue about confidentiality is that mobile agents should be able to keep their identity and location unknown to other agents, except in some special cases where collaboration between agents is needed. Platforms, however, should be able to identify agents in order that agents can be held responsible for the actions they perform and services they use on the platform.

Integrity

Data integrity refers to data that cannot be manipulated by unauthorised parties without being detected [21]. In multi-agent systems, data integrity is threatened in many ways. Inter-platform messages and migrating agents must be protected from attempts at tampering and unauthorised modification. The agent itself cannot be effectively protected from malicious hosts attempting to alter the agent, however, measures can be taken to detect this data alteration [9]. Also, the agent platform's integrity has to be protected from unauthorised principals. In the case of mobile agents, which use code mobility, the integrity of the platform must be handled with special care because of the possibility that a malicious agent may try to attack an agent platform.

Integrity of the multi-agent system can be protected using replicated agents or multiple agents capable to perform similar tasks. Hence, the failure of one agent does not necessarily crash the whole system; instead other agents can perform actions on behalf of the failed agent [18].

Accountability

Accountability is the ability of a system to keep track of who or what accessed and/or made changes to the system [22]. In the MAS, it means that agents and human users can be held responsible for actions they have performed. That is why every agent and human user has to be uniquely identified, authenticated and all of their security-relevant actions have to be recorded on an audit log. Audit logs also have to be protected carefully and they can be used when the platform is trying to recover from the failure.

The authentication mechanism is essential in order that the platform can decide who is the owner of the remote agent and thus responsible for its actions. Despite the fact that a platform has to be able to identify the actors in the system,

agents should be able to keep their identity unknown to other agents. However, anonymity between agents has to be broken if, for example, a commercial transaction is about to take place between them. If a malicious principal can cheat in authentication so that it is thought to be someone else, it does not only harm the platform, but also the person who it is falsely claiming to be. That is why accountability has an important part in building trust in multi-agent systems.

Availability

Availability is an assurance that the communication or data reaches its intended recipient in a timely fashion [21]. In the MAS, an agent platform should be able to ensure availability for both local and remote agents to data and services. It should also be able to handle all service requests within a reasonable time. If requests cannot be handled, either an intentional or unintentional denial-of-service situation occurs. In the case of an intentional denial-of-service attack, someone is purposely exploiting platforms computational resources or services in order that the platform cannot handle real service requests. When the event that the platform cannot handle the computation or communication load happens, it should be able to provide graceful degradation of services, notify remote agents residing in it that it can no longer provide services, and give the chance for remote agents to migrate back to their home platforms.

Agent platforms should be able to detect and recover from software and hardware failures. Agents, however, can usually be assumed to handle their fault-recovery on their own [9]. This ability to recover from faults is sometimes considered as a separate quality attribute, survivability, but here it is handled under availability.

2.5.5 Robustness

Robustness is the ability of software to withstand exceptional input and stressful environment conditions. A piece of software that is not robust, fails when facing such circumstances. Lack of robustness also offers the possibility for a malicious intruder to take advantage on the situation, for example, through a denial-of-service attack [3].

Although robustness weaknesses are caused by programming mistakes, those are not usually detected during programming. Robustness problems do not either manifest themselves during normal operations but they become visible only when something exception happens, such as someone injects corrupted data inside the software [3].

Even if the mobile agent system could be made theoretically secure with the help of cryptographic algorithms and various other security mechanisms, implementation vulnerabilities can weaken the security of the agent system and make it vulnerable. Implementation vulnerabilities are defined as security hazards resulting from programming mistakes [23]. Unfortunately, implementation vulnerabilities cannot be completely avoided. Since, despite careful programming, it is not economically reasonable to search for every vulnerability in the software. At some point, it is more profitable to release the product and take care of any possible consequences in other ways.

There are three different ways of handling software faults proactively and thus enhance the robustness of the software [23]:

- Vulnerability avoidance: Developing software using techniques that prevent the introduction of vulnerabilities into it.
- Vulnerability elimination: Searching for the vulnerabilities from software using testing or other activities and removing problems.
- Vulnerability tolerance: Building tolerance to the (potential) vulnerabilities in software and ensuring that acceptable results are produced despite of them.

In this thesis, vulnerability elimination is used to point out programming errors from the implemented mobile agent system. Robustness testing is addressed in Chapter 6. Although vulnerability avoidance and vulnerability tolerance are not used as special techniques in this work, the implemented system is programmed carefully trying to avoid programming mistakes. Because all software faults and vulnerabilities cannot be pointed out, even with a special robustness testing method, also run-time management of programming mistakes has to be considered. However, in this study the discussion is limited to vulnerability management during the development of the agent system.

3. Security technologies for mobile agents

Security technologies that are used in mobile agent systems are in many ways closely related to or even the same as standard technologies used in distributed computer systems. This chapter gives a short introduction to what these common technologies that are used as basic building blocks for providing security for mobile agent systems are.

In the first section, we take a look at the few cryptographic algorithms, which are used in our case study to protect agents during their migration. The second section concentrates on introducing digital signatures based on public key infrastructure, which are used in the case study to provide authenticity of the agent or the message. Certificates are presented in the next section in order to provide background information about the authorisation mechanisms used in the auction implementation. In the last section, Java security architecture is considered from the viewpoint of the case study, where a few of the most essential features are examined in detail.

3.1 Cryptographic algorithms

A mobile agent system has to provide communication security for agent migration and the messages that are sent between agent platforms. One way to attain the required security is to use cryptographic algorithms and digital signatures to provide confidentiality and integrity of exchanged messages. This section introduces basic cryptographic algorithms that are used in the case study of this thesis and are thus essential to understand.

Symmetric algorithms, also called secret-key algorithms, are algorithms, where the same key can be used for both the encryption and decryption of data. If the key is revealed, security is lost and can no longer be used. Therefore, secret keys have to be kept secret from unauthorised parties. There are two categories of symmetric algorithms: stream algorithms, which operate on plain text a single bit at a time, and block algorithms, which encrypt larger blocks of data. Maybe the most important advantage of using symmetric algorithms is their efficiency. However, there are also several problems related to them that have to be taken into consideration. Firstly, key distribution must be kept secret, otherwise keys

are revealed and security is compromised. Secondly, every communication channel needs a specific key and the task of maintaining a large number of shared secret keys may turn out to be quite burdensome [24, pp. 211–212].

The most well known symmetrical algorithm is the *Data Encryption Algorithm* (DES). It encrypts data in 64-bit clear text blocks, with a key size of 56 bits. The algorithm itself is quite old since it has become a standard in the 1970's. Despite its age, it is still widely used in the world. The major weakness of DES is in its key size, because an exhaustive search through 56-bit key space can be done in weeks or moths, even with common equipment. Because of the limitations of DES, Advanced Encryption Standard (AES) has been introduced to replace it. AES specifies an algorithm that supports larger block and key sizes. Another option to extend the key size is *triple* DES, which uses three 56-bit DES-keys instead of one. It has also proved to be practical and is widely accepted [24, pp. 212–213]

The auction implementation presented in Chapter 5 supports both DES and triple DES encryption algorithms. The default key size of the triple DES algorithm used is 112 bits, which is achieved with two intermediate keys, but a key size of 168 bits can also be used. In addition, AES could be used with minor changes, but both the DES and at least the triple DES should provide adequate protection for agent migration.

Asymmetric algorithms (also called public key algorithms) use different keys for encryption and decryption. Encryption can be made public using a public key that can be openly distributed, but decryption is achieved with a private key that must remain secret. The public key and the private key compose a key pair, which are algorithmically related to each other. However, deriving the private key from its corresponding public key should not be feasible. Thus, anyone can encrypt data with the public key of the intended receiver, but only the receiver (assuming that no one else has its private key) can decrypt the data. Also, digital signatures use public-private key pairs, but with those, messages are encrypted with a private key and decrypted with a public key [24, pp. 211–212].

The RSA algorithm is a well-known public key algorithm and is also used in many mobile agent systems, including our case study. It is named after its developers, who are Rivest, Shamir and Adleman. RSA is suitable for both the

encryption of the data and digital signatures. The mathematical base of RSA encryption will not be discussed in this paper, however, the description of the algorithm can be found from [20, pp. 173–182]. The RSA algorithm has two major weaknesses: encryption with RSA is quite slow compared to symmetrical algorithms and it is not sensible to use RSA in large data blocks, because encrypted data would have to be divided into blocks that are smaller than the common part of the RSA key pair [20, pp. 173–178].

3.2 Digital signature

The objective of using digital signatures in mobile agent systems is to ensure the authenticity of inter-platform messages and migrated agents. The message sender adds a digital signature to the message, whereupon the receiver is able to verify that the received message came from the claimed source and it has not been altered en route. Digital signatures also support non-repudiation, since only the owner of the private key can be the sender of the message signed with it. Non-repudiation can be, however, invalidated if the sender's private key is exposed or stolen by some malicious entity that then uses it to send messages with a false identity.

The digital signature scheme consists of creating the digital signature and verification of it. There are two ways of creating digital signatures; they may be formed by encrypting the entire message with the sender's private key, or as depicted in Figure 6 by encrypting a hash-value of the message with the sender's private key. A hashing algorithm is used for calculating fixed-size hash-value sometimes also called a message digest. Some well-known hash algorithms are, for example, MD5 and SHA (Secure Hashing Algorithm) from which the SHA is considered to be the more secure against cryptanalysis and brute-force attacks. After calculating the hash-value, the digital signing algorithm is used to encrypt the hash-value and the digital signature is formed. The RSA algorithm can be equally used for signing as for encryption. However, on the contrary to encryption, digital signature algorithms do not need to be reversible. Another generally used signing algorithm is the Digital Signature Algorithm (DSA) [20, pp. 299–301].

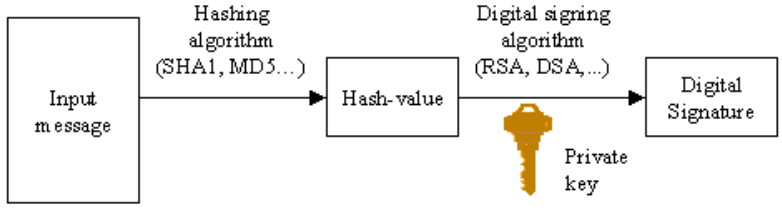


Figure 6. Creation of digital signature.

Verification of the digital signature calculated from the hash-value is a two-stage process. The original hash-value is decrypted from the digital signature using the receiver's public key. In addition, the new hash-value is calculated from the signed message using the same hashing algorithm as the original one. Then these two hash-values are compared to each other and, if they are equal, the message is authenticated and its integrity is proved. Otherwise, a message has been changed during transmission and neither its sender's identity nor message content can be trusted. Figure 7 visualises the verification process of digital signatures [20, pp. 299–301].

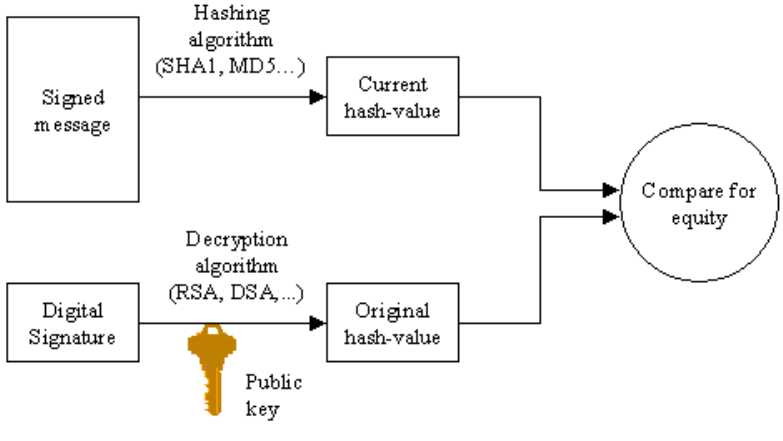


Figure 7. Verification of digital signature.

3.3 Certificates

A public-key certificate is a digitally signed statement from one entity that serves to validate the subject's authority and name. Certificates are usually signed by the Certificate Authority (CA). The CA acts as a Trusted Third Party, who issues and manages public keys and thus guarantees the link between the user and the cryptographic key used [24, p. 219]. The CA is assumed to create only valid and reliable certificates. Thereby, entities that are applying for the certificate have to be able to authenticate themselves to CA. Currently there exists many public Certificate Authorities, such as VeriSign [25] and Entrust [26], who offer chargeable certification services for organisations and individuals.

In mobile agent systems, certificates may have various purposes of use. For example, in the Grasshopper 2 agent platform introduced in 2.4.2, certificates are used for authentication purposes. Our agent implementation also uses certificates in the authentication of agents and the agent platform. In addition, certificates are used by agent platforms for storing the public keys of trusted communication parties.

The X.509 Standard defines the information from which the certificate is built upon. In addition to the signature field, all X.509 certificate versions include the following fields [27]:

1. *Version:* Identifies which version of the X.509 standard applies to the certificate. Currently there are three different versions defined.
2. *Serial Number:* The entity that has created the certificate is responsible for issuing a unique serial number to distinguish it from other certificates it issues.
3. *Signature algorithm identifier:* This field identifies the algorithm that is used by the CA to sign the certificate.
4. *Issuer name:* X.500 name of the CA that has created and signed the certificate.
5. *Period of validity:* This field includes the starting and ending dates and time when the certificate is valid.

6. *Subject name:* The name of the entity whose public key this certificate identifies. All of the names use the X.500 standard and are intended to be unique.
7. *Subject's public key information:* The public key of the entity being named and all necessary algorithm identifiers.
8. *Signature:* Contains the hash code of the other fields encrypted with the CA's private key. Algorithm identifiers and parameters are also included.

The fields describe all of the data that is found from the X.509 certificate Version 1, which is the most generic certificate format. Newer versions include additional fields: Version 2 also has issuer and subject unique identifier fields, which are used to uniquely identify the entities. Version 3 includes a field for extensions, whereby anyone can define an extension and include it in the certificate, for example, to limit key usage to particular purposes.

3.4 Java security

For several years, Java has been the most used programming language in the field of developing mobile agent systems. Reasons for that may be various, but the advanced security model that Java provides, is not certainly the smallest of those.

Java is not just a programming language, rather a complete software platform including application program interface (API) and Java Virtual Machine (JVM). JVM has many built-in security mechanisms that enable the creation of a secure mobile agent system. From a robustness and security point of view, the following properties can be considered as important [28, pp. 59–61]:

- type-safe reference casting,
- structured memory access (no pointer arithmetic),
- automatic garbage collection,
- array bounds checking, and
- checking references for null.

The reason why Java security is considered in this thesis is that the case example, presented in Chapter 5, is implemented using Java. Therefore, being aware of Java's security mechanisms is required. In particular, when proceeding to robustness testing in Chapter 6, knowledge of Java's built-in security mechanisms is needed for the successful test design.

The Java security architecture, presented in [29], consists of several built-in security mechanisms. These mechanisms play an important part in developing secure mobile agent systems and the most essential of them from the viewpoint of agent systems are listed as follows:

- sandbox model,
- secure class loading,
- class verifier,
- access control mechanisms, and
- security management.

Most mobile agent systems whose agent migration is based on mobile code make use of these mechanisms. The sandbox model is used since it provides an agent platform possibility to categorise remote code into individual domains. A secure class loader co-operates with the sandbox model and its function is to import binary data that defines the running program's classes and interfaces into the JVM. Access control mechanisms, on its behalf, can be used in defining access rights for mobile code. In spite of the importance of these mechanisms to some mobile agent systems, these are not essential parts of the agent system presented in this thesis, and therefore, they are not reviewed in detail. However, two of these security mechanisms are needed for our case study in order to make it as secure as possible. Class verifying and security management are discussed separately to give a brief introduction to these important features. In addition, two security-related tools needed for the case study (PolicyTool and keytool) are also presented shortly.

