MOBILE AGENT SYSTEMS AND TRUST, A COMBINED VIEW TOWARD SECURE SEA-OF-DATA APPLICATIONS

SUBMITTED TO UNIVERSITAT AUTÒNOMA DE BARCELONA
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

by Sergi Robles
Bellaterra, July 2002

Abstract

Agent technology is clearly an important enabler of new distributed applications. Mobile agents provide a further step in this direction and make possible new types of applications, such as sea-of-data applications or specific pervasive (ubiquitous) computing.
Nevertheless, the drawback of the new capabilities featuring this technology is the arising of new branches of security issues. It results hard to design security solutions for applications using mobile agents, especially in sea-of-data applications.
There is not a definitive platform in which implement these applications and still offering security and ease to program. We present the start of the development of MARISM-A, an **A**rchitecture for **M**obile **A**gents with **R**ecursive **I**tinerary and **S**ecure **M**igration. This platform intends to observe commonly accepted agent standards FIPA and MASIF, while providing flexibility to design secure sea-of-data applications.
We use trust to find out requirements of these new applications and present a novel model an methodology to achieve security solutions. We apply the model to some scenarios of a MARISM-A application. The same idea of using trust in sea-of-data applications can also be used to solve security issues in pervasive computing.

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Bellaterra, July 2002

Dr. Joan Borrell Viader
(Principal Adviser)

To M^{a}Carmen and my parents

Acknowledgements

I would like to thank my supervisor, Joan Borrell, for his invaluable support and encouragement. I am also grateful to John Bigham and Stefan Poslad for their valuable assistance and helpful discussions and for supporting my research stay at Queen Mary University of London.

I would like to thank other members of the MARISM-A project team, in particular Joan Ametller, Mercè Lazo and Ivan Luque, for their valuable contribution to the implementation of the MARISM-A platform.

I would also like to extend my gratitude to all the people of the Combinatorics and Digital Communication group for allowing me the opportunity to pursue the doctorate and complete this degree, and especially to Quim Borges for sharing his knowledge about graphs, and Joan Mir for his contributions to agent itinerary protection.

This work was supported by the Spanish Government Commission CICYT, through its grant TIC2000-0232-P4, and by the Catalan Government Department DURSI, with grant 2001SGR 00219.

Publications

Some of the work described in this dissertation has been published. Most relevant publications are [RLN+02, RB02b, RPBB01b, BHBR01, RPBB01a, BCH+00, BRSR99].

Contents

List of Figures

Chapter 1

Introduction

Trust in Allah, but tie your camel — Arabian proverb

Fortune cookie drawn few days before this writting.

This dissertation compiles the research I have done during the last 5 years in the Combinatorics and Digital Communication unit of the Computer Science Department. In the beginning of this period we analysed the just then born agent technology and how it could resolve some security related issues in voting applications, as well as other problems such as scalability or code distribution. We soon realised that behind the seeming solution for all problems, some deep hard serious security concerns were hidden, ignored by the main users of the technology at the moment, and awaiting to be addressed. After some time doing research to provide security to systems by using agents, we went on to the field of the security analysis within the agents communities themselves. This is how we run into trust as a social extension of security to face some human-like requirements of the agent world. Some of the key points in this research have came by chance: while analysing the voting scheme developed by my supervisor, during my research stay in the Electronic Engineering Department of the Queen Mary University of London, or applying agent technology to some other applications. Eventually, all these different stages have been connected (related) each other as if they were in the same plot from the very beginning. This thesis shows all relevant stages of this research interlaced on a sole continuous scheme. During the course of this research we have gone through many open problems, deserving each a full thesis

for its own. It has been difficult to crop all the branching derived from this research and hold on to a main trunk. In this process we have always given more priority to the more applicable lines. All these open lines are referenced along the chapters of this dissertation.

It seems sensible to start this work by wondering about one of its main grounds: security. Security has been a big issue since the beginning of practical usage of computational systems and the first computer networks. After more than 25 years of research and engineering, network applications are still at risk: security is still one of the most difficult issues to address while designing systems. Why is the security problem not solved yet? There are no simple answers to this situation. Addressing the problem requires understanding the dynamic nature of security and appreciating the need for a scientific approach. There is no direct way of facing the problem of security, and design of security solutions is nowadays based on risk management. This is a familiar process in the financial and insurance communities, but not well understood within the computer science community. In fact, risk management as practised today, and so security design, involves more art than science. Moreover, security solutions found in this way often result very specific. The analysis is customised to the system and so is the solution. It is difficult to reuse these security protections except from trivial safeguards. In many complex systems it is extremely hard to resolve the security issue by providing an scalable and flexible solution and it often results more expensive than paying damages. As a matter of fact, the trade-off between cost and security generally makes more profitable having systems with many security flaws.

Some factors, including last trends, have recently made even more difficult to solve security. Firstly, we have the Internet that has been increasingly becoming an important channel for application intra-communication and has progressively been introduced in heterogeneous environments. This high connectivity has seriously changed the risk environment. The use of complex interactions between multiple parties that play multiple roles and entail transactions in multiple domains is a common feature in new applications that benefit from Internet advantages. These interactions are often between parties with opposite interests, which is an added difficulty. Openness is also a determining factor complicating the security design. Another factor contributing to the complexity of security design is the high socialisation included in recent software designs such as in the e-business or data-oriented applications. Most of the times, this socialisation is achieved though the use of distributed

architectures such as agent technology, where software objects autonomously act on behalf of a principal. This has been a natural adaptation to the situation. Agent technology has made more evident the need for interactions between autonomous domains. At the same time, it is clearly an important enabler to underpin and to help automate all these interactions. Agent technology is establishing as the dominant paradigm for the implementation of complex systems.