3.4.1 Class verifier

The main purposes of the class verifier is to check that a particular class file conforms to the Java language specification and that there are no violations of the Java language rules or name space restrictions. The class verifier also confirms that common memory management violations, such as stack overflows or illegal data type casts, do not occur [30, p. 5]. In mobile agent systems, the importance of the class verifier comes up when considering the robustness of the agent platform, since the verifier recognises and rejects malformed mobile agents and, as a result, the exception will be thrown. The class file verifier catches problems caused by buggy compilers, malicious crackers or innocent binary incompatibility. Thus, the verifier improves the security of Java greatly compared to other programming languages where, for example, stack overflows may present serious security risks.

The class verifier shows its importance to the agent system implemented in this thesis when moving to the testing phase. Testing includes inserting invalid elements to the mobile agents, which are then tried to inject back into the agent platform. Also, some invalid elements that break the rules of Java will be inserted, which might cause serious misbehaviour. However, because of the existence of the class verifier, invalid elements are recognised and exception is thrown before a malformed agent is allowed to enter to the platform and damage is done.

3.4.2 Security management

The purpose of the security manager is to control access to external resources like files or network connections. This is realised in a way that the Java API supports the security policy by asking the security manager for permission before any possibly unsafe actions will be taken. Asking for permission is carried out by invoking check methods on the security manager object. For example, when a ServerSocket receives a connection request, the checkAccept() method in the security manager is called to check if it is allowed to open a new socket to the specified host address and port number. The security policy that is followed by the security manager is defined in the policy file. The policy file, on its behalf, can be configured by the PolicyTool, which is a graphical user

interface that assists a user in specifying, generating, editing, exporting, or importing a security policy. The tool can be run from the command line [29].

PolicyTool can be used in our agent system implementation to restrict allowed connections between agent platforms. This form of security management is particularly useful if the agent platform is repeatedly disturbed by, for example, malicious agents from a particular address. In this case, all connection requests from that specific address can be denied and platform resources will be saved.

Another important security-related tool provided by Java is the keytool, which is a key and certificate management utility. With the keytool, users can administrate their own public/private key pairs and associated certificates. Also, certificates referring to trusted communication parties can be managed by the keytool. User's own keys, certificates and certificates from trusted parties are all stored in the same key store from where they can be referenced by an "alias". This key store is protected with a password from unauthorised users. In addition, private keys are also protected with individual passwords to guarantee that they do not unintentionally fall into the wrong hands. The keytool can be run either from the command line or users can create their own security applications using the KeyStore class provided by the java.security package. With the keytool users can display, import and export X.509 certificates. It also allows users to specify any key pair generation and signature algorithms supplied by any of the registered cryptographic service providers [29].

In the case study presented in this thesis, the keytool is used to generate public/private key pairs and certificates from the command line. Management of the key store, instead, is handled by the key management service implemented particularly for the case study. (More information about implemented management service is presented in subsection 5.5.1.)

4. Mobile agent security threats and protection

Although security plays a very important role in developing mobile agent systems, many of them are developed without a deeper knowledge of the security, leaving it open to be taken care of in the future. However, in order for mobile agent technology to make a breakthrough in the area of commercial applications and gain widespread use, security issues need to be first addressed properly [21].

Security and openness are often said to be opposites in a sense that you cannot have both of them at the same time. There seems to be a dilemma about how the same agent can be both secure and mobile. The answer is simple, mobile agents do not have to be perfectly mobile and they are not also meant to be fortresses that can hold massive attacks. They are a combination of both on a suitable scale and the relation of those usually depends on the task that agent is intended for.

This chapter discusses the security issues of mobile agents. It has been divided into two main parts. The first part concerns security threats of the mobile agents, while the second tries to find out ways how to protect the agent system against those.

4.1 Threats

Many threats in mobile agent systems can also be found in traditional distributed network environments. However, introducing mobile agents significantly broadens the opportunities for misuse. Making an agent able to migrate between nodes exposes the agent to danger. A migrated mobile agent residing in a remote platform also raises numerous security issues to be taken care of.

To be able to protect agent platforms and mobile agents, which move between them, we have to know exactly what kind of threats an agent system faces. These threats can be divided into four main categories [31]:

1. agent against agent platform,
2. agent against other agent,

3. agent platform against an agent, and
4. external entities against agents and agent platforms.

Security threats of the mobile agent system are visualised in Figure 8. Each threat is also discussed in detail in the following subsections.

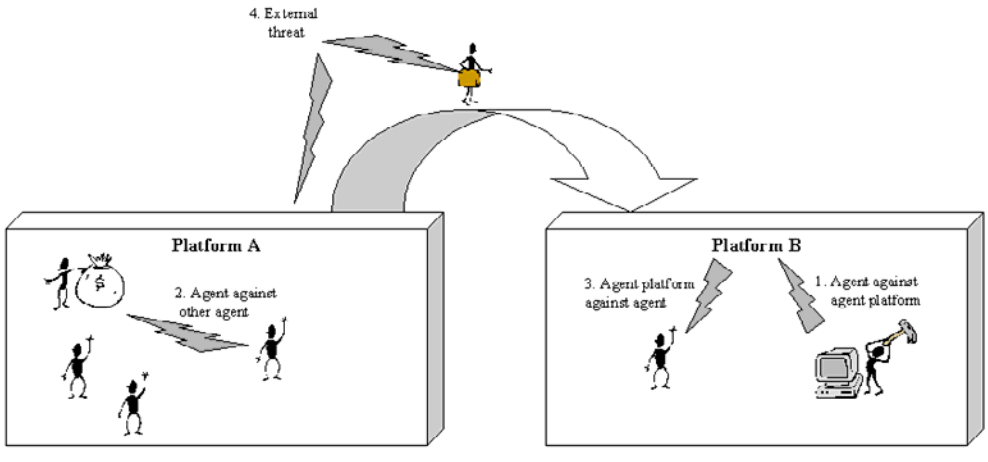


Figure 8. Security threats of the mobile agent system.

4.1.1 Agent against agent platform

The attacking agent has two main ways to inflict harm to the agent platform. An agent can try to gain unauthorised access to the information residing inside the agent platform, or an authorised agent can try to cause harm to the platform. Unauthorised access can be acquired by *masquerading*, which means that an agent pretends to be some other agent that is trusted by the agent platform [31]. When unauthorised access has been gained, an agent has several ways to cause serious harm to the platform, for example, by revealing classified information. It is much harder to protect the platform from an agent that has authorised access to the platform and thus is trusted. However, losing one's trust works as an efficient deterrent for trusted agents to not behave irresponsibly.

An agent can attack the agent platform even without gaining access to it. This can be achieved with a denial-of-service attack, which is used to deny platform services to other agents by exhausting the platform's computational resources

[31]. The denial-of-service attack can be performed, for example, by creating an innumerable amount of malicious agents trying to simultaneously log into the agent platform, with their only intention to exhaust it, and thus preventing benevolent agents accessing the platform.

4.1.2 Agent against other agents

A malicious agent has several approaches to attack other agents. It can take actions to falsify transactions, eavesdrop on conversations, or interfere with an agent's activity [31]. One of the threats is masquerading, where an agent pretends to be an agent other than it really is, causes some kind of harm to the agent system and then leaves the accusations of the community to the real agent. This may cause serial damage to the trust relationships and the agent whose trustworthiness has been hurt has a great deal of work to vindicate itself.

Denial-of-service is another attack that an agent can direct at other agents. It can be achieved in two ways:

1. An agent that provides important services denies those services from a particular agent, thus causing its performance to degrade.
2. Some malicious agent continuously sends service requests to a service provider agent, thus placing undue burden on it.

Success of the denial-of-service attack also depends on the architecture of the agent system. If the agent platform allows direct inter-platform agent-to-agent communication, an agent's denial-of-service attack against other agents would be easier than in the case where all inter-platform messages go through the platform's communication service [9].

Repudiation is the form of an attack in which an agent denies that a legitimate transaction has ever occurred. Repudiation may be caused either deliberately or unintentionally by a misunderstanding between parties. In both cases, it may result in a serious weakening of trust relationships and is a difficult dispute to solve. However, with proper countermeasures, non-repudiation between agents can be rather well assured.

If the agent system uses mobile code in migration and agent platform does not have good control mechanisms, it may result in serious threats between agents inside the platform. An agent may be able to eavesdrop on the conversation of other agents or even worse, it may be able to access other agent's resources without permission or modify agent's code [31].

4.1.3 Agent platform against agent

How to protect an agent from the agent platform is probably the most difficult and discussed problem in the field of mobile agent security. It is usually referred to as a malicious host problem and occurs when the agent has arrived at the remote platform. After that, the home platform looses its control over the agent and little can be done to stop the remote platform from treating the agent as it likes [21]. The remote platform can easily, for example, check the information that the agent is carrying, deny requested services, alter the agent's data or even terminate the agent completely [31].

Masquerading is one possible threat carried out by a malicious platform that falsely pretends to be another trusted platform. It is directed towards mobile agents, which are not yet inside the platform and thus can be deceived about the destination where they will migrate to. Consequently, agents will log themselves into the malicious platform believing that they have arrived at some other platform that is trusted. Once the fake platform has succeeded in deceiving, it has several other possible attacks that may be launched at agents residing in it.

An agent under a remote platform's control is vulnerable to several threats and protection against those threats is very difficult. If the agent consists of data and state information, the platform is able to eavesdrop on all unencrypted data and communication associated with the agent. The life-cycle of the agent is also under control of the remote platform and if the platform wants, it can choose to suspend or even terminate the agent completely. In the case of mobile code there is still another threat, even more serious: The platform may be able to alter the mobile agent's code and thus is able to turn a benevolent agent into malicious one and change the agent's behaviour.

4.1.4 Other entities against agents and agent platforms

Even if agents and agent platforms could be trusted to behave properly, there still exist other entities that may try to disrupt, harm, or subvert the agent system [21]. Possible threats in this category are wide, because it includes many conventional attacks. For example, an external entity may direct its attack at the operating system level under the agent system and thereby try to gain unauthorised access to the agent platform. Also, attacks aimed at communication protocols are possible and quite an effective way to break into the agent system. Because there is such a range of these conventional attacks, discussion of these is beyond the scope of this thesis, and we concentrate on direct attacks against either agents or agent platforms.

Eavesdropping, altering, copying or replaying migrating agents or inter-platform communication are serious threats that can be caused by an external entity that is simply monitoring network traffic, for example, by using some freeware analyser program. Most of these threats can be effectively protected by taking proper countermeasures such as the encryption of a migrating agent or using timestamps and sequence numbers.

Termination of migrating agents and inter-platform messages is also a serious threat that has to be considered. Although it is very hard to protect against termination, some actions can be performed in order that the termination can be detected and proper countermeasures can be taken. These protecting actions are discussed in the following sections in more detail.

4.2 Protecting the platform

The agent platform is vulnerable to attacks by malicious platforms, malicious agents and other malicious entities unless it takes proper action to protect itself. Fortunately, a platform has a wide range of common protection mechanisms traditionally used in communication security or trusted systems that can be used to provide analogous protection mechanisms for it [32].

A number of security mechanisms have been introduced in papers concerning mobile agent security. Some of them are designed for unsafe languages like C as

a software-based fault isolation, which isolates application modules into distinct fault domains [20]. Others, for example, safe code interpretation, are implemented in interpretative programming languages like Java. The idea behind safe code interpretation is that security is enforced through strong type safety, and byte code verification is used to check the safety of the code [32]. Many of these mechanisms concentrate on mobile code problems, and thus they are not essential or directly applicable in our case. However, the best practices of them have been applied in our conceptual architecture of the agent platform security services introduced in Figure 9.

Despite the fact that protection of the agent platform and the agent itself are concerned separately here, the overall security of the agent system is their integration, and thus they should be developed simultaneously.

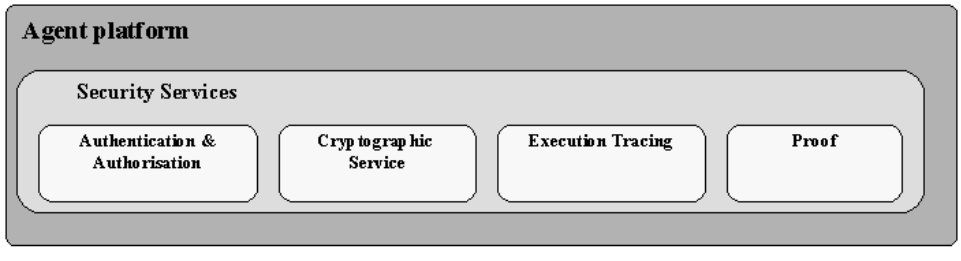


Figure 9. Architecture of the agent platform security services.

4.2.1 Authentication

When a migrating mobile agent arrives at an agent platform, it is an important part of the platform's security to authenticate the incoming agent and thus clarify the agent's identity. The platform can try to associate an agent with the agent's original author, the agent's sender or both [33]. If authentication uses the agent's sender to identify an agent, it must be able to trust it explicitly, thus knowing that the trusted platform would not have sent anything suspicious. This may lead to complex trust chains, where the system trusts someone, that trusts someone else and so on, leading to that the system does not exactly know what it is trusting. For this reason, if misuse occurs, it is hard to track down who is responsible for it. However, if the platform authenticates agents according to

their original authors, it knows exactly whom it is trusting and such confusion will not occur.

Authentication is commonly carried out using digital signatures introduced in Section 3.2. A public key cryptography is usually used for signatures. The signing party computes the hash value over the bits of the agent and encrypts the result with its private key. Then the agent migrates and the receiving platform verifies the signature with the public key and compares it with a locally counted hash value [33]. As a result, if the verification of the digital signature succeeds, the agent platform is assured about the agent's identity and the integrity of the data inside the agent.

Every mobile agent that migrates to the agent platform, including mobile agents that are returning to their home platforms, has to be authenticated. Otherwise access has to be denied or access rights must be restricted to the lowest level, making it impossible for the agent to maliciously cause harm to the platform.

4.2.2 Authorisation

Once we know who is responsible for the mobile agent, we can assign rights to it. Agent platforms usually have different levels of access rights, which are defined in the security policy. The common formalisation of a security policy is an access control list that associates principles with their rights [33]. Decisions on the level of access rights and thus defining which resources and services the agent is allowed to use, are carried out according to the trust that the agent platform has on the visiting agent.

Defining the level of trust is far from simple. If the mobile agent comes directly from a trusted agent platform and the author of the agent is also trusted then there is no problem. However, this is an ideal case but not quite usual. Consideration as to what should be done if things are not that simple has to be taken. For example, a mobile agent may come through an untrusted platform, or the validation date of the certificate corresponding to the digital signature in the agent might have expired. These questions have to be considered in advance, and the decisions to be taken depend on the security policy that the platform wants to impose.

4.2.3 Cryptographic service

A cryptographic service has two purposes: it is used by the agent platform to encrypt and decrypt inter-platform communication, and it provides an encryption service for residing agents. Partial result encryption that uses an encryption service is discussed separately in subsection 4.3.2. This part concentrates on another issue in a cryptographic service.

Because all of the threats that a mobile agent faces during migration, it has to be encrypted before transmission at the sending platform and decrypted after migration at the receiver's side. This encryption is managed by the cryptographic service, which can use, for example, the public key method in encryption and decryption process. The sending party uses the public key of the intended receiver, which is generally available to encrypt data to be sent. Once data has been received, the destination receiver decrypts data with its private key. Because the private key is kept in secret, no one else is able to find out the content of the message even if the sent message gets into the wrong hands.