Agents make feasible the easy implementation of a wider variety of applications, hard to develop by using classic paradigms. Agents are also an enabler of new types of applications, standing out the so called sea-of-data applications. In these applications a massive amount (sea) of information is distributed among internetworked elements and must be accessed from a remote place to provide an outcome. A program can not just fetch the information and process it locally because several reasons, including the ratio volume of information / available bandwidth, the way this information is stored (it might be acquired by a special device), or legal reasons (medical images in hospitals, for example). Agent technology is the basis to make these applications come true. The price for this is a new branch of open issues concerning security. This is common in agent technology, but especially tricky on mobile agent systems, where agents are not static and can migrate from one environment to another to continue their execution. The difficulty of finding security solutions has been considerably increased when agents have been endowed with this mobility.

Several new security requirements emerge from this new scenario [WSB00]. The classic concept of security to support perfect and reliable protection of systems, via authentication, authorisation and confidentiality is difficult to achieve in these new scenarios. Solutions based on global infrastructures are not valid here and it is necessary to find new paradigms that fit into the new approach. The assumption that a single security mechanism such as public key cryptography provides security must be given up and accept a more complex, multidimensional and dynamic natured security.

Simultaneously, similar security requirements are foreseen on a new scenario. Technology is moving beyond the personal computer to everyday devices with embedded technology and connectivity. Pervasive computing (also called ubiquitous computing) is the idea

that almost any device, from appliances to cars or homes, can be connected to an huge network of other devices. The goal of pervasive computing is to create an environment where the connectivity of devices is embedded in such a way that the connectivity is unobtrusive and always available.

Actual security mechanisms address the symptoms, no the root cause, of most security problems. In this dissertation we introduce an alternative approach to solve the issue of security in multi-agent systems that can be applied to sea-of-data applications and pervasive computing while observing the basic requirements of openness and generalisation. This approach is based on trust.

Trust, used in computational systems, is usually related to many concepts without a specific word in our language and unmistakable connected with a social dimension of security. This situation leads us to an overuse of the term which makes it difficult to split all different semantic domains referenced in the bibliography and related to this topic. Thus, we might find trust articles about reputation systems or trust articles about certification authorities in public key infrastructures. The origin of this situation partially comes from the nonexistence of a common definition for this word. Probably, this lack of an agreed definition for trust comes from the absence of a generic form suiting and characterising trust requirements from all domains. Another reason for these discrepancies seems to come from the different main interests of the security and the artificial intelligence research communities. Whilst the former focus on control aspects of trust (formal logics for cryptographic analysis of protocols [BAN90], [Gam90], users and certification authorities relationships in public key infrastructures [AR97], [BBK94], [Jøs96], [Mau96], [YKB93], or distributed security architectures [EFL+99]), the latter do it in more social aspects and used in artificial intelligence systems (reputation mechanisms [Mar94], [ARH00]).

How trust can be used to provide security in those special scenarios? The relationship between trust and security is a key topic when referring to trust. If we take common accepted definitions for trust (Oxford English Dictionary, Webster English Dictionary), security is usually defined as the quality of being out of peril, risk or fail, that can be absolutely trusted. We can then stand that increasing the security of a system is equivalent to increase the trust on it for peril, risk or fail absence. An absolutely secure system is that in which we have total trust, or absolute certainty. Why we use this trust for instead of going on using

security if they are equivalent words? In fact, security and trust are not neither the same nor equivalent. Whilst security is a quality (property) of the system, trust is conceived as a relationship representing the perception of some principal about other or something. Moreover, trust can be transmitted to other parties as a mutual relationship. Using trust instead of security will allow us to use different mechanisms, from different nature, to obtain more secure systems not *per se* but for the members of the system and for its users (actors). This is specially useful for systems based on mobile agents and pervasive computing.

With this new concept of trust we can define a plane of trust in which it is easy to represent the relationships between all members of the system. The first step toward this new approach is the definition of a model for trust. This is not an easy problem to solve. Within this plane of trust it is easy to show the real trust requirements of the system, but we must also provide a methodology to allow us to choose and combine different trust solutions and therefore find a security solution..

The transformation from the security plane to the trust plane allows to screen the relevant issues concerning security from the not so important in the specific scenario of mobile agent applications. In this trust plane the solutions are quite more likely to be found. The transformation back to the security plane will be trivial since the trust solutions directly provide security (are security solutions) by definition. By using this technique, found solutions will fulfil scalability, distribution and integration requirements within an unique and holistic security/trust scheme.

Objectives

The first objective of this dissertation is to analyse the problem of security in mobile agent environments and to contribute with mechanisms to solve particular aspects of it in agent based applications.

A second objective is to start the implementation of Multi Agent System platform in which it is easy to add security mechanisms. It must be as flexible as possible and compliant with existing standards. Security requirements of this platform must be outlined using traditional methods. The platform has to be prepared to be used as a testbed for secure sea-of-data applications.

Trust is foreseen as an alternative way to achieve security in mobile agent applications.

Another objective of this dissertation is to characterise the concept of trust by finding out its basic elements. This characterisation of trust must take into account different points of view.

The last objective is to provide the definition of a model for trust to find out trust/security requirements in mobile agent applications and apply it later to solve security issues in sea-of-data applications. This work is not trying to find a generic definition for modelling trust in a deductive fashion and specifically defining applicable trust solutions afterwards. This is an open problem that requires more research and evaluation and is out of the boundaries of this dissertation. Rather, we seek to focus on an inductive approach of the model based on elemental components of trust. This model must allow to be used to achieve security solutions in mobile agents systems through a methodology.

Outline of Dissertation

The rest of the dissertation is structured as follows.

Chapter 2 reviews agent technology, providing basic descriptions of elements in this technology and highlighting security issues. Trust is also introduced in this chapter

Chapter 3 is devoted to the MARISM-A Mobile Agent System. This chapter talks about the features and implementation of this system and analyses security requirements.

Chapter 4 shows how trust can be used to build security solutions to mobile agent applications. In this chapter we present our trust model and a methodology to be applied. Some scenarios from MARISM-A system, described in chapter 3, are used both to show the application of this methodology and provide security solutions.

Chapter 5 describes some mechanisms we have developed regarding security or fault tolerance in multi-agent systems. Some of the ideas discussed could be used in the platform described in chapter 3 or be part of the mechanisms repository mentioned on chapter 4.

Chapter 6 concludes by showing how the research goals have been met, and our future research road-map.

Chapter 2

Agent technology and trust

Mobile agents provide a new programming paradigm and a totally new scenario to develop complex applications. In some cases, this technology is one of the few available (often the only) to implement applications with special requirements, such as sea-of-data applications. In these applications, huge amounts of distributed information process is required to have some outcomes. Network bandwidth limitation, data acquisition mechanisms, or client weak connection (intermittent and not guaranteed), lead to the necessity of using mobile agents. New security requirements emerge from this new scenario. Some of them are inherent in agent nature, many are due to mobility and other are specific to sea-of-data applications. The first part of this chapter is devoted to review agent technology, especially focusing on security related issues.

Classic concept of security to support perfect and reliable protection of systems is based on the traditional taxonomy of security threats (confidentiality, integrity and availability). It is difficult to study the protection issues in these new scenarios. Common security solutions are unworkable due to the aforementioned restrictions (they rely on on-line connectivity to central servers). Traditional solutions are not the definitive solution here and therefore it is necessary to find new paradigms that fit into the new approach. An alternative is to benefit from the high socialisation of agent systems and introduce trust as a more suitable way to achieve security.

First part of this chapter shows a review about agent technology we did in 1999. We have kept tracking on most security approaches since then and have updated the review

with relevant information. Second part of the chapter introduces the concept of trust.

2.1 Agents

An agent is a software object on an execution environment that is acting on behalf of a person or organisation [JW98]. The agent is reactive (it is sensible to the environment and acts depending on its changes); autonomous (it has its own thread of execution and therefore it can have initiative to execute tasks and control its own actions); proactive (its behaviour is objective oriented) and it is continuous in time (it is constantly under execution). Furthermore, an agent can be communicative if it can interchange messages with other parties, learning if it is able to adapt using previous experience, or mobile if it can move (migrate) from an execution environment to another [Whi95].

A Multi Agent System is a software environment in which software agents run. It provides support for software agents to execute, to manage their execution, to access system resources, and to guarantee integrity and protection of agents and the platform itself. Multi Agent Systems are implemented through agent platforms, the underlying software for the system. The platform defines a standard around which a system can be developed. Once the platform has been defined, software developers can produce appropriate agent applications. Platforms may provide support for migration, naming, location and communication services. The system consists of one or more execution environments or agencies of the system. Some authors go further and define places, independent execution environments within agencies.

The are a large number of agent platforms. Many of them are still in a development stage, and others are ready to be evaluated. Examples of agent platforms are listed in figure 2.1.

2.1.1 Mobility

An agent may stay at just one execution environment, communicating (locally or remotely) with other agents, and not being able to leave the agency. These are stationary agents. In the other hand we have mobile agents, not limited to only one execution environment.

Name	Language	Author
Ajanta	Java	U. of Minnesota, USA
ARA	Tcl, C, Java	U. of Kaiserslautern, Germany
AgentTcl	Tcl	Dartmouth College, USA
Aglets	Java	IBM, Japan
Concordia	Java	Mitsubishi, USA
CyberAgents	Java	FTP Software, Inc., USA
ffMAIN	Tcl, Perl, Java	U. of Frankfurt, Germany
FIPA-OS	Java	emorphia, UK
Grasshopper	Java	IKV, USA
Java-2-go	Java	U. California Berkeley, USA
Kafka	Java	Fujitsu, Japan
Messengers	M0	U. of Zurich, Switzerland
MOA	Java	The Open Group, USA
Mole	Java	U. Stuttgart, Germany
MonJa	Java	Mitsubishi, Japan
Odyssey	Java	General Magic, USA
SOMA	Java	U. of Bologna, Italy
Tacoma	Tcl, C, Python,...	Cornell, USA i Tromsø, Norway
Telescript	Telescript	General Magic, USA
Voyager	Java	ObjectSpace Inc., USA

Figure 2.1: Mobile Agent Systems

Mobile agents can move between internetworked agencies transporting their code, data and state information to continue their execution on different environments. There are several methods to achieve this mobility. We will see these different mobility schemes in this section.