4.2.4 Execution tracing

Although execution tracing is a mechanism for providing security mainly for mobile agents, its actions are mostly executed by the agent platform, and therefore it is categorised here under platform security. Execution tracing is a technique for detecting unauthorised modification of an agent by sustaining an indisputable log of security relevant actions of agents and platforms [34]. The work is done by the agent platform, which creates the audit log and adds traces of agents residing in the platform to it. Many agent systems submit a trace summary to the mobile agent, when it is about to migrate to a new platform. However, in large-scale mobile agent systems, the size of the log that the agent would be sustaining would probably get too big to retain. Therefore, in the case study presented in this thesis, audit logs are stored by agent platforms.

In addition to benefits of detecting unauthorised modifications and making repudiation more difficult, execution tracing can be used to improve system survivability. Since an audit log contains specific state information of the agent

and the platform, it is easier for the platform to recover from an error with the help of an audit log.

The confidentiality of the audit log is very important for the platform and agent security. Therefore, proper security mechanisms must be taken to retain the confidentiality and integrity of the audit log. This can be achieved, for example, by using cryptographic mechanisms to encrypt an audit log and keeping encryption keys safe from unauthorised entities.

4.2.5 Proof

When a mobile agent is about to migrate to a new agent platform, the agent or the author of it should consider the reliability of the platform. If the platform is not able to assure the agent about its security and benevolence, the agent might decide to cancel its migration to the possibly malicious platform. Thus, to achieve the trust of agents, the platform needs to prove that it possesses the safety properties previously stipulated by agents. Verification of proof can be carried out by sending the proof to the trusted platform where the agent is waiting for migration and the safety properties can be verified [32]. If the verification succeeds the agent migrates, but otherwise, migration is cancelled. Proof should be structured so that verification can be handled without complex cryptographic techniques.

Proof can be provided by the trusted third party that is generally trusted by the agent society. A major drawback is the cost of generating proof and updating it due to the updates to the platform, because proof has to be designed in such a way that attempts of tampering with either the proof or the platform's code will result in a verification error of the proof [32].

Trust of the agents can also be improved by introducing trusted hardware, where the whole agent platform runs on hardware supplied by a trusted third party. In this way, the trusted third party can control security sensitive tasks executed on the platform and assure that the platform acts fairly to all residing agents [21]. However, the high cost of the solution is a major drawback.

4.3 Protecting the agent

Traditionally, program developers have been able to rely on the execution environment not to act maliciously against the executing program. When considering the protection of the mobile agent, the traditional scenario is no longer valid. Mobile agents have to be designed paying attention to the idea that attacks can also arise from the execution environment in this case being the agent platform itself [32].

Security risks cause different problems depending on how far from the home platform an agent is allowed to move. Arising risk is referred to as a single-hop problem when the agent migrates from the home platform to the remote platform, does what it is intended to do and returns directly back to the home platform. If a mobile agent is allowed to visit several platforms before returning home, the risk resulting from this is referred to as a multi-hop problem and it is much harder to mitigate. The multi-hop problem usually means that the mobile agent leaves the trusted network. Problems can be postponed by extending this network of trusted platforms to some extent, but finally the decision between free mobility of agents and scalability of the network has to be made. If the mobile agent is still allowed to enter untrusted platforms, new security mechanisms have to be introduced to protect the agent through its itinerary.

Today, there exist several protection and detection mechanisms introduced to cope with the security problems of mobile agents. Some of them such as obfuscated code are still at a quite theoretical level and there are problems in applying them in practice [35]. Others seem to be quite practical and these are introduced in a security layer for mobile agents as depicted in Figure 10, on purpose to cover diverse aspects of the security of mobile agents. These security mechanisms are discussed in detail in the following subsections.

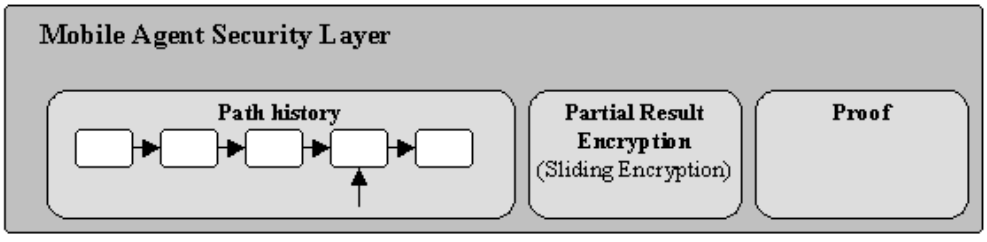


Figure 10. Security layer for mobile agents.

4.3.1 Path histories

The basic idea behind Path Histories [32] is to maintain an authenticatable record of a mobile agent's itinerary. Records consist of path entries, which include the current platform and next platform's identity signed with a digital signature in order that authenticity can be verified. A complete itinerary record is transmitted along with the migrating agent, and each platform adds a new entry to the path in addition to the previous ones. With the help of path histories, each platform can view the agent's itinerary in order to check if the mobile agent has been in untrusted platforms. As a result, an agent platform is able to decide if the mobile agent can be trusted and thus be given access to the platform. At the end of agent itinerary at the trusted platform, the agent's path history can be viewed or every signature of each path entry in the history can be individually authenticated to confirm the identities of visited platforms. If verification of some specific platform identity fails, information gathered from that platform can be neglected or considered with special caution. While the path history does not prevent agent platforms acting maliciously towards agents, it acts as a strong deterrent, since signed path entry is non-repudiatable. Major drawbacks of this mechanism are the growing size of the path history, while a number of visiting platforms increases, and the cost of the verification of those entries [32].

4.3.2 Partial Result Encryption

In a multi-hop situation there exists a severe security problem: how to protect confidential information that one platform has received from other platforms, when the agent is continuing its itinerary. A practical approach to this is to encapsulate the results of the agent's actions at each visited platform.

Encapsulation may be carried out for different purposes with different mechanisms: For example, providing integrity and accountability using digital signatures or providing confidentiality using encryption. There are three alternative ways to carry out the encapsulation of results [32]:

- providing an agent that is able to encapsulate received information,
- relying on the agent platform's cryptographic service for encapsulation, or
- relying on the trusted third party which is capable to encapsulate results of a mobile agent.

All of these ways have their pros and cons and none of these alternatives prevents a malicious platform from terminating the results attained by previously visited platforms. However, eavesdropping and the alteration of information can be effectively prevented.

Usually the amount of information that an agent has gathered is quite small compared to the size of the encryption key and thus the size of resulting cipher text. A solution called sliding encryption [36] has been introduced allowing small amounts of data to be encrypted and added to cryptogram, still yielding efficient sized results. In the encryption of the data, a public key that the mobile agent carries is used. Due to the nature of asymmetric cryptography, a corresponding private key is required to get access to encrypted data. However, even a mobile agent itself cannot access the encrypted payload until it returns to its home platform, which contains the private key for the decryption [21].

4.3.3 Proof of agent's identity

Proof is a security mechanism for providing an assurance of a mobile agent's identity to agent platforms. Thus, this mechanism is quite similar to the proof considered in subsection 4.2.5. Proof can be provided in many ways. One method is to calculate a digital signature from the mobile agent. The agent is then sent with the signature and a certificate including a public key that corresponds to the signature. In consequence, a receiving agent platform can verify the identity of the mobile agent. As a drawback, an agent's signature expires if information in the agent changes. Therefore, either information has to

be stored somewhere outside the agent or the signature must be recalculated every time the agent is changed.

Security risks arising from the mobile code are very complicated to cope with. The main problem from the viewpoint of the agent itself is the risk of a malicious entity altering the code of the mobile agent and thus turning it into a malicious one. One possibility to protect the agent's code against that is a security mechanism called Proof Carrying Code [20]. However, defining accurate protecting mechanisms for agents that use mobile code is again out of the scope of this thesis.

4.3.4 Co-operating agents

One of the main quality requirements of the mobile agent systems was interoperability. One form of it, co-operation between agents, can also be used to protect agents against malicious entities. Information and functionality can be split between two or more agents in such a way that even if malicious entity gets access to resources or gains control of one particular agent, it cannot perform the whole task [21].

Probably the most well known realisation of co-operative agents is to divide functionality between mobile agents and stationary agents in order that they complement each other. A stationary agent is assigned to do most security-critical functions, which would be too risky for mobile agents to carry out. A mobile agent on its behalf is assigned to carry out tasks that can be performed most efficiently on remote platforms and are not too risky.

Co-operative agents can also be used in a variation of Path Histories called Mutual Itinerary Recording [37] where an agent's itinerary is tracked and recorded by another co-operating agent. A mobile agent moving between agent platforms conveys the last platform, current platform and next platform information to the co-operating peer through an authenticated channel. Thereby, malicious actions by the platform against the mobile agent can be detected efficiently and the proper action can be taken if misuse is noticed. Co-operative agents should avoid migration to the same untrusted agent platform, since in that case misbehaviour may not be detected.

5. Case study: auction for mobile agents

5.1 Description of the agent system

The system implemented in this case study is a mobile agent system, which is used as an auction place for software agents. The system is composed of various nodes that can be, for example, personal computers or PDA's. Every node that is connected to the system needs an agent platform running on it to be able to provide services and a place to reside for the agents. The agent platform and agents are implemented with the Java language, thus the Java virtual machine makes the system platform and operation system independent and easily portable. The only additional packet needed by the agent system is the cryptography API provided by BouncyCastle [38]. Agent platforms in different nodes are usually homogenous, though a restricted version may be used on handheld devises because of resource limitations. Nodes are connected to each other via the TCP/IP interface. Visualisation of the simple agent system that is used for the auction is depicted in Figure 11.

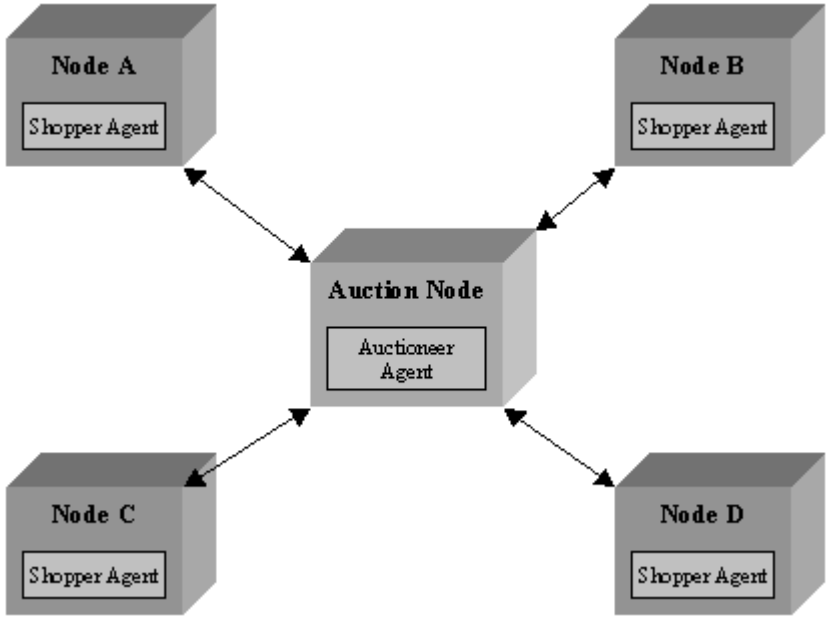


Figure 11. Overall view of the system.

The system includes at least two different types of agents: shoppers and an auctioneer. As seen in Figure 11, the platform in the middle named auction node contains the auctioneer agent while other nodes contain shopper agents. When an auction event begins, shopper agents migrate themselves to the auction node, where the auction takes place. After completion of the auction, shoppers migrate themselves back to their home nodes.

Co-operation of the agents is based on client-server architecture, where the auction node acts as a server as depicted in Figure 11. However, the system's co-operation architecture is actually much more complicated, since any node can act as an auction node. Therefore, it does not have a permanent structure and the co-operation architecture can be even changed at run-time.

Certain decisions have been made to limit the size of the system and to simplify the implementation. At first, the home platform is always considered as trusted and protected. Thus, agents do not face any threats while residing in the home platform. In addition, the remote platform has to be trusted by the home platform in order to allow confidential communication between platforms. For this reason, the platform requires either a safe way to deliver its certificates or valid certificates, provided by the trusted third party, to be made available. However, providing a trusted third party to accept and deliver certificates would be too costly for this work. Therefore, the certificate delivery is handled manually in this case study.

For another thing, the auction node is in direct contact with all other nodes from which agents migrate to the auction place. Thus, nodes do not need to act as a mediator in the message sending process and if the message is not interesting to them it can be trashed without worry. Consequently, implementation mainly uses single-hop agents, which means that mobile agents are usually only one hop away from their home platform. This leads to a higher level of trust in the source of an agent, because it is well known in advance where the agent will visit during its itinerary. Nevertheless, multi-hop agents can also be used and the security implementation has been designed considering issues arising from multiple hops. However, this requires either extra control from the users of the agent platform or the introduction of new functionality to the platforms.

The objectives of this case study are to find out answers to following questions:

1. How can a secure mobile agent system be implemented?
2. What compromises have to be made between agent mobility and the security of the system?
3. What obstacles have to be overcome to apply the developed technology in commercial applications, which will require significant security?

The adaptability of the developed architecture to different implementations is also an interesting issue, but understandably only one agent system has been implemented and references to make a comprehensive summary are too scarce.

5.2 Agent platform

The major function of the agent platform is to provide services for agents living in the agent system. Usually, an agent has to be inside the platform to be able to access platform services. For that, a platform has an abstract place called "place", where agents reside while they are logged into the agent platform. An agent platform also provides some services for the users that support the operation of the agents. The platform has, for example, user interfaces for controlling agents, assigning tasks to them, managing platform security and controlling communication interfaces between nodes.

Visualisation of the agent platform is depicted in Figure 12. Platform services have been divided into four service components: management, registration, security and communication. All of these components cover one aspect very extensively, but interaction between these is also very important. Each service component is discussed in detail in the following subsections.

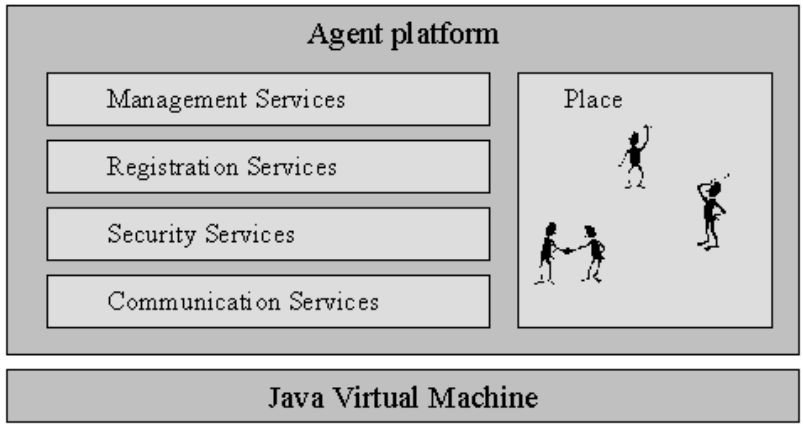


Figure 12. Agent platform.

5.2.1 Management

The management service is the most versatile of the platform's services. It provides, for example, agent life cycle management. The first time the user usually realises it is when a user interface for the agent creation is needed. A user interface consists of an agent pool, where a user is able to select the type of agent that will be created. After the selection, the agent's parameters are entered and with help of a profiler, a new profile for the agent is created. Finally, the new agent is injected into place and a user interface for the agent is created. Now the agent is fully operational and the management service can be used, for example, to force the agent to migrate to a new platform or to reconfigure the agent's parameters. An agent's life-cycle is depicted in Figure 13 [39] and its state transitions are described more specifically in Table 2. An agent's life-cycle is similar to all primary agents with one exception; only mobile agents are able to migrate between nodes, whereas this function is lacking from stationary agents.