Mobile agents provide many advantages. First, they reduce network overloading allowing the code to be executed at data location, instead of fetching remote data from code's emplacement. Also, network latency is decreased. Agents can be used to execute (act) locally where the control is required, reducing latency time in real-time applications. Mobile agents provide an asynchronous and autonomous execution which is ideal to work with in environments with expensive or fragile network links. Heterogeneity is a natural feature inherent in mobile agents. They are hardware and transport layer independent and hence

they provide optimal conditions for an uniform system integration. Robustness and fault tolerance are two more advantages easily provided by mobile agents.

Mobile agents provide a new network programming paradigm [GV97] to be add to the traditional client/server paradigm, remote code invocation and code on demand. In the client/server model, a server offers a service usually related to some resource access. The code implementing the service is in the server side and it is executed there. In this model execution environment, code and resources are all in the server side. The client trying to access the resource communicates with the server providing it using a pre-established protocol. The remote code invocation model works slightly different. The code is still at server side, but it is used from the execution environment at client side. The client does not use an specific protocol as the paradigm itself resolves this communication. In the code on demand model client demand the code to execute locally. Mobile agents are an evolution of these paradigms: the code is now going to the data and only results go back to client side. The key aspect of the new mobile agents model lies in the distribution of code, resources and execution environments among a set of internetworked computers. Mobile agents are indeed a specific case of a wider concept which is out of the scope of this work: code mobility. *Java Applets* and *ActiveX* are examples of code mobility. In these cases is the destination itself who starts the transfer of the code.

There are several types of mobility: remote execution, weak migration and strong migration. These types are shown on figure 2.2.

Transport of Code and Data	Migration of Code and Data	Migration of Code, Data and State of Execution
→ Remote Execution	→ Weak Migration	→ Strong Migration

\- ———- Mobility degree ———— +

Figure 2.2: Types of mobility

In the case of remote execution, the agent is sent before it starts to be executed. When it arrives at destination it is executed until it finishes. In this case the agent is transfered once.

When it is executing it can use the same remote execution mechanism to start the execution of other agents. In the remote execution the destination of the agent is determined by the execution starter.

The agent can do a weak migration by sending its data along with its code. Usually, the implementations of this scheme allow to choose which part of data will be transferred to the new location of the agent. In this case, the agent programmer might design some mechanism based on the value of agent's data to resume the execution from some point.

Strong migration is the highest degree of mobility. Using this scheme not only agent code and data is sent, but also the state of execution. When the agent arrives to destination, it is fully restored and its execution is resumed from the same execution point it was just before migration. Strong migration turns to be complex, since it involves low level internal mechanisms for execution restoring that must be standard to provide migration transparency in heterogeneous environments.

In [GH99] we found some considerations about agent migration. In this very section we noticed the reduction of network loading as an advantage of using agents: the agent goes to the data and brings back the results instead of fetching a big volume of information and process it locally. Nevertheless, we must analyse what happens if the agent itself is very big. If applications using agent technology had big agents with lots of code inside, results would not be as good as expected. The basic idea in [GH99] is a programming model called FOAM (*Fragmented Object Agent Model*) based on the object model of AspectIX architecture and with a communication layer, CORBA compliant, beneath. Using FOAM an agent does not have to be completely transferred during migration, but only the required fragment to fulfil the required tasks at the destination agency. The approach is further detailed in [GH00].

Agent itineraries

A basic element on mobile agents is the itinerary. The itinerary is the relation of agencies that an agent must visit to achieve its objectives. This itinerary allows to have a flexible specification of the travel plan of an agent. This information may be implicit, in the code of the agent, or explicit, as a separated element.

Explicit itineraries are structures containing the agencies that are going to be visited by

the agent on its life cycle. This concept was first used in the Concordia system [WPW$^+$97] and improved in [SRM98]. [MB02] defines a flexible itinerary describing agent destinations and the tasks to accomplished at each one.

Itineraries are composed by basic structures, or entities, which represent an agency, or host, and the code and data to be used by the agent in that agency. The form of each of these entities e_i is formed by a tuple

$$e_i = (host_i, data_i \ + \ methods_i).$$

Entities can be arranged in sequences, sets and alternatives. These structures can be recursively combined to build complex itineraries. In a sequence, the agent will migrate to each agency one after the other. Formally, it is a list $[e_1, \cdots, e_n]$ with $n \geq 1$ entries defining that the nodes specified by entry e_i ($1 \leq i < n$) must have been visited before the nodes of entry e_{i+1} are visited. In a set, a group of agencies will be visited by the agent in no special order, i.e. a list of entries $\{e_1, \cdots, e_n\}$ with $n > 1$ specifying that the elements $e_1, \cdots, e_n$ are handled in any order as long as each element is visited exactly once. Finally, in an alternative only one agency of those listed will be visited by an agent, depending on some conditions. That is to say, a list $< e_1, \cdots, e_n >$ with $n > 1$, in which one entry (usually chosen by the mobile agent) is visited from the whole set of entries. Note that set structures involve cloning of agents. If cloning is not desired then a set structure can be replaced by an alternative with all possible sub-itineraries (permutations of entities) as sequences randomly chosen:

$$\{e_1, \cdots, e_n\} \equiv < [e_1, e_2, e_3, \cdots, e_n], [e_2, e_1, e_3 \cdots, e_n], \cdots, [e_n, e_{n-1}, \cdots, e_1] >$$