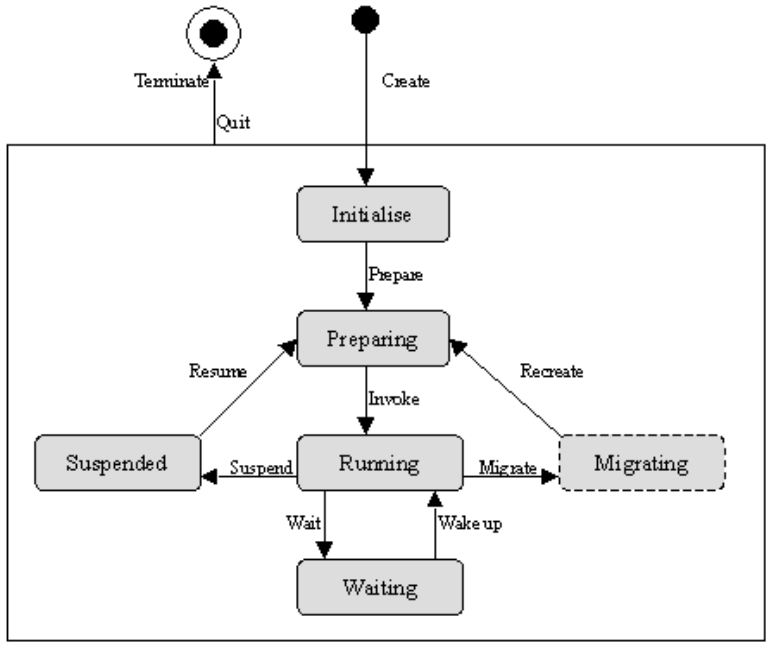


Figure 13. Agent life cycle.

Table 2. Description of the state transitions.

Transition	Description
Create	Creation of a new agent is initiated by the user. Agent's life cycle begins.
Prepare	The agent is prepared for action.
Invoke	The agent is invoked by the management service. It is fully operational and can make its own decisions.
Suspend	Agent execution is halted temporarily by either the management service or the agent itself.
Resume	The agent is resumed from the suspended state.
Wait	The agent puts itself into the waiting state.
Wake up	The agent is woken up by the management service.
Migrate	The agent is serialised and put into migrating state. The decision to migrate can be made by the agent itself or by the management service.
Recreate	The agent is deserialised and recreated after migration.
Quit	The agent itself chooses to end its life cycle.
Terminate	The management service forces the agent to terminate.

In addition to controlling agent's life cycle, the management service also provides an interface for the user to send messages to agents residing on the same platform and to manage connections between nodes.

5.2.2 Registration

The registration service creates the place where agents reside inside the platform. New agents can be added or old ones removed from the place, and information about residing agents is available to authorised users. Agents are able to reside only in one place at a time. Therefore, if an agent migrates to a new place, it will be removed from the earlier one. The registration service is also used to join together agents acting for the same purpose. For example, agents participating in a certain auction are registered to it with the help of the registration service.

5.2.3 Security

Security of the agent system can be divided into security provided by the platform's security services and the agent's security layer. Although the security of the agent system is a tight combination of both of these components, in order to clarify the discussion of different security mechanisms, platform and agent security are considered in this thesis separately.

The platform's security service contains various services that can be used to protect the platform, agents residing in it, and the communication between nodes. These services are key management, authentication, authorisation, encryption and decryption, execution tracing services, and proof. Functionality of the security service is described in detail in Section 5.5.

5.2.4 Communication

The communication service provides a transparent channel for inter-agent communication. Therefore, agents do not have to know the location of their communication partners. Possible communication types are localcast, broadcast

and unicast messaging. *Localcast* messages are sent only to the agents residing in the same platform, thus being the most secure communication type. The two other communication types use TCP/IP and are more vulnerably to malicious action. *Broadcast* messages are sent to every platform and agent that can be contacted. Thus, it is the responsibility of the receiving platform to decide what to do with a received message. If the size of the agent system increases, useless sending of broadcast messages may cause a massive burden on the communication channels. Therefore, using this communication type should always be carefully considered. The third communication type, *unicast* message, is used to send messages to a predefined receiver. Sensitive information that has to be delivered between two agents residing in different nodes is sent using unicast messages.

5.3 Agents

There exist two main agent types, referred to as a shopper agent and an auctioneer agent, which are used in auction events. Both agent types are descendants of the agent super class and have similar functions, for example, the same communication interfaces. Both of the agents also use the same agent language that enables co-operation between them.

Auxiliary agents could be used to support the actions of primary agents. Such an agent could be a banker agent, which could help the user in money transactions. However, secure implementation of the agent that has access to the user's bank account or carries a users credit card number is very difficult, and therefore money transactions should be the responsibility of the user. Another auxiliary agent that is used mostly in generating and maintaining agents' profiles is a profiler agent. However, the profiler agent does not have the same significance as shopper agents or auctioneer agents. In addition, it is not as agent like as the other agents in the system, because it is mainly controlled by the user and thus it doesn't act very autonomously.

5.3.1 Shopper agent

A shopper agent is the most intelligent agent in the system. Its main goal is to buy items, which its author is interested in. When a user creates a shopper agent, he/she is requested to fill up a profile of his/her own interests. An example of a simple profile intending to be used in a furniture auction is depicted in Figure 14.

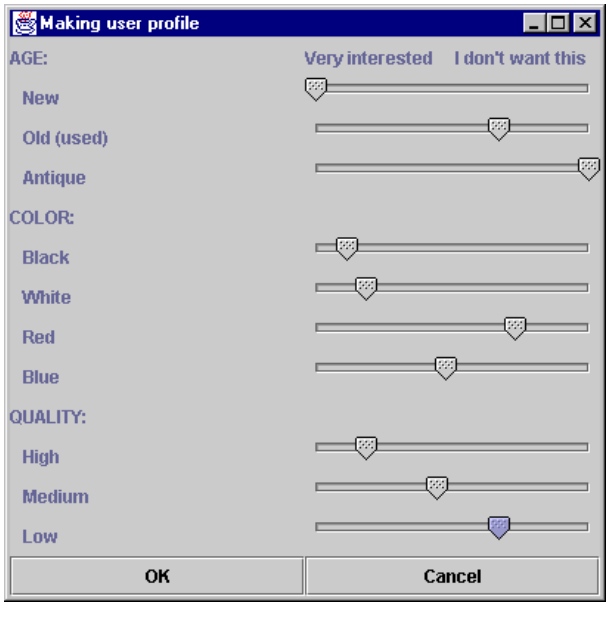


Figure 14. Screenshot of the user profile creation.

After a shopper agent has analysed the profile, it uses the information received to decide if the user is interested in a particular product that has been offered by the auctioneer agent. Thus, the agent can filter out all of the product offers that do not attract its author. Only the most interesting product offers are forwarded to the agent's author, who is asked to confirm his/her interest in the product. An example of a product offer and confirmation request is presented in Figure 15. If the user accepts the request, a shopper agent will migrate to the auction place to take part in the auction. An agent's ability to autonomously make decisions can also be used in the bidding situation, where the agent itself decides, according to the user profile, how high bids it should place. If the author does not feel confident on an agent's ability to decide on the bidding price, he or she can set an absolute limit price that will not be exceeded by the agent.

Figure 15. An example of a product offer.

The shopper agent has certain requirements for its size and security. Since it is a mobile agent, it should be able to easily migrate from one platform to another. Thus, it should not be too heavy even for wireless devices. This sets some restrictions on how extensive the profiles gathered from the user can be. Security issues that the shopper agent must take care of are considered in subsection 5.5.2.

5.3.2 Auctioneer agent

The purpose of the auctioneer agent is to sell products to the shopper agents and try to get as high a price as possible. The auctioneer agent knows the products that it is auctioning and knows their value. It may have a minimum price for products so that if shoppers are not willing to bid above that, it will buy the product back for itself. The auctioneer can receive assignments from clients containing product lists with numerous products on them. Those products are then auctioned one by one from the list and the results of the auction are transmitted to the client. In addition to managing the auction event, the auctioneer sends adverts of products, which are going under the hammer. Adverts are only sent to those shopper agents that are registered with the auction in order to reduce network traffic.

The auctioneer agent is a stationary agent and it does not have such a complex profile as shopper agents have. Therefore, implementation of the agent is simpler

and it is smaller in size. Security threats against the stationary agents are much easier to cope with than in the case of mobile agents. Thus, the agent platform provides the necessary protection mechanisms in order to guarantee the security of the auctioneer agent.

5.4 Concrete architecture

This section describes the concrete architecture of the designed and implemented agent system. The architecture is divided into three views, which are named structural view, behavioural view and deployment view. Each view is a representation of the whole system from the perspective of a related set of concerns. The architecture model specified in this section has been drawn according to the QADA-method [40].

5.4.1 Structural view

The Structural view describes an agent by defining its components and connections between them. The structural view is represented using the composition diagram of the agent, which is depicted in Figure 16.

An agent architecture model that is used in the agent system is based on the layered agent architecture introduced in subsection 2.4.1. According to the presented agent architecture [14], the agent has an internal structure, which consists of co-ordination layer, organisation layer and definition layer. In addition, a new layer is presented in this work, which is referred to as a security layer. It has been added to the agent's architecture in order to enhance the security of agent mobility. Another addition to the agent architecture is the introduction of agent management, which controls these agent layers with control signals. Messages from the platform are first directed at agent management. If the message concerns the agent, it is forwarded to the security layer, which is able to ensure its security. Implementations of the mechanisms in the security layer are described in more detail in Section 5.5.

When the security layer has checked and accepted the received message, it is forwarded to the co-ordination layer, which is responsible for the co-ordination

between agents. In this case study, agent co-ordination is most clearly seen between the auctioneer agent and shopper agents during the auction event, where agents negotiate the price using the auction protocol. Next, data is forwarded to the organisation layer, which can refine received data. An agent's knowledge of its responsibilities in the organisational level is located on this layer.

Finally, data is forwarded to the definition layer, which contains the actual intelligence of an agent. In this layer, the agent makes decisions on how it should act, for example, if a new product offer is received. Decisions to raise the bid or to give up are also originated in the definition layer. When the agent decides to send a message, initiation comes from this layer and in order that the message can be sent, it has to go through layers in reverse order.

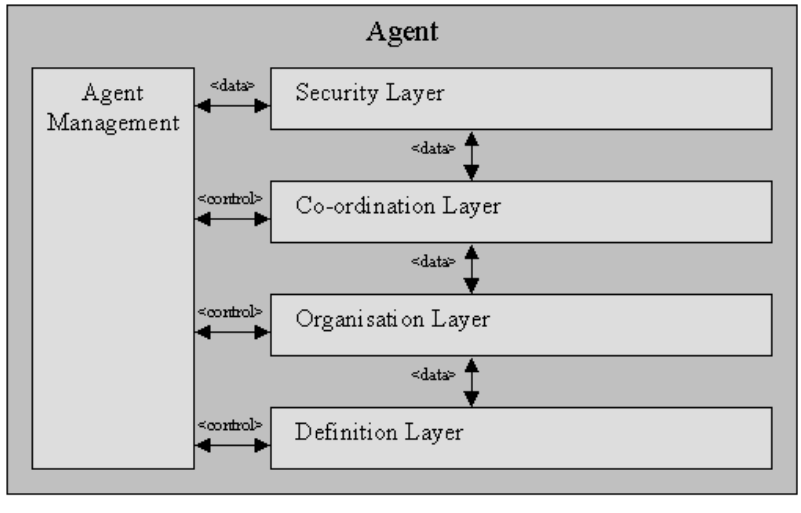


Figure 16. Structural view of the agent.

5.4.2 Behavioural view

The purpose of the behavioural view is to describe the behaviour of a concrete system [40]. This is achieved mainly using sequence diagrams that visualise different events such as auction and agent migration.

Figure 17 describes the migration of a shopper agent from node A to node B. The initiative for migration is usually induced by the product offer received from

the auctioneer agent. Thereafter, the agent requests the platform's management service to carry out migration and the agent's knowledge base is packed into the message. The message is sent to the security service, which takes care of protecting the agent for migration with help of data encryption, digital signatures and time stamps. Then the message is wrapped into an envelope and the communication service serialises the envelope; after that, the serialised data is sent to node B.

On the receiver's side, the communication service receives the data and deserialises it. The received envelope is forwarded to the security service, which decrypts it and verifies both the signature and the time stamp. If there are no detected errors, the knowledge base of the migrated agent is transmitted to the management service that recreates the agent. In order to complete agent recreation, the agent must also be registered in node B, which is carried out by the registration service. When an agent's recreation and registration to the new platform has successfully been completed, an acknowledgement is sent back to node A to notify that migration has succeeded and that the instance of the migrated agent can be removed. If the acknowledgement is not sent, node A considers agent migration to have failed and tries to send the agent again until it receives an acknowledgement or an agent's task becomes obsolete.

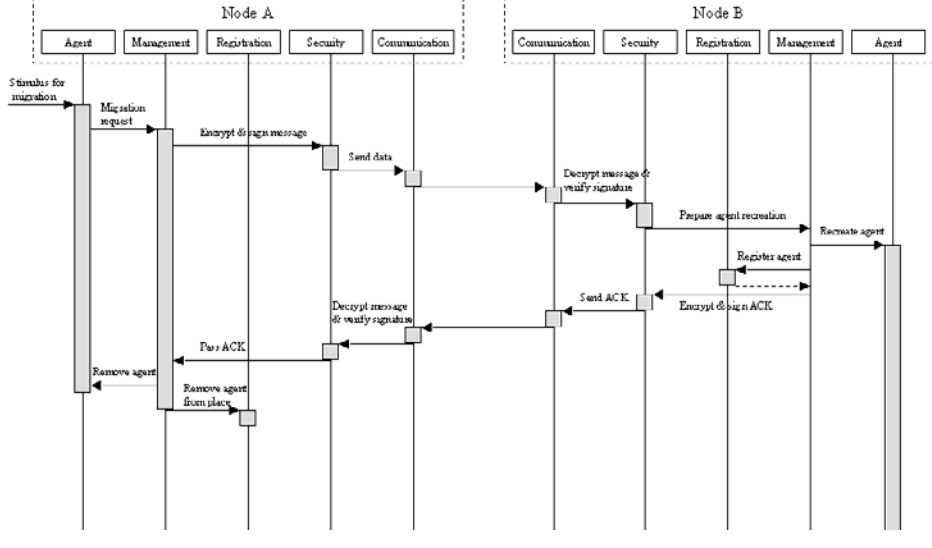


Figure 17. Agent migration.

The auction protocol used in the case study follows the pattern of an English auction, in which an auctioneer directs participants to beat the current, standing bit. At the beginning of the auction, the auctioneer predefines a minimum bid, and since the first bid, every new bid must increase the current bid by a predefined minimum increment. The auction ends, when no one is interested in outbidding the current standing bid. The auction protocol is presented in Figure 18.

The auction protocol begins with the auctioneer agent sending a product offer to shopper agents. Shopper agents compare the offer received to the profiles representing their authors' interests. If a product is interesting and the author also agrees it, the agent accepts the offer by migrating to the node where auction takes place and registers itself with the auction. If the predefined time to start the auction is on hands, and only one shopper agent is registered with the auction, the auctioneer has to inform the shopper that the auction has been cancelled due to the lack of participants. However, usually there are enough bidders and when the predefined time has come, the auction begins and shopper agents can start bidding. When a new bid is sent by a shopper agent, the auctioneer checks it and if it is appropriate, a message informing acceptance of the bid is sent back. Also, every other participant is informed about the new bid. This continues until no one can be found to outbid the current standing bid. Then, the auctioneer agent waits a while and announces that the auction has finished and the participant, who made the final bid, is the winner and is obligated to pay the bid amount.

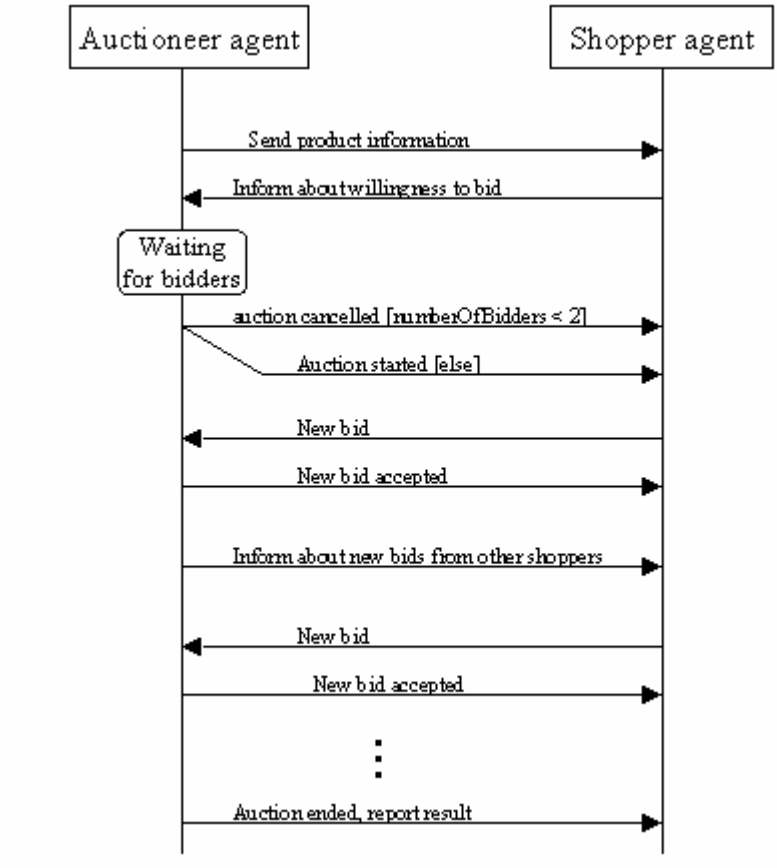


Figure 18. Auction protocol.

5.4.3 Deployment view

The deployment view describes the system's hardware and software components and their relationship. To visualise the view, we introduce the deployment diagram presented in Figure 19, which is a graph of nodes connected by communication associations. The system's auction and shopper nodes are similar to each other in the way that they contain the same agent platform and thus provide the same services. However, there are differences in the active objects between nodes, because usually only one node is assigned as an auction node, containing the instance of the auctioneer agent, while other nodes contain only instances of shopper agents. Instances of the agents can be created and

terminated during the run-time, thus the auction place can be changed without reinitialising the agent platforms.

Nodes are connected to each other through the Internet using the TCP/IP protocol. Every node has the ability to accept a new connection request and make connection requests to other nodes. Connections are handled manually by the agent platform administrator, although only one connection to the auction node is usually needed. Since the administrator has to create a connection to the auction node, he or she has to know the host IP address and the port number. If the system would contain a large amount of nodes, the directory facilitator would be needed to get information about the addresses of auction nodes and improving their availability. However, the directory facilitator is not implemented in this case study, since the functionality of the agent system can also be proved without it. A certificate authority is another node that should be implemented, if the system is used on a large scale and no other secure way of changing certificates can be found. The directory facilitator and the certificate authority are depicted with boxes of dashed lines to represent their absence in this case study.

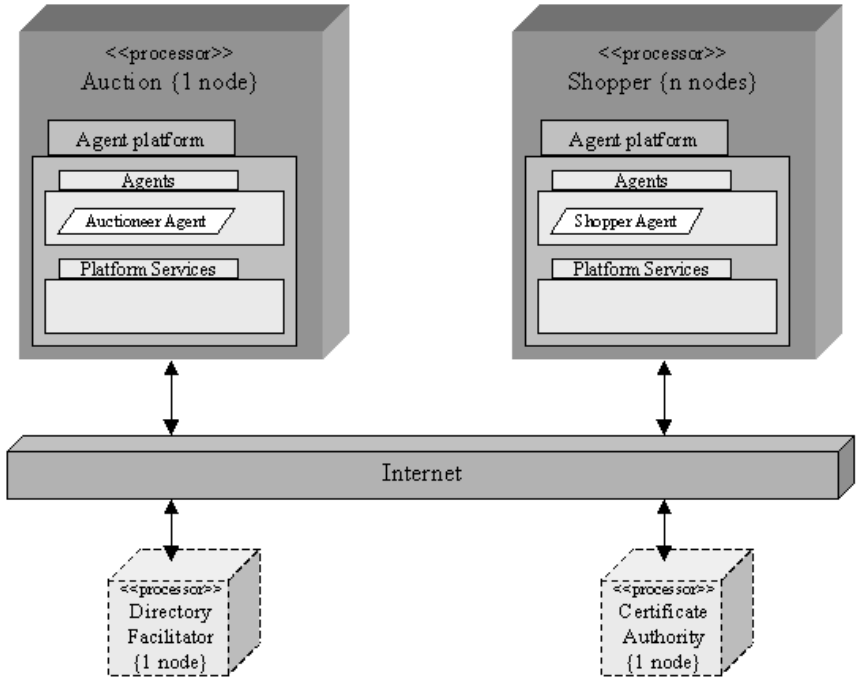


Figure 19. Deployment diagram of the system.

5.5 The security framework of the mobile agent system

Security of the mobile agent system is based on the security services provided by the agent platform and agent security layer. These build up the security framework for the agent system, which is a diverse combination of different security mechanisms. Each of these security mechanisms has a specific purpose and all of them are kind of a compromise with the availability of the system or trust upon the other parties involved. In the implementation of this mobile agent system, we have covered the security mechanisms as widely as reasonable without going too deep into specific details of certain mechanisms. Our aim has been to guarantee the well-balanced overall security of the agent system, because the security of the system is always as weak as its weakest link.

The principles of the mechanisms used in this Section are described in Chapters 3 and 4. In the following subsections, we explain how these mechanisms are implemented in our case study.

5.5.1 Platform security

The security of the agent platform is based on the security services it provides. Each of these services provides security for the platform from one aspect. For example, the key management service manages keys and certificates effectively and securely. However, security cannot be provided by the individual mechanisms, which would leave too big vulnerable gaps between them, but by a tight combination of these mechanisms. Thus, there are various interactions between these services that shape the security services into a tight net that cannot be passed easily without authorisation.

An overview of the platform security services is depicted in Figure 20. The figure illustrates the security services provided by the platform and files that are connected to those services. The internal connections of the security services are not provided in this figure to keep the representation clear. A description and implementation of each service are presented in the following.

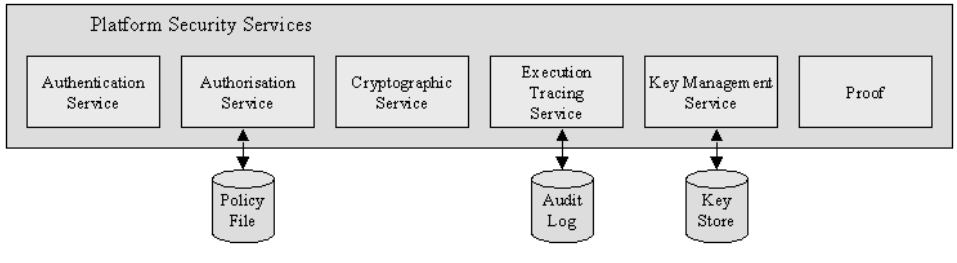


Figure 20. Platform security services.

Many of the security threats that the agent system faces are connected somehow to inter-platform communication. Thus, the form of messages sent between platforms is also essential when considering the security of the system. Figure 21 illustrates the type of envelope and message that are used to send data between agent platforms. Both include certain fields that are used to protect data during

transmission. The purpose of those fields is described in detail in the following subsections.

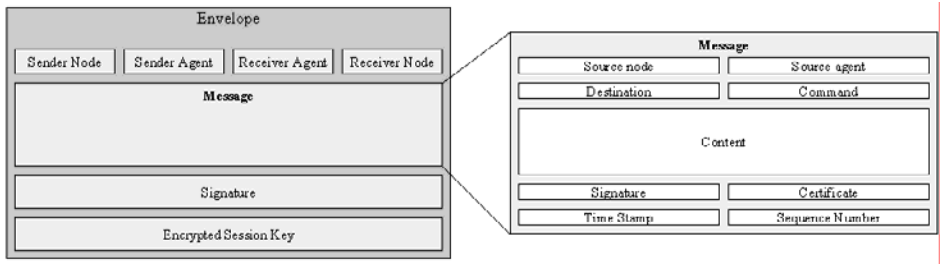


Figure 21. Structure of the message and the envelope.

Authentication

The main function of the authentication service is to sign agents and messages, and correspondingly verify their signatures. Digital signatures, introduced in Section 3.2, are created using the sender's private key for encryption. The algorithm used to calculate message digest is SHA-1. The encryption algorithm used in the process is RSA, which is reviewed in Section 3.1. Since Java does not provide the RSA algorithm, Bouncy Castle Crypto API has been used as the provider of the algorithm. Verification of the signature is performed using the public key of the sender. Digital signatures can also be used as proof of message integrity.

The reason that both the message and the envelope include a field for a signature is in their different usage. The envelope is signed by the sending platform, which guarantees the authenticity of the message, whereas the message signed by the agent proves the agent’s identity. Envelope signature changes every time a message changes. Therefore, the signature of the envelope is valid for only one transmission. On the contrary, an agent's signature can be used for the whole itinerary. However, the requirement for this is that all changes to the agent and the results are stored outside of the core agent, which thus remains unchanged.

The authentication service is used to verify an agent’s path history and to sign new path entries to it. Signing and verifying path histories uses the same algorithms that are used in singing messages and agents. If the agent platform notices that the path history of the mobile agent is deficient or invalid, the

authorisation service is notified of that, and proper action is taken to handle the found defect.

Authorisation

When a new mobile agent has migrated to the platform and its identity has been authenticated, the authorisation service is asked to issue rights for the agent. This is achieved by verifying the identity of the agent with the database of trusted and untrusted identities. If an agent's identity is found to be trusted, it is assigned with full access rights to the platform's services. On the other hand, if agent's identity is found from the list of untrusted identities, the agent's access to the platform may be totally denied or it may be assigned with restricted access rights. In the third case, if the agent's identity is not known to the platform, it is also assigned with restricted access rights. In the implemented system, agents are divided into four groups whose access rights can be separately defined. Those groups are local agents, trusted visitors, restricted visitors and untrusted visitors. If an agent is untrusted, its access to the platform is denied. Unknown visiting agents receive restricted access rights and they are not allowed to participate in auctions. Local agents and trusted visiting agents have full access to the platform's services.

Decisions on granting access to the platform and the level of access rights are dependent on the platform's security policy. It defines, for example, if an agent can be trusted, when a certificate including agent's public key has expired, how to act if an agent cannot be identified, or what to do if the time stamp of the message including an agent is no longer valid. These are questions that have to be considered beforehand in order for the platform to be able to make decisions by itself and those decisions are in a clear line with each other.

Cryptographic service

The cryptographic service is used by the platform to encrypt the message that is about to be sent. Since the principles of encrypting messages using both symmetric and asymmetric algorithms have already been discussed in Section 3.1, this section concentrates on defining their implementation in this case study.

Encrypting of the message begins with creating a random *session key* that by default uses the triple-DES encryption algorithm. With this symmetric key, a whole message is encrypted reliably, however, the delivery of the session key has still to be protected. Thereby, the session key is encrypted once more with the receiver's public key using the RSA encryption algorithm. As a result, the message is encrypted with the session key and the session key is encrypted with the public key and all of them are stored in the same envelope. A private RSA key cannot be used directly to encrypt the message, because the size of the message is too large for the algorithm and encryption with the public key algorithm is also much slower than using the conventional symmetrical algorithm.

The decryption procedure on a receiving platform begins after the communication service has deserialised the envelope and passed it to the security service. First, the encrypted session key is decrypted with the receiver's private RSA key, and after that, the resulting session key is used to decrypt the message inside the envelope. Now the plain text message can be read and proper action can be taken depending on the content of the message.

If there are shopper agents that are continuing their itinerary from one remote platform to another, and if the data they have gathered from a particular platform is too sensitive to be revealed to other platforms, the cryptographic service can be used by agents to encrypt sensitive gathered data. This encryption is carried out by the platform, but the key used in the encryption is the agent's public key, which it carries during its itinerary. These encrypted results are stored outside the core agent, which makes it possible for the next platform to authenticate the agent by verifying its signature. If the core of the agent, where the agent's personality is located, is changed, the signature is no longer valid and agent migration to new platforms is denied because of mistrust. When the agent finally returns to its home platform, the agent's private key, found from the platform's key store, can be used to decrypt the data that is collected during agent itinerary.

Execution tracing

Execution tracing monitors security-critical operations that are performed either by the platform or by the agents residing in it. The most important of such operations are:

- migration of an agent,
- recreation of a migrated agent,
- termination of an agent, and
- announcing a bid in an auction.

Single trace contains, at least, the identity of the principal that made the operation, the description of the operation and the time when it occurred.

Traces of the security-critical operations are stored in a non-repudiatable audit log, which is a file locating on the same node as the platform. The audit log is protected from unauthorised parties, because some of the information it contains may be used, if in the wrong hands, to do harm either to the agent platform or to some agents whose information can be found from the log.

Key manager

The key manager has an important task in the system, since it is responsible for all cryptographic keys and certificates that are used by the agents and agent platforms. These keys and certificates are stored in the key store file, located in the node where the agent platform is running. A screenshot of a key manager implemented for the agent system is depicted in Figure 22.

Figure 22. Screenshot of the key manager.

The key manager is responsible for storing users' own public/private key pairs and public keys in the form of certificates of their communicating peers. Only

secret keys, which are created randomly and used only once, are not stored in it. Although the key manager lacks the possibility to create certificates, Java's keytool (introduced in Section 3.4) provides creation of self-signed certificates, which are used in the system. If some certificate authority was linked to the agent system, it could be used to create real certificates. However, because of the high cost of using services from certificate authorities, self-signed certificates will meet the cause. The key manager can be used for adding already created certificates, which have been received with messages from the communication peer, to the key store or removing existing certificates from it. With the key manager, a user can also create new key stores, load already existing key stores from files or save key stores to files. The default key store is loaded during the platform initiation phase.

Using cryptographic methods to protect communication between nodes effectively solves communication security problems. Attackers are not usually interested in wasting their time on complex cryptanalysis. Instead, they might be looking for an easier way to break into the system, such as gaining access to the cryptographic keys, which would make all communication encrypted with those keys available for to attacker. That is why the key stores used are protected against malicious principals with passwords. In addition, all private keys within the key store are protected with specific passwords that are only known by the authorised principals. This password protection is provided by Java, thus the reliability of the technical implementation will not be concerned in this thesis.

Proof

Proof (also called as credentials) introduced in subsection 4.2.5, provides an assurance that the agent platform conforms to a certain security policy. Reliable implementation of the proof would require a trusted third party that would be able to certify the agent platform, assure that it does not act maliciously, and that it is what it claims to be. However, because of the lack of a trusted third party, we have to either trust blindly unknown platforms or restrict an agent’s operational area to trusted platforms. Neither of these two options is good, but we decided to limit the agent system to trusted platforms in order to keep the security of the platform at a high level. As a result, agents know every agent platform in the system personally. Therefore, being able to authenticate a platform should provide acceptable proof of its benevolence. If the agent

platform cannot be authenticated, agent migration to it can be denied until it has identified itself.

5.5.2 Agent security

Assuring the security of mobile agents in multi-agent systems is a very complicated task. In order to make this task a little easier, some assumptions and restrictions to the system have to be made. Introducing trusted platforms, which do not act maliciously against residing mobile agents, make it easier to cope with security problems, since without it the mobile agent is exposed to many severe threats on remote platforms.