In the following, we show the traditional example of mobile agent itinerary [SRM98] to clarify this definition. The owner of the mobile agent is planning a romantic evening and orders his agent to buy some flowers, a ticket to the theatre, and to reserve a table in a restaurant. The mobile agent can choose between two theatres, and depending on the decision it has to reserve a specific restaurant near the theatre. The sequence specifies that the mobile agent has to go first to the theatre to buy the ticket and go afterwards to the restaurant. The main set structure of the itinerary allow the agent to buy flowers either before or after restaurant reservation:

$$
\begin{aligned}
I = \ & \{ \quad (BestFlowers, buyFlowers), \\
< \ & [\quad (CentralTheatre, buyTicket), (KingsInn, reserveTable)], \\
& [\quad (ModernArts, buyTicket), (BeefHouse, reserveTable)] > \}
\end{aligned}
$$

Mobile Agents Systems

Mobile agents paradigm is only a set of basic ideas about how software could be developed. We need some specifications and implementations to bring this programming model to practise and get profit from its features. A mobile agent system consists of a specification of an execution environment, an agent migration mechanism and an agent structure specification. Furthermore, the resulting system can be endowed with other mechanisms to achieve security or reliability.

2.1.2 Standards for Multi Agent Systems

As we have seen, there is not just one way to implement an agent system. Sometimes there are problems with no obvious solution, such as the migration of multi threaded java agents when there are more than one thread running during the migration. All multi agent systems differ widely in architecture and implementation, thereby impeding interoperability and rapid deployment of applications running in heterogeneous systems. To promote interoperability, some aspects of agent technology need to be standardised.

There has been some initiatives to achieve a global standard for multi agent systems. As a result of this, there are currently two standards for agent technology: The Object Management Group's Mobile Agent System Interoperability Facility (MASIF) [MBB+98] and the specifications published by the Foundation for Intelligent Physical Agents (FIPA) [FIP00].

MASIF

In 1995 the OMG started working on a standard, called Mobile Agent Facility (MAF), in order to promote interoperability among agent platforms. In 1997, a joint submission by

IBM, General Magic, The Open Group, GMD FOKUS, and etc was presented to the OMG. And the standard's name was changed from MAF to Mobile Agent System Interoperability Facility (MASIF) [MBB+98]. In 1998 this specification was accepted as an OMG standard. The current edition was issued in 2000. IKV's Grasshopper agent system [IKV00] (version 2 available in 2000) observer the MASIF standard. It is a mobile agent and runtime platform developed in Java, built upon a distributed object-oriented middleware. As submitters of the MASIF standard, IBM and GMD FOKUS agreed to implement the MASIF spec within their own agent platforms. (GMD stands for German National Research Center for Information Technology, FOKUS stands for Research Institute for Open Communication Systems.) There is another mobile agent system called Secure and Open Mobile Agent (SOMA) System [BCS99], developed by Universitá di Bologna in Italy. SOMA has been developed by "closely considering compliance with MASIF."

This standard provides interfaces between agents and agent systems. MASIF is limited to the inter-working of agent systems using the same language. It does not standard local agent operations, as agent interpretation, serialisation or execution. It basically concerns these five topics:

- **Agent Management.** This comprises agent creation, suspension, resumption or finalisation.
- **Agent Migration.** Agents can move from agency to agency within a common infrastructure.
- **Agent Tracking.** Agencies, places and agents are registered in a region registration component.
- **Agent and Agency Naming.** It standards the syntax and semantics of the names of agents and agencies, enabling identification.
- **Agent Type and Location Syntax.** Agent migration can not be done if the agency does not support the agent type. The location is standardised in order to enable to localisation.

MASIF has a kernel consisting of two components (at the moment just interfaces): `MAFAgentSystem`, defining the operations for creating, managing and transferring agents, and `MAFFinder`, providing operations to register, unregister and locate agents and agencies.

MASIF relies on CORBA (Common Object Request Broker Architecture) services. This makes MASIF much simpler: for instance, it does not specify the remote interaction between agents and agencies due they are CORBA objects. MASIF does not address the agent communication aspect.

FIPA

The Foundation for Intelligent Physical Agents (FIPA) was formed in 1996 to produce software standards for heterogeneous and interacting agents and agent-based systems. It is a non-profit association formed under Swiss law. Its members include companies and universities.

FIPA identified a list of agent technologies deemed to be specifiable in 1997 and standardisation work started. There is a set of spec called FIPA 97 and another called FIPA 98, both are now on Obsolete Status. The current spec is FIPA 2000, half of which is in the Preliminary Status, another half on Experimental Status.

About 16 FIPA-compliant platforms have been implemented by diverse companies, some of which are freely accessible under open source code, such as FIPA-OS. FIPA-OS is an agent runtime and development platform using Java.