Despite the fact that agent security against the platform it resides in cannot be effectively assured without trust, few other security mechanisms for agents can be introduced to make mobile agents' journeys safer. The agent security layer contains three mechanisms, as depicted in Figure 23, which provide protection for the agent in a particular aspect. Specific details of these mechanisms are presented in the following subsections.

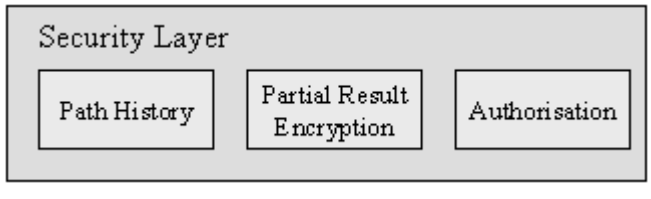


Figure 23. Agent security layer.

Path histories

Path history is a mechanism for protecting the agent by authenticating the itinerary that the agent has travelled during its journey. Path history provides the ability to detect if the agent has been to untrusted platforms. If this is the case, results from the agent can be regarded with suspicion. An incomplete path history is also a signal that something suspicious may have happened during the agent's journey. Detailed background information about the Path histories method is presented in subsection 4.3.1, and thus only the implementation is described here.

Digital signatures used in path entries are signed and verified by the agent platform authentication service, which provides specific methods for this. The signing algorithm used by the platform is the same that is used in signing agents and envelopes. Path histories are stored outside an agent's knowledge base, because they have to be easily accessible to remote platforms, and platforms should be able to add their own path entries to it. Updating the agent's path history is the agent platform's responsibility and thus success of this protecting mechanism requires good co-operation between the agent and agent platforms.

Encrypted payload

Encrypted payload is a mechanism to protect sensitive information, which a mobile agent has gathered during its itinerary from other agent platforms. The mechanism is based on partial result encryption introduced in subsection 4.3.2. The information that the mobile agent has gathered is encrypted with the agent's public key, stored inside the agent. Because of the characteristics of the public key cryptography, encrypted data cannot be decrypted without the private key, which is located at the agent's home platform.

To be able to encrypt received data, the agent needs help from the platform's security service that performs encryption on behalf of the agent. In the implemented system, information achieved from a particular platform is encrypted in one single block of encrypted data, while some data collected from another platform is encrypted and stored in another block. Although the system does not thereby support sliding encryption, this approach is more suitable for the mobile agent systems, where agents' itineraries are not very long. As a result, the encryption of results can be done easier but the size of the encrypted data becomes bigger when the itinerary gets longer.

Authorisation

In addition to the agent platform deciding which agents can be authorised to act within the agent platform, the agent itself can decide which agents they want to co-operate with. A precondition of the succeeded authorisation by the agent is that the agent platform has authenticated every principal that is acting inside the platform. Also, communication coming from principals outside the agent platform has to be authenticated.

Authorisation is realised with the list of principals that are unwanted communication partners by the agent. Every time the agent receives a message from another agent, it is forwarded to the agent's security layer. The identity of the message sender is compared to the list of unwanted identities, and if correspondence is found, the message is rejected. This method prevents direct intercourse with agents that do not want to collaborate, but indirectly this is still possible. In the auction event, for example, two shopper agents that deny collaborating with each other may, however, bid on the same product at the same time without any problem.

5.6 Results of the implementation

Implementation of the mobile agent system was a challenging and time-consuming task. Building the system started with relatively light objectives, but in time, new requirements for the system came up. Basically, iterative development process meant that new functions were added with slight changes to the existing system. However, in the development phase where security became the main viewpoint, some thorough changes to system architecture had to be made and many already existing software components needed some sort of revision.

The biggest decision that guided the development of the agent system was made already at the beginning. The degree of agent mobility, which was quite exceptionally decided not to support code mobility, set restrictions on many things, especially on the security architecture. Because of the chosen type of mobility, the decision that agents are only allowed to migrate to trusted platforms had to be made, since otherwise mobile agents would be unacceptably vulnerable to various threats caused by remote platforms. The most positive issue of the selected agent mobility was that the implementation of the platform security became easier, and protection of the platform could be mainly concentrated on authentication and authorisation of mobile agents.

The introduction of trusted platforms lead to two new problems: how to prove the trustworthiness of an agent platform and how certificate distribution and revocation can be arranged. The answer to the problems is a trusted third party, but the cost of building it proved to be beyond our capacity. Thus, only self-

signed certificates that can easily be created with Java were used in the agent system. However, it is evident that self-signed certificates do not provide enough security and trust. In addition, even trusted third parties are not completely safe unless the identities of principals are authenticated personally, which is without doubt unfeasible in large-scale agent systems. Therefore, this issue has to be solved before the developed agent system can be applied in commercial applications.

In some cases, the number of agent platforms connected to the agent system may increase so that the assumption of extending the network of trusted platforms is no longer reasonable. This scenario shows one weakness of the implemented agent system, which is poor scalability. Every new platform that is about to join the agent system has first to be able to assure its trustworthiness, and only after that it is it accepted to the system. This is an important issue to be taken care of in order to apply the developed agent system on a large-scale.

Although there are many things in the agent system that have some deficiencies, agent security against other agents and attacks from outside the agent system is implemented quite effectively. Implementation of inter-platform communication and agent migration has been realised using cryptographic mechanisms. Therefore, possible attackers do not waste their time with cryptanalysis, but they probably look for other vulnerabilities that can be more easily exploited. The agent system is also developed in such a way that agents residing in the same platform do not pose notable threat to each other.

An additional objective of building the agent system was to research the reusability possibilities of agents. Reusability of an agent platform has already been proved, for example, by FIPA. It can be achieved using standardised agent languages and communication interfaces, but there is still little evidence of agents' reusability. Agents developed for the auction are designed with a layered architectural style that should support reusability well. The case study showed that the agent definition layer, which is the "brain" of an agent, is very specific and its reuse is pointless in most cases. Developed agents also had co-ordination and organisation layers but functionality of these layers turned out to be quite low. One possibility might be to combine these layers into one in order to reduce complexity in agent data flow. However, too direct conclusions from their reusability should not be made based on this case study. In conclusion, the agent

security layer seems to be the most easily reusable, since it can quite directly be applied to different agents. However, it should be remembered that even adding a well-designed security layer to a new agent will not guarantee an agent's security. The security must be considered comprehensively on the agent level, because it is, at least, indirectly linked to every component of the system.

6. Robustness testing

When complex software is developed, its developers want to assure other parties that the software obeys the quality requirements put on it. This is not easy without some kind of concrete proof. Therefore, this chapter focuses on providing satisfying proof of the security of the implemented agent system.

As we have seen in this work, the security of mobile agent systems is a very broad issue. The main security mechanisms that were implemented in the case study, like authentication or agent encryption, rely on cryptographic algorithms, which are mostly provided by Java. It would be interesting to test the vulnerability of these algorithms with the help of cryptanalysis tools, however, in that case the test would be mainly directed at the security of the cryptographic algorithms provided by Java, which is not our purpose. Many other attack scenarios against the system also seem interesting. For example, eavesdropping or replaying the packet can be seen as potential threats. However, thorough testing of the security of the auction implementation would require too many resources to be feasible for this thesis. Therefore, testing of the developed agent system is limited to testing its functional robustness.

Robustness of the auction implementation is tested by inserting illegal exceptional elements, called anomalies, inside the mobile agent. The agent then migrates to the auction place in order to participate in the auction. The agent platform's reactions to the altered agent are observed and all misbehaviour is reported. If the agent platform detects that the received agent has been altered, its access to the platform is denied. However, the platform’s ability to detect altered agents is not perfect, and an altered agent may be accepted into the platform. When the platform is exposed to a misbehaving agent, various problems may occur since the platform has not been specially designed to handle agent misbehaviour.

Testing of the system’s robustness is based on a mini-simulation method, which uses multiple simple miniature simulations instead of a single complete one. Background information about robustness testing and a mini-simulation method is discussed in the following section. Actual test cases are generated using the Codenomicon mini-simulation toolkit that is described in Section 6.2.

6.1 Mini-simulation method

Software robustness can be considered as the ability of software to tolerate exceptional input and stressful environment conditions. If a software component that is not robust faces such a circumstance, it will fail, and the opportunity for a malicious intruder to take advantage of robustness shortcomings is offered. Deficiencies in software robustness can be identified and quantitative figures about software's robustness can be provided with the help of robustness testing. Thereby, robustness testing also provides the possibility to increase software quality particularly through the enhancement of security and reliability [3].

The mini-simulation method was originally developed for functional robustness testing. The robustness testing methodology can be used to test any protocol implementation. The testing method is intended for use during the initial implementation phase, all the way to validating released products and evaluating other available implementations. The method's main requirement is the ability to generate a large amount of messages with one or few exceptional elements but otherwise legal content [23].

The mini-simulation method is based on software *fault injection*, which means that artificial faults called anomalies are intentionally injected into software components for analysis purposes. Analysing is realised by monitoring the output of the software after an anomaly has been injected into the system. The anomaly is considered to be an unexpected event that has the potential to alter software behaviour through the alteration or corruption of its internal state. The main objectives of fault injection are to understand the effects of real faults, forecast expected system behaviour and get feedback for system correction and enhancement. Software faults are always the result of incorrect design or implementation [23].

6.2 Mini-simulation toolkit

A prototype of the mini-simulation toolkit was originally developed in the PROTOS project, which was a joint effort of Oulu University Secure Programming Group (OUSPG) and VTT Technical Research Centre of Finland.

Later on, in 2001, Codenomicon was founded to carry on the development work of the testing framework.

Building of the test tool begins by writing a protocol specification using a formal language for input into the toolkit. Then the toolkit reads the specification and builds an internal model of the protocol grammar. In the next phase, the model can be modified by inserting various anomalies into it. When the protocol model has been modified, all valid and invalid protocol exchanges are produced one by one. As a result, a number of individual test cases are created.

6.3 Test cases

This section takes a closer look at test case design and the test cases, which were implemented for testing the security of the case study. Before the test material for studying the robustness of the agent system was ready to be run, a lot of test design and work with the mini-simulation toolkit had to be done. The test design process that was followed before entering the actual testing stage, included five phases:

1. Writing the protocol specification.
2. Creating a valid test case.
3. Designing and inserting anomalies.
4. Selecting test cases.
5. Generating and running test material.

The first step of the test design had fortunately already been provided by the Codenomicon toolkit, since it supported the Java serialisation format, which is also the format used in agent migration. Thus, the test design could proceed straight on to creating a valid test case. A valid test case was needed to confirm that the agent platform accepts the agent, which was processed with the toolkit, although anomalies had not yet been injected in it. After validating the valid case, anomalies were designed and injected into agents. The shopper agent that was used for test purposes was intentionally made simple enough to keep the number of invalid test cases reasonable. The decision on which anomalies to be

used in the test cases was made by considering the possible effects they would have on the system. Finally, a set of individual test cases was generated with the toolkit and they were injected into the agent system one by one straight from the output of the toolkit.

Although there are a number of different complications and problems that inserted anomaly may cause to the testable system, robustness testing has few very common mistakes that may be caused by tested software components. While testing the agent system, these expected mistakes were monitored very carefully. Three of the most common and severe misbehaviours are [3]:

- Crash, followed by a possible restart.
- Hang: a busy loop leading to a denial-of-service situation.
- Failure of a component leading to a denial-of-service.

Selected and implemented test cases are reported in Table 3. The table contains the test case number (#), type and name of the variable where the anomaly is inserted, original value of the variable, inserted anomaly, and the effect of how the system responded to the inserted anomaly. In most test cases, several anomalies are linked to the same variable with the same original value. Although there exist numerous anomalies that would have the potential to inflict the system, to keep the test case number reasonable, only those that were seen as most potential sources of misbehaviour were included in Table 3.

Table 3. Test cases.

#	Type / Variable name	Original value	Inserted Anomaly	Effect
1	Boolean isAuctioneer	0x00	0x01	No effect
2			null	No effect
3	Boolean isShopper	0x01	0x00	Exception thrown and handled by the platform. Creation of the agent cancelled.
4			0xFF	No effect, agent is handled as a shopper agent.
5			null	Exception thrown and handled by the platform.
6	Boolean isInterested	0x01	0x00	Agent does not start bidding.
7			null	Exception thrown and handled by the platform. Creation of the agent cancelled.
8	Integer priceLimit		negative values or 0	Agent does not have willingness to bid at all. If every bidder has priceLimit<=0, auctioneer buys product. No errors.
9			2147483647 or other extremely big integer	Agent is actually willing to bid to the highest price and wins the auction. If two agents are willing to bid to the highest price, progress of the auction will be extremely slow since they use the smallest allowed raise (100). *Possibility for denial-of-service attack, by slowing down the operation of the auctioneer agent, detected.*
10	String name	Smith	null	Agent creation failed because of the null pointer exception.
11			100 x "\r\n"	Agent name is displayed incorrectly every time it is referred. No other misbehaviour detected.

#	Type / Variable name	Original value	Inserted Anomaly	Effect
12	String type	shopper	null	No effect on agent creation, but null pointer exception is thrown, when trying to view the list of agents residing in that platform. Thus, neither the inserted agent nor any other agent can be viewed or reconfigured. *Possibility for denial-of-service detected.*
13			auctioneer	The platform displays the wrong agent type when viewing information of residing agents. No other misbehaviour detected.
14	String homeNode	PC2	null	Migration back to the home node failed and null pointer exception thrown. The agent does not have the home node, and therefore it will be terminated by the remote platform.
15			PC1	The agent tries to migrate back to the home node, but it goes to a wrong platform and will be terminated.
16	Object AgentKnowledgeBase		null	Class not found exception, creation of agent cancelled.
17			16000 x "a"	Java HotSpot(TM) Client VM warning: increase BUFLEN in ostream.cpp -- output truncated java.lang.ClassNotFoundException. --> Exception caught and agent creation cancelled.
18	Object DfBase		null	Exception caught and creation of the agent cancelled.

6.4 Results of the tests

As results of the tests on the implemented agent system, two misbehaviours by the agent platform were detected. Both of them leaded to the possibility of a denial-of-service attack, but any severe misbehaviour that would have led to the crash of the agent system was not recognised. Testing of the system's robustness was done on a quite small scale, which was reasonable in this thesis. However, it is probable that there are many implementation vulnerabilities, which were not found in these tests. Therefore, if the implemented system was a commercial application, its robustness should be tested much more carefully. If errors are detected after the release of the product, consequences are much more troublesome and expensive.

The first misbehaviour occurred when an extremely large integer was inserted in the priceLimit-variable. Normally this is prevented by checking that the value, which a user has inserted, is within convenient limits, but if an anomaly is inserted after the acceptance to participate in an auction, an invalid value will not be detected. If a malicious user succeeds in infiltrating two altered agents in the same auction, where both agents have extremely high bidding limits, it will cause the auction to last for many days or in the worst case for even months. As a consequence, all participating shopper agents and the auctioneer agent would be tied to the auction event far too long, which would also burden the platform’s other resources.

Detected vulnerability is not very easy to implement, since it would require either an ability to alter the mobile agent during its migration or, alternatively, a malicious user should inject an anomaly to the agent already at the platform. In the first case, cryptographic protection mechanisms protecting an agent during migration would have to be broken, which is not an easy task considering the time limits due the time stamp included in the message. In the second case, a malicious user would probably succeed in deceiving the platform, but once the fraud is detected, the trust on the user would also be lost, which acts as a strong deterrent. Thus, the criticality of the vulnerability can be considered as moderate at the maximum.