In addition to agent management, the FIPA spec cover agent communication and agent message transport that MASIF does not cover. FIPA developed an Agent Communication Language (ACL). It is a high level language with a precise syntax and semantics allowing agents to communicate. An agent message consists of two parts. The envelope conveys information for the transportation. The message body contains the actual message. The FIPA ACL is based on the Speech Act Theory: message are actions, or "communicative acts", as they are intended to perform a specific action by virtue of being sent. FIPA ACL derives from Knowledge Query and Manipulation Language (KQML).

Agentcities [AC01] is a new initiative aiming to build a worldwide, publicly accessible, test bed for the deployment of FIPA Agent based services. Aims of Agentcities include

creating a common resource for developers, providing a benchmark environment to validate and test compliance and acting as a focus for discussion.

Comparison of MASIF and FIPA

MASIF is based on agent platforms and it enables agents to migrate from one platform to another. FIPA spec is based on remote communication services. The former is primarily based on mobile agents travelling among agent system via CORBA interfaces and does not address inter-agent communication. The latter focus on intelligent agent communication via content languages and do not say much about mobility (but see below about FIPA 2000). FIPA adopts an agent communication paradigm, which can better express the nature of cooperation and is more suitable for integration with other AI technologies. MASIF adopts a mobile agent paradigm which is more appropriate in situations where dynamic and autonomous swapping, replacement, modification, and updating of application components are required.

The are some attempts to bring FIPA and MASIF together. A FIPA 2000 spec deals with the mobility aspect of agents. It tries to integrate FIPA and MASIF. An Annex of the ACTS (Advanced Communications, Technologies and Services - an European research collaborative program) baseline document analyses the possibilities of MASIF/FIPA integration. For the moment, Grasshopper-2 has been available since 2000 and is both MASIF and FIPA compliant, through a MASIF and a FIPA add-on respectively.

2.1.3 Security in Mobile Agent Systems

Mobile agents clearly enhance the potential of static distributed systems allowing programs to be executed in environments that are normally not administrated by their author. Obviously, some guarantees are needed to assure that none of the parties involved in the execution will make any damage [Hoh98a]. We all bear in mind the threat of virii, worms or Trojan horses. The owner of an execution environment does not know the nature of an incoming program and this also happens with mobile agents. Security aspects in mobile agents must be well understood and protected with proper mechanisms.

Nowadays there are some mechanisms to get some degree of protection. Unfortunately,

a global solution for the problem of security in mobile agents is not available yet. Many aspects are still open and under research.

The security issue in mobile agents can be classified into three strands: between agents, between agencies, and between agents and agencies. There exists mechanisms to prevent attacks in the first two, since the problem is similar to traditional distributed systems. The problem of security between agents and agencies is not symmetrical. Protecting agencies from agents is also a known and solved problem. There are many mechanism providing this security, such as the *sandbox* used in the Java architecture [Gon97]. On the other hand, protect agents from malicious agencies is not a trivial issue.

Protecting programs, or agents, from their execution environments, or agencies, is very complicated. The agency controls every step in the execution of the agent: it can read every byte of code, every data and state, even interfere with the executed algorithm since the agency itself is interpreting the code. Possible attacks include break privacy (private keys, electronic cash), code manipulation (virus inclusion), and denial of service attacks. This is an essential issue to use mobile agents in open systems, normally referred as the malicious hosts problem. In [Hoh98a] and [Rob99] some of the attacks performed by a malicious host are identified. These attacks are listed below:

1. -Code spying
2. -Data spying
3. -Flow control spying
4. -Code manipulation
5. -Data manipulation
6. -Flow control manipulation
7. -Denial of execution
8. -Agency impersonation
9. -Communication eavesdropping
10. -Communication manipulation
11. -False system calls return values
12. -Kill an agent
13. -Re-execution of agents
14. -Itinerary manipulation

The agency must read the code and data of the agent in order to execute it. Since the code is unveiled the agency knows the strategy of execution (problems 1, 2 and 3) and could manipulate all this information (4, 5 and 6), provide false results (11) or even deny execution (7). This is a critical problem should the agent carries private cryptographic keys or electronic cash, and does not admit an easy solution. Agencies might intercept agents and execute them, doing an impersonation of the real destination agency and therefore

deceiving agent (8). An agency could spy and manipulate messages sent between agents (9, 10). Cloning agents or killing them (13, 12) may have important consequences in e-business applications. Manipulate itinerary (14) may be critical on some applications, for example in electronic voting [BRSR99].

We consider interesting the approach introduced in [DM01] because it lets to analyse the problem from an external and more general point of view. [DM01] set forth many security problems arising from the use of mobile agents through an analogy with a real-world complex scenario, the Byzantine Princes Problem (BPP). It provides a methodology consisting in finding out resemblances between the problem and the mobile agent application to better understand security issues. In the set up of the scenario there is an old king with a large fortune, married twice during his live and with two sons for each wive. Each of the four princes have his own principality, run ruthlessly. The princes do not dare leave their respective castles, for fear of being killed by their own subjects. The old king is dying, and plans to distribute his wealth as an inheritance to the winners of a chess game. Siblings of same mother will play together, alternating who makes the move each side. Since officials can be killed, detained or bribed, the king must devise mechanisms/procedures that are built into the chess playing process that allow the game to continue in the face of these unsavoury activities. If cheating is discovered, the game is continued from the point of the last legitimate move.