Another detected misbehaviour is connected to the software component that is used in viewing and configuring agents residing on the platform. It is exposed to

a denial-of-service situation if the value of the type-variable is null. The null value causes the platform to not be able to view or configure any of the agents residing inside the platform, because the exception is thrown at every attempt. Additionally, the platform cannot know which agent causes this denial-of-service situation. Thus, the only opportunity is to start migrating agents to their home platforms or terminating them until the failure is found. However, the criticality of this vulnerability is not very severe, since the platform's ability to view or reconfigure agents is not crucial.

Both detected vulnerabilities are quite simple in a sense in what improvements have to be carried out on the agent system to take these into consideration. Only small changes to the agent platform's code have to be made to prevent these vulnerabilities.

As mentioned before, the implemented agent system involved various security mechanisms, which would all be good to test in order to confirm the security and robustness of the agent system. Although, these implemented tests took only one aspect into account, it was quite essential considering the overall security of the agent system.

One possible weakness of the implemented testing procedure was that the only person who participated in the test case design, was the actual designer and realiser of the agent system. Thus, there is a possibility that during the test case design, some essential viewpoints that should be considered might have been left out, because of the system designer's blindness to its own faults. Therefore, it would be better if someone from outside the systems development team would also participate in test design process.

7. Discussion

This chapter analyses the presented work against research problems that were described in Chapter 1. In addition, future development possibilities of the presented case study and the future of mobile agent systems will be discussed.

In this thesis, the security mechanisms needed for building a secure auction implementation was studied to build an agent security framework, which contained the platform's security service and agent security layer. Used mechanisms covered, a wide-range of different security aspects and the security framework was implemented quite successfully. In particular, communication security and the security of the agent platform were provided well. Although finding a complete answer to the malicious host problem was considered unlikely beforehand, the problem turned out to be surprisingly severe and caused greatest concern.

In order to apply the security framework to another application, some basic things have to be considered. Firstly, the required level of security is dependent on the purpose of the agent system. Therefore, some mechanisms will probably be unnecessary and some other has to be added in order that the different requirements of other applications can be met. Secondly, if the type of mobility changes by resorting to code mobility, the whole arrangement must be reconsidered, since code mobility sets additional security requirements that were not considered in this work.

The difficulties of implementing an auction for mobile agents were numerous. Most of these were somehow associated with the problem of securing a mobile agent inside a remote platform. The biggest concern is that there is no efficient way of preventing the agent platform from eavesdropping data inside the agent. Thus, the platform can easily find out how high the mobile agent is willing to bid. Then it can raise the price using its own shopper agents to a suitable level, and get the highest possible price from the product. Introducing trusted platforms, which are so reliable that they will not perform malicious actions, would solve the problem. However, in real life, the reliability of trusted platforms should be at the same level as banks; otherwise this assumption is not reasonable. Therefore, the implemented agent system is too insecure to be used in real life applications.

Additionally, people are not yet ready to let mobile agents handle tasks that involve financial aspects; they would rather do them by themselves. Thus, a better task for mobile agents would be information gathering. For example, an agent could find interesting products from different virtual auctions, verify those with the user profile and inform the user where the interesting product was found and when the auction is to be held. Then the user could handle the bidding process, in which case there would not be the possibility to deceive the user to bid higher than he or she wants.

7.1 Realised quality requirements in the case study

In Section 2.5, various quality requirements for mobile agent systems were identified. To summarise, five of the most important requirements were examined in detail. In addition, security, which was emphasised in the case study as the most important quality requirement, was further divided into four subcategories. These recognised requirements are:

- interoperability,
- scalability,
- mobility,
- security,
  - confidentiality,
  - integrity,
  - accountability,
  - availability, and
- robustness.

In the following, we try to analyse how these quality requirements were met in the agent system implemented in this thesis.

The implemented agent system included two agent types, which were designed to co-operate with each other in order that the auction event could be followed through. In addition, there also existed competition between shopper agents, which were bidding in the same auction. *Interoperability* between agents was

attained successfully using a shared agent communication language and a predefined format of context of communication. Another important part of interoperability between agents was the common auction protocol, which assured reliability and functionality of auction events. However, interoperability between agent systems, in other words, the ability of our agents and agent platforms to communicate or share tasks between other agent systems has not been taken into account when designing and implementing the agent system. Thus, this is an independent agent system, which does not obey interoperability standards developed for mobile agent systems. Therefore, it cannot be used with the co-operation of other agent systems, at least, without modification.

The fact that the implemented agent system does not obey interoperability standards also weakens the *scalability* of the agent system. However, the effect on scalability is not very severe, since the auction implementation is designed to be used for special purpose. Thus, it does not have to understand agents from different agent systems. From the viewpoint of a single agent, the agent system's scalability can be considered to be quite reasonable. When the number of agents in a system increases, some systems services may slow down. For example, an agent may have to wait before it is allowed access to the platform. However, every agent that takes part in the auction event has similar possibilities to make their bids despite the number of participating shopper agents. On the other hand, an increase of the number of agents in the system results in a light weakening of the system's ability to provide services to agents. However, if the burden increases too much, services and auction events can be distributed to new platforms in order to lighten the load that is directed to the specific platform. The most severe deficiency in the scalability is the difficulty to add new platforms to the agent system, which is caused by the assumption of trusted platforms. Every time a new platform is added to the system, the platform has to be first authenticated properly in order to provide proof of its trustworthiness. This has to be done partly manually, which causes additional work for the system administrator.

Contrary to generally used code *mobility* in agent systems, our auction implementation used a more restricted type of mobility. Experiences of the tested type of mobility were two-fold. On the one hand, the security of the agent platform can be provided better in this way. Other security threats did not have significant differences compared to using code mobility. On the other hand, use

of this type of mobility restricts the agent system's ability to adapt dynamically. All agents of the same type have the same available operations. If new operations are added to agents, all agent platforms have to be updated as well in order that they can accept these altered agents. Therefore, an agent's ability to operate in heterogeneous environments is also decreased.

The agent system has to be able to keep data inside it confidential and accessible only to authorised principles. Providing *confidentiality* for the data during its transmission is handled quite efficiently with conventional security mechanisms. Also, confidentiality of the agent platform's private data is protected effectively. However, the biggest problem occurs when a mobile agent that contains private data, migrates to a remote platform, where it is submitted to the control of the platform. Thus, if the platform is malicious, it can easily eavesdrop on confidential information in the mobile agent. One solution for the problem is partial result encryption, which was used in this work to encrypt the results of mobile agents to ensure that collected data is not accessible to other remote platforms. However, this mechanism cannot be used to protect private data, which has to be accessible to the agent itself during its itinerary. There are also other proposed protection mechanisms, but one single efficient mechanism for preventing misuse has not yet been developed. Therefore, most of the agent systems, including ours, resort on trusted platforms. The trusted platform is, however, only a temporary solution to the problem and it raises new problems to be answered. Although trust relationships inside an agent system might work on a small scale, in a large-scale real-world agent system, there always exists somebody who will exploit trust. Thus, this is the biggest problem in a way of making the use of mobile agents secure.

Integrity of the data in the agent system is, along with confidentiality, the most important security requirements. In the implemented case study, integrity of the platform's data was protected effectively. During the migration of mobile agents, data integrity is also protected efficiently using digital signatures. Although malicious principals would be capable of altering or terminating the mobile agent during its transmission, illegal actions are detected and the mobile agent is sent again until it has been received successfully.

Protecting the integrity of a mobile agent's data is much more troublesome, because a malicious agent platform cannot be totally prevented from altering a

mobile agent's data or terminating the agent completely. However, there are a lot of mechanisms for detecting the alteration, which can be used in order to provide data integrity for mobile agents. In this work, digital signatures were used to protect the integrity of the mobile agents. Using digital signatures requires that the mobile agent stays constant during its journey. In other words, all acquired results have to be stored outside the agent's core otherwise the signature becomes invalid. In addition, mobile agents should not be irreplaceable, in which case the termination of the agent would cause significant damage. Instead, the author of the agent has to prepare for the worst case in which the agent does not return from its journey. Using replicated agents it is possible to improve a mobile agent's resistance against the threat of termination.

Accountability is an important quality requirement in e-commerce agent applications. There has to be the possibility that authors of mobile agents' can be held responsible for the actions of agents, which they have authorised. In this work, accountability was carried out by authenticating all principals acting inside the agent system and recording all security-relevant actions in audit logs. However, in large-scale applications, there exist some malicious principals that may succeed in either bypassing the authentication procedure or exploiting the authority of other principals. If the trust relationships between parties in the agent system are not personal, the opportunity to misuse trust increases, since the deterrent of losing ones trust because of misuse is no longer very efficient. Therefore, a procedure of handling misuses has to be considered beforehand.

Availability of the agent system is very difficult to test, since the number of agents during tests is much smaller than in real-life applications. However, some conclusions of it can be made. A possible vulnerability in our agent platform is a potential denial-of-service situation, where malicious principals intentionally send a number of mobile agents to the platform in order to block platform authentication functions. As a result, decent mobile agents are prevented from accessing the agent platform. If this attack originates from several different sources at the same time, there is no effective mechanism for preventing it, which is quite a severe weakness. On the contrary, availability of the services for agents, which have already been authenticated and accessed the system, is quite good. Mobile agents cannot intentionally overburden platform services and the number of agents that are allowed to be registered to the same place at the same time can be restricted. Thus, except the two denial-of-service possibilities,

reported in Section 6.4, availability of platform services for mobile agents is protected. Yet another thing is the restrictions to the availability because of relying on trusted platforms. For this reason, the agent system's availability to possible users who have not been authenticated is weak.

Robustness is the requirement that was tested in Chapter 6. Only two security vulnerabilities were found, which alludes that the robustness of the agent system is very good. However, the robustness testing was implemented on quite a small scale. Thus, the robustness cannot be defined accurately based on these results and more tests should be made, because there are still probably many implementation vulnerabilities that have not been detected.

Since many quality requirements have been laid on the agent system implemented in this work, there exist lots of different interactions between these requirements. These interactions are reported in Table 4, where the legend is as follows: o = no interaction, + = positive interaction, and - = negative interaction. In addition, if there are more + or - marks in the same cell, it means that the interaction is stronger. The table is read as follows: when the quality requirement on the vertical column is enhanced, how are the other requirements on the horizontal rows affected.

Interactions reported in Table 4 are based on the case study in this thesis. Thus, these results are based only on one reference and therefore direct generalisations cannot be made. However, the expected results from the affect of mobility to the security can be seen. An increase in the degree of mobility decreases the total security of the system and, in particular, confidentiality is affected. In addition, if the agent system is more robust, it is also able to provide confidentiality of the data inside it, because malicious principals therefore are not able take advantage of programming mistakes made to the agent system.

Table 4. Interactions between quality requirements.

	Interoperability	Scalability	Mobility	Robustness	Confidentiality	Integrity	Accountability	Availability
Interoperability		o	+	o	+	+	+	+
Scalability	+		o	o	o	o	o	o
Mobility	o	+		o	o	o	o	o
Robustness	o	o	-		o	o	o	o
Confidentiality	o	-	---	++		+	++	o
Integrity	+	-	--	+	+		+	o
Accountability	o	-	-	+	++	++		+
Availability	+	+	-	+	o	+	o	

7.2 Future of mobile agent systems

A few years ago, when mobile agents were a big trend, it seemed like agents would solve almost every problem in the future. Now, when the hype has died down, we can look at the possibilities of agents more realistically, and see the real opportunities that they can offer. Future mobile agents are no longer seen as omniscient applications able to learn and do all kinds of tasks, but rather as applications, which are designed for specific tasks assigned to them.

Despite the reported security weaknesses in the implemented agent system, the future of mobile agent systems can be seen as promising, however, a lot of work has still to be done. In particular, the answer to the question, how the mobile agent can be protected efficiently while it resides in a remote platform, has to be answered. If the answer can be found, relying on trusted platforms could be forgotten, which would open a number of new opportunities for agents. Security mechanisms such as partial result encryption are small steps in the right direction, and also many other mechanisms have been proposed in order to improve mobile agent security. Although the complete answer to the problem

could not be found, mobile agents can be used in many tasks using today's technology.

Degree of mobility is another important question. In this work, we resorted to the type of mobility, which was assumed to be safer and more appropriate for applications that require high security. However, the introduced mobility could not help with the malicious host problem and its advantages, compared to using mobile code, were relatively small. On the other hand, the advantages of using mobile code are preferable. Agent mobility based on mobile code might have been more appropriate for this type of auction application, because if the mobile agent is running on the restricted area on a remote platform, it has better possibilities to protect itself than in the agent system implemented in this work.

In the future, research will be probably focus around FIPA, which is developing interoperability between agent systems and the standardisation of agent technologies. Therefore, it might have been better if our agent system would also have been developed according to the interoperability standards of FIPA. In that case, our agent system would have better supported the challenge of future agent development, which is concentrating on developing robust large-scale problem-solving activity and supporting functions for it.

Today, developed agent systems are mainly in the form of applications, but with time, those will become part of the operating system and application environment [5]. New tools will arise to make it easier for non-specialists to develop agent applications. Agent-related COTS-components are also appearing on the market to provide the building blocks for agent system developers and thus helping in the building process. The success of mobile agent technology in the future will depend largely on unsolved security issues around them. If answers to the problems can be found, the opportunities of mobile agents are limited only by our imaginations.

8. Conclusions

Problems in security have been seen as an obstacle in the way of success of mobile agent technology. In this thesis, one specific mobile agent application was developed intending to make it as secure as possible. For this purpose, the quality requirements of mobile agents were studied and several security mechanisms were examined in order that the most suitable ones could be selected to be included in the agent security architecture used in the implementation.

Security of the implemented agent system was provided with various security mechanisms. Communication security for agent migration was implemented successfully using traditional security mechanisms such as cryptographic algorithms, digital signatures and time stamps. Agent platform security was also provided effectively, but the biggest problem faced with was the securing of mobile agents residing in remote platforms. Satisfactory security for mobile agents could not be provided without resorting to trusted platforms. However, the use of trusted platforms caused new problems, because the certificate and key exchange had to be carried out manually, which resulted in scalability problems. In addition, the assumption that trusted remote platforms are benevolent and that they do not take advantage of the information, which is easily available in mobile agents, is not justifiable in real life large-scale applications. Therefore, in order for this kind of mobile agent auction to be used as a commercial application, the security problem of the malicious platform has first to be solved.

The type of mobility used in this thesis provides good protection for the agent platform. However, if the protection of the agent platform does not need to be particularly effective, advantages of the presented mobility against code mobility decreases. In addition, a migration mechanism based on code mobility evidently has its own advantages in dynamics, and it also provides better possibilities in protecting mobile agents. In summary, the presented mobility should only be applied in special applications that require specific protection for the agent platform, and thus code mobility would be a more appropriate solution for the migration mechanism to most agent applications.

To conclude, the time is not yet ready for the virtual auction, where mobile agents make decisions on behalf of their owners. Most people want to decide on their own business, which include financial aspects, by themselves. At least, if there is a danger that the mobile agent might make a false decision. However, mobile agents can be used for many other tasks, which do not involve financial aspects, and thus are less risky.

References

[1] Bradshaw, J.M., Greaves, M., Holmback, H., Jansen, W.A., Karygiannis, T., Silverman, B., Suri, N. & Wong, A. (1999) Agents for the Masses? IEEE Intelligent Systems, Vol. 14, No. 2, pp. 53–63.

[2] The Foundation for Intelligent Physical Agents (23.10.2003) URL: http://www.fipa.org/.

[3] Codenomicon (16.12.2003) URL: http://www.codenomicon.com.

[4] Franklin, S. & Graesser, A. (1996) Is it an agent, or just a program: a taxonomy for autonomous agents. In: 3rd International Workshop on Agent Theories, Architecture, and Languages. Springer-Verlag. Pp. 21–35.

[5] Agent Platform Special Interest Group (2000) Agent technology, Green Paper. Object Management Group. 67 p.