Considering that the princes will not leave their castles, the king will need to send one or more caravans and perhaps messengers through the principalities to have the game played. The people in the caravan(s) and the messengers may change due to a number of reasons such as death or retirement. Because of this, recognition of caravans' personnel or the messengers alone cannot be used as authentication that they represent the official party. Other credentials are needed to verify authenticity.

This is the non-exhaustive ([DM01]) list of the possible misdeeds that could occur:

1. A caravan may be destroyed while in transit.

2. The board position carried by a caravan could be changed while in transit.

3. The caravan personnel may be exchanged with people with a different mission, including ones who wish to misuse the access rights of the caravan, or want to modify

the legitimate outcome of the game.

4. The caravan personnel could be bribed to act improperly.
5. Imitators of the caravan may arrive at a castle.
6. A prince may have the caravan destroyed while it is at his castle.
7. A prince may attempt to change the board position before his move.
8. A prince may attempt to make an illegal move.
9. A prince may attempt to never make his move.
10. A prince could send a imitation caravan ahead to see what the next round of moves would be, and then give his illegally wiser chess move to the real caravan, and send it on.
11. The caravan could mistakenly record a prince's move.
12. Imitation castles may be set up to deceive the caravan into visiting the wrong prince, or someone who isn't a prince at all.
13. A prince could send out a duplicate caravan as well as the real one, so that multiple games are being played. At a later move, the prince or his brother may destroy the caravan with the less favourable chess position.
14. A prince may falsely claim that an illegal move has been made to try to force a number of moves to be taken back.
15. A prince's subjects may lose confidence in the prince if his ineptness were revealed by disclosure of the board's status.
16. The caravan personnel harm the prince.

Nevertheless, we have noted that the Byzantine Princes Problem does not suit all type of mobile agent applications. New issues not addressed in this list arise, for instance, in the case of sea-of-data applications. All this problems will be analysed deeper in chapter

3 in the framework of a Mobile Agent System. In that chapter we present a novel problem based on BPP which addresses more issues found in general mobile agents applications.

2.1.4 Solutions

There are many solutions to specific security problems in mobile agent systems. Other issues are so hard that some authors argue that it is impossible to find solutions. This is the case of the malicious agency problem [HCK95]. However there are some approaches that partially solve the problem. An added difficulty to find solutions to the malicious agency problem is that it is not enough to show a mechanism solving it, but a formal demonstration or verification must be provided to guarantee that it is a good solution.

Solutions can be classified into three categories: organisational approaches (avoiding security problems assuming a closed environment), specific problem solutions (claiming to solve some specific problems), and holistic approaches (trying to protect from any attack).

Organisational approaches

All proposals using an infrastructure of agencies controlled by one operator or allowing migration only to restricted agencies belong to this category. The result is a restricted mobile agent system, not fitting all applications. The main drawback in this type of solutions is this loss of openness and consequent general purpose. An example is the Telescript system [Whi94], which gets rid of some security problems only allowing the execution of agents that come from trusted agencies. [Che97] contributes with a security infrastructure to authenticate agents and owners using cryptography on a centralised infrastructure.

Specific problem solutions

In this category there are some proposals that try to prevent some specific attacks. Most of them provide mechanisms for detecting data and code manipulation [Vig98],[Yee97], [NL96] or agent re-execution [SRM98].

In [Vig98], the owner of an agent can detect and prove modification attacks *a posteriori*. No other attacks are prevented and it is assumed a legal framework for liability when

damage is done. This framework might not exist in an open agent system, without a central organisation. [Yee97] offers a view of other partial solutions, such as fault tolerance techniques. An interesting method is found in [NL96], the Proof Carrying Code, which protects data and code through the validation of proves.

[SRM98] proposes two approaches to solve two problems: agent re-execution avoidance and agent persistence. No re-execution is needed when an agent's owner has to be certain that the agent will be run exactly once. For instance, if the agent carries out commercial transactions. To avoid re-execution [SRM98] uses a protocol based on message transactional queues. Persistence is provided by using several agencies to run an agent. A similar solution for the re-execution problem is in [Bae98]. The proposed protocol detects clone agents by using a central coordination element.

Some solutions have been developed to protect itineraries. In [BRSR99] we use a non-repudiation protocol to protect the steps of the itinerary. [KAG98] bases the protection on digital signature chaining and [Rot98] needs cooperating cloned agents to guarantee itinerary integrity. [SRM98] and [WSB98] have also addressed this problem by using a fault-tolerance agent system and by adding a tamper-proof hardware to each host of the mobile agent system.

Holistic approaches

There are proposals that try to protect the agent from any attack coming from the agency. Some of these solutions do not use any specific strategy for security since they assume some special tamper-proof hardware beneath, such as *Citadel* [Pal94], a cryptographic co-processor. This hardware performs all crytographic operations and has to be installed on all agencies, which is a very restrictive requirement.

Other proposals achieve this protection without any special hardware [Hoh98b], [ST97], [ST98], but not solving the problem of re-execution and persistence.

[ST97] protects the agent by using mobile cryptography. This method is based on the execution of special functions that operate encrypted data to obtain encrypted results. Data can not be manipulated by the agency because it does not know the secret needed to decrypt the information. The problem is that, by the time being, only polynomial and rational functions can be used. This is a strong limitation to program arbitrary agents. [Hoh98a]

questions if this model is resistant to all attacks because a finished algorithm to make agents was never provided. In [ST98] there is a complete description of this approach.