[6] Bradshaw, J.M. (1997) Software Agents. AAAI Press / The MIT Press, 480 p.

[7] Pham, V.A. & Karmouch, A. (1998) Mobile Software Agents: An Overview. IEEE Communications Magazine, Vol. 36, No. 7, pp. 26–37.

[8] Cabri, G., Leonardi, L. & Zambonelli, F. (2000) Weak and strong mobility in Mobile agent applications. In: 2nd International Conference and Exhibition on The Practical Application of Java, April, Manchester, UK. 15 p.

[9] Jansen, W. & Karygiannis, T. (2000) NIST Special Publication 800-19 Mobile Agent Security, National Institute of Standards and Technology. 38 p.

[10] Flores, R.A. (1999) Towards the Standardization of Multi-agent System Architectures: An Overview. ACM Crossroads, Special Issue on Intelligent Agents, Association for Computer Machinery, Vol. 5, No. 4, pp. 18–24.

[11] Lucena, C., Garcia, A., Sardinha, J., Castro, J., Romanovsky, A., Alencar, P. & Cowan D. (2003) Software Engineering for Large-Scale Multi-Agent Systems. In: International Conference on Software Engineering, May 3–11, Portland, Oregon, USA. Pp. 1–2.

[12] Milojicic, D., et al. (1998) The OMG Mobile Agent System Interoperability Facility. In: Rothermel K. & Hohl F. (Eds.) Mobile Agents. Springer-Verlag, Stuttgart, Germany. Pp. 50–67.

[13] Richards, M. (12.11.2003) The state of security standards for mobile agents. URL: http://www.sbaer.uca.edu/Research/2002/dsi/papers/053.pdf.

[14] Ndumu, D.T. & Nwana, H.S. (1996) Research and Development Challenges for Agent-Based Systems. In: IEEE Proceedings on Software Engineering, Vol. 144, No. 1, pp. 2–10.

[15] Grasshopper (11.12.2003) URL: http://www.grasshopper.de.

[16] Ajanta - Mobile Agent Research Project (11.12.2003) URL: http://www.cs.umn .edu/Ajanta.

[17] Karnik, N. & Tripathi, A. (2001) Security in Ajanta Mobile Agent System. Software - Practice and Experience, Vol. 31, No. 4, pp. 301–329.

[18] Mylopoulos, J., Kolp, M. & Giorgini, P. (15.11.2003) Agent-Oriented Software Development. http://www.science.unitn.it/tropos/hai-jm.pdf.

[19] Lee, L.C., Nwana, H.S., Ndumu, D.T. & Wilde, P.D. (1998) The stability, scalability and performance of multi-agent systems. BT Technology Journal, Vol. 16, No. 3, pp. 94–103.

[20] Stallings, W. (1999) Cryptography and Network Security: Principles and Practice, Second Edition. Prentice Hall, Inc. 569 p.

[21] Borselius, N. (2002) Mobile agent security. Electronics & Communication Engineering Journal, Vol. 14, No. 5, pp. 211–218.

[22] Software Engineering Institute (8.12.2003) URL: http://www.sei.cmu.edu/str /indexes/glossary/accountability.html.

[23] Kaksonen, R. (2001) A Functional Method for Assessing Protocol Implementation Security. VTT Publications 448. Technical Research Centre of Finland, Espoo. 128 p. + app. 15 p.

[24] Gollman, D. (1999) Computer Security. John Wiley & Sons, Inc. 320 p.

[25] VeriSign. Inc. (15.12.2003) URL: http://www.verisign.com.

[26] Entrust (15.12.2003) URL: http://www.entrust.com.

[27] X.509 Certificates and Certificate Revocation Lists (CRLs) (1.12.2003) Sun Microsystems. URL: http://java.sun.com/j2se/1.4.2/docs/guide/security /cert3.html.

[28] Venners, B. (1999) Inside the Java 2 Virtual Machine. McGraw-Hill, Inc. 703 p.

[29] Gong, Li (15.12.2003) Java Security Architecture (JDK 1.2). URL: http://java.sun.com/products/jdk/1.2/docs/guide/security/spec/security-spec.doc.html.

[30] Fritzinger, J.S. & Mueller, M. (2.12.2003) Java Security, A White Paper. Sun Microsystems. URL: http://java.sun.com/docs/white/index.html.

[31] Jansen, W.A. (20.1.2004) Mobile Agents and Security. URL: http://citeseer.nj.nec .com/jansen99mobile.html.

[32] Jansen, W.A. (2000) Countermeasures for Mobile Agent Security. Computer Communications, Vol. 23, No. 17, pp. 1667–1676.

[33] Tschudin, C.F. (1998) Mobile Agent Security. In: Klusch, M. (Ed.) Intelligent Information Agents - Agent based information discovery and management on the Internet. Springer-Verlag, Germany. Pp. 431–445.

[34] Vigna, G. (1997) Protecting Mobile Agents Through Tracing. In: 3rd ECOOP Workshop On Mobile Object Systems, Jyväskylä, Finland. Pp. 137–153.

[35] Hohl, F. (1998) Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts. In: Vigna, G. (Ed.) Mobile Agents and Security. Springer-Verlag, Lecture Notes in Computer Science No. 1419. Pp. 92–113.

[36] Young, A. & Yung, M. (1997) Sliding Encryption: A Cryptogarphic Tool for Mobile Agents. In: Fast Software Encryption: 4th International Workshop, January 20–22, Haifa, Israel, Lecture Notes in Computer Science 1267. Pp. 230–241.

[37] Roth, V. (1998) Secure Recording of Itineraries Through Cooperating Agents. In: 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations, Brussels, Belgium. Pp. 147–154.

[38] BouncyCastle (29.12.2003) URL: http://www.bouncycastle.org.

[39] FIPA Agent Management Specification (8.12.2003) The Foundation for Intelligent Physical Agents, Switzerland. URL: http://www.fipa.org/specs /fipa00023/.

[40] Matinlassi, M., Niemelä, E. & Dobrica, L. (2002) Quality-driven architecture design and quality analysis method. A revolutionary initiation approach to a product line architecture. VTT Publications 456. Technical Research Centre of Finland, Espoo. 128 p. + app. 10 p.

Published by

Series title, number and report code of publication

VTT Publications 538
VTT–PUBS–538

Author(s)
Wallin, Arto

Title

Secure auction for mobile agents

Abstract

In this work, a secure auction place for mobile agents has been developed and implemented. In the implemented auction application, software agents are able to bid on different products independently without user intervention in a secure manner. For implementation, quality requirements for mobile agent systems were studied. By defining the possible threats that a mobile agent system may face, a set of protection mechanisms were selected to build a security architecture that was used in protecting the agent system. The agent platform and the agents were designed based on the security architecture and the implementation was carried out using Java. Finally, in order to provide proof about the robustness of the implemented system, it was tested using a mini-simulation method.

As a result of the implementation, the communication security for transferring a mobile agent between nodes could be provided effectively by using traditional security mechanisms. However, the distribution of keys and certificates is required to be done manually so that the counterparts of the auction events can trust each other, which on the other hand, causes scalability problems. As a difference to most other agent platforms, the type of agent mobility was more restricted and code mobility was not used. For this reason, the security of the agent platform could be provided better. However, restrictions that had to be made to the system were quite big resulting in a decrease in the agent system's ability to adapt dynamically.

In summary, there are still a few problems that have to be overcome before the time is ready for large-scale mobile agent auctions. In particular, the security of the mobile agent residing on a remote platform has to be guaranteed without the assumption of trusted platforms.

Keywords
multi-agent systems, mobile agent systems, security architecture, robustness, security threats, protection

Activity unit
VTT Electronics, Kaitoväylä 1, P.O.Box 1100, FIN–90571 OULU, Finland

ISBN			Project number
951–38–6394–8 (soft back ed.) 951–38–6395–6 (URL:http://www.vtt.fi/inf/pdf/)			E1SU00357
Date	Language	Pages	Price
June 2004	English, Finnish abstr.	102 p.	C
Name of project		Commissioned by	
Puujulkaisujen toiminnallinen paloturvallisuustarkastelu		Food Focus Oy	
Series title and ISSN		Sold by	
VTT Publications 1235–0621 (soft back ed.) 1455–0849 (URL: http://www.vtt.fi/inf/pdf/)		VTT Information Service P.O.Box 2000, FIN–02044 VTT, Finland Phone internat. +358 9 456 4404 Fax +358 9 456 4374	

Julkaisija

VTT

Julkaisun sarja, numero ja raporttikoodi

VTT Publications 538
VTT–PUBS–538

Tekijä(t)
Wallin, Arto

Nimeke

Tietoturvallinen huutokauppa liikkuville agenteille

Tiivistelmä

Tässä työssä kehitetään ja toteutetaan huutokauppasovellus liikkuville ohjelmistoagenteille siten, että tietoturvallisuutta painotetaan tärkeimpänä laatuattribuuttina. Toteutettavassa sovelluksessa agentit, jotka pystyvät toimimaan itsenäisesti ilman ulkopuolista käyttäjän apua, käyvät huutokauppaa keskenään. Toteutusta varten määriteltiin liikkuvien agenttien laatuvaatimukset agenttijärjestelmälle. Lisäksi selvitettiin agenttien kohtaamat mahdolliset uhat, jotta voitiin muodostaa tietoturva-arkkitehtuuri, jonka avulla voitaisiin suojella agentteja toteutetussa agenttijärjestelmässä. Agenttijärjestelmän toteutuksessa kehitettiin sekä kaksi erillistä agenttityyppiä (ostaja ja huutokaupanpitäjä) että agenttialusta niille. Järjestelmä toteutettiin Java-ohjelmointikielellä. Lopuksi järjestelmän kyky sietää virheellistä informaatiota testattiin käyttämällä hyväksi minisimulaatiomenetelmää.

Työstä saatujen tulosten perusteella agentin siirtyminen agenttialustojen välillä pystytään turvaamaan hyvin perinteisiä tietoturvamekanismeja käyttäen. Avainten ja sertifikaattien vaihtaminen osapuolten välillä tuottaa kuitenkin ongelmia. Jotta vaihtaminen voitaisiin tehdä täysin luotettavasti, olisi se tehtävä henkilökohtaisesti. Tämä aiheuttaa puolestaan ongelmia järjestelmän skaalattavuutta ajatellen. Monista muista agenttialustoista poiketen tässä työssä käytettiin liikkuvaa koodia yksinkertaisempaa liikkuvuuden tyyppiä. Tämän ansiosta järjestelmän tietoturva, etenkin agenttialustan osalta, saatiin hieman paremmaksi, mutta käytetty liikkuvuuden tyyppi aiheutti vastaavasti agenttien dynaamisuuden heikkenemistä.

Yhteenvetona voidaan todeta, että liikkuvien agenttien järjestelmään liittyy vielä ongelmia, joita ei ole kyetty ratkaisemaan. Erityisesti etäagenttialustalla asustavan liikkuvan agentin tietoturvallisuutta ei kyetä suojelemaan tarpeeksi hyvin. Näin ollen täysin turvallisen liikkuvien agenttien huutokaupan toteuttaminen ei ole vielä toistaiseksi mahdollista.

Avainsanat
multi-agent systems, mobile agent systems, security architecture, robustness, security threats, protection

Toimintayksikkö
VTT Elektroniikka, Kaitoväylä 1, PL 1100, 90571 OULU

ISBN			Projektinumero
951–38–6394–8 (nid.) 951–38–6395–6 (URL: http://www.vtt.fi/inf/pdf/)			E1SU00357
Julkaisuaika Kesäkuu 2004	Kieli Englanti, suom. tiiv.	Sivuja 102 s.	Hinta C

Projektin nimi	Toimeksiantaja(t)
Puujulkaisujen toiminnallinen paloturvallisuustarkastelu	Food Focus Oy
Avainnimeke ja ISSN VTT Publications 1235–0621 (nid.) 1455–0849 (URL: http://www.vtt.fi/inf/pdf/)	Myynti: VTT Tietopalvelu PL 2000, 02044 VTT Puh. (09) 456 4404 Faksi (09) 456 4374

VTT PUBLICATIONS

517 Forsén, Holger & Tarvainen, Veikko. Sahatavaran jatkojalostuksen asettamat vaatimukset kuivauslaadulle ja eri tuotteille sopivat kuivausmenetelmät. 2003. 69 s. + liitt. 9 s.

518 Lappalainen, Jari T. J. Paperin- ja kartonginvalmistusprosessien mallinnus ja dynaaminen reaaliaikainen simulointi. 2004. 144 s.

519 Pakkala, Daniel. Lightweight distributed service platform for adaptive mobile services. 2004. 145 p. + app. 13 p.

520 Palonen, Hetti. Role of lignin in the enzymatic hydrolysis of lignocellulose. 2004. 80 p. + app. 62 p.

521 Mangs, Johan. On the fire dynamics of vehicles and electrical equipment. 2004. 62 p. + app. 101 p.

522 Jokinen, Tommi. Novel ways of using Nd:YAG laser for welding thick section austenitic stainless steel. 2004. 120 p. + app. 12 p.

523 Soininen, Juha-Pekka. Architecture design methods for application domain-specific integrated computer systems. 2004. 118 p. + app. 51 p.

524 Tolvanen, Merja. Mass balance determination for trace elements at coal-, peat- and bark-fired power plants. 2004. 139 p. + app. 90 p.

525 Mäntyniemi, Annukka, Pikkarainen, Minna & Taulavuori, Anne. A Framework for Off-The-Shelf Software Component Development and Maintenance Processes. 2004. 127 p.

526 Jäälinoja, Juho. Requirements implementation in embedded software development. 2004. 82 p. + app. 7 p.

527 Reiman, Teemu & Oedewald, Pia. Kunnossapidon organisaatiokulttuuri. Tapaustutkimus Olkiluodon ydinvoimalaitoksessa. 2004. 62 s. + liitt. 8 s.

528 Heikkinen, Veli. Tunable laser module for fibre optic communications. 2004. 172 p. + app. 11 p.

529 Aikio, Janne K. Extremely short external cavity (ESEC) laser devices. Wavelength tuning and related optical characteristics. 2004. 162 p.

530 FUSION Yearbook. Association Euratom-Tekes. Annual Report 2003. Ed. by Seppo Karttunen & Karin Rantamäki. 2004. 127 p. + app. 10 p.

531 Toivonen, Aki. Stress corrosion crack growth rate measurement in high temperature water using small precracked bend specimens. 2004. 206 p. + app. 9 p.

532 Moilanen, Pekka. Pneumatic servo-controlled material testing device capable of operating at high temperature water and irradiation conditions. 2004. 154 p.

534 Kallio, Päivi. Emergence of Wireless Services. Business Actors and their Roles in Networked Component-based Development. 2004. 118 p. + app. 71 p.

535 Komi-Sirviö, Seija. Development and Evaluation of Software Process Improvement Methods. 2004. 175 p. + app. 78 p.

537 Tillander, Kati. Utilisation of statistics to assess fire risks in buildings. 2004. 224 p. + app. 37 p.

538 Wallin, Arto. Secure auction for mobile agents. 2004. 102 p.

Tätä julkaisua myy	Denna publikation säljs av	This publication is available from
VTT TIETOPALVELU PL 2000 02044 VTT Puh. (09) 456 4404 Faksi (09) 456 4374	VTT INFORMATIONSTJÄNST PB 2000 02044 VTT Tel. (09) 456 4404 Fax (09) 456 4374	VTT INFORMATION SERVICE P.O.Box 2000 FIN–02044 VTT, Finland Phone internat. +358 9 456 4404 Fax +358 9 456 4374

ISBN 951–38–6394–8 (soft back ed.)
ISSN 1235–0621 (soft back ed.)

ISBN 951–38–6395–6 (URL: http://www.vtt.fi/inf/pdf/)
ISSN 1455–0849 (URL: http://www.vtt.fi/inf/pdf/)