[Hoh98b] proposes a different scheme, applicable to any existent agent. The method is based on a time limited agent protection against attacker analysis and hence against possible attacks. In this approach an agent is a black box if, during a time interval, code and data are not modifiable. This black box has a use by date. To convert an agent to a black box code obfuscation mechanisms or cryptography are used. An attack to this approach is to analyse output depending on the input. An agency can deduce the code by analysing the output of a group of clone agents executed at the same time with an specially designed input. [HR99] avoids this attack using a trusted agency to act as a register. Communication between agent and agency must be secret and authentic. This solution depends on the calculated time of execution of an agent. Since this time is not a constant for all platforms the technique is not very accurate.

It seems difficult to find some approach to solve all security issues in mobile agents applications at the same time. The variety of application domains tend to make authors design tailored solution not valid everywhere. Instead of the search of this panacea, efforts must focus on finding out new strategies to face the problem on a more sensible fashion. Our contribution into this area is based on trust modelling. Trust provides a new prospect to face this problem. Security requirements can be found from trust relationships between entities of the system (agencies, agents, users, etc.). This approach is shown on chapter 4, but before analysing it we need to understand trust.

2.2 Trust

One of the main problems regarding trust is that there is not a consensus about its definition. Before using it as the basis for a model to solve the problem of security in mobile agent systems, we must have an approximation of trust.

The lack of existence of a common definition for trust and the lack of words to define specific aspects of trust make it harder to talk accurately about it. This situation leads to an overuse of the term which makes it difficult to split all different semantic domains referenced in the bibliography and related to this topic. This lack of a widely agreed definition

for trust possibly indicates that there is no single generic form of trust that fits and characterises trust requirements in all domain applications. This is partly why we focus on an inductive approach and on a component based trust model rather than on a more deductive approach of working from a generic trust definition through formalisms and only then to define application specific trust solutions. Both inductive and deductive approaches are useful and complementary and both need further research and evaluation.

There exists a lack of coherence among researchers in their definitions of trust. We might find trust articles about reputation systems or trust articles about certification authorities in public key infrastructures. One of the reasons for this is the different view points on trust derived from the primary interests is the security and artificial intelligence communities. We distinguish carefully between these two approaches. The computer science or network security approach largely focuses on a control view of trust, whereas the social or AI view largely focuses on a more social view of trust.

Trust in the field of computer science security is often not explicitly defined. It is usually related to many concepts without a corresponding word in our language and unmistakable connected with a social dimension of security. A considerable body of work concerning trust in this field is in the area of security. These are mainly in the form of formal logics to analyse cryptographic protocols for design flaws and correctness [BAN90] [Gam90], or to define relationships among users and certificate authorities in public key infrastructures [AR97] [BBK94] [Jøs96] [Mau96] [YKB93]. Trust mechanisms are also a key requirement for mobile agent migration [WSB00], for example, trust is defined as a policy, that is, a set of rules prescribing a principal's behaviour for all relevant situations. Trust in computer science mainly concerns simple public key infrastructures or distributed security infrastructures [EFL+99].

Other work on trust in the field of computer science security is associated with being able to authenticate the identity of someone and being able to control the access that someone has to some resource such as information. If you have a distributed framework or web to control the authentication of someone and the authorisation of someone to access something, then this framework can be trusted, it is a trusted third-party and you can trust that the results of applying the framework, i.e., you can trust that someone has the authority to do something and that they are authenticated [EFL+99]. If you can control authorisation

and authentication, then you can entrust a second party, i.e., confide in them some information or charge them to do something. Trust in a second party frequently requires trust in a third party [MvOS96]. More exactly, entrusting a second party requires entrusting a third party, or rather a network of third-parties, to provide security such as authentication, authorisation, privacy, information integrity and non-repudiation.

Other trust models are based on real world, social properties of trust, founded on work from the social sciences [Mar94], [ARH00]. This trust is mainly used in artificial intelligent systems. These models use are complex, based on metrics and empirical rules and are mainly oriented to reputation mechanisms. The AI or social aspect of trust considers the externally observable behaviour of a second party from the viewpoint of the first party and the society in which they both operate. Various numerical models are used to classify and accrue the observations about a party from one or more other parties. A first party can then use a society view and personal view of the observations of the second party to evaluate whether or not it should interact with the second party and what the risk is entrusting the second party with specific actions or information.

A trust model spanning both low-level security aspects and high-level concepts does not exist. Moreover, work in other trust related concepts such as liability or fairness is very dispersed.

It is easy enough to make broad generalisations about trust, but in fact this issue is extremely complex. Even the most superficial look at this issue raises fundamental questions about its definition. Trust is an elusive notion that is hard to define because it has several facets and can be a matter of subjective interpretation. Trust is multi-dimensional and definitions are domain specific. We have analysed different trust dimensions and have discussed their dependences. Dimensions of trust include, but are not limited to:

Direct / Indirect

A clear dimension of trust is concerned with its direct/indirect character. The trust an actor is directly confiding to an element of the system is called direct trust or immediate trust. This type of trust allows an agent to access its own resources, or transmit information through an environment. That is to say, is the trust used when entrusting on something. Issues with direct trust in big and complex systems are well known. However, this type

of trust is not enough to face scalability needs in new systems (take the case of pervasive computing, for example).

The rest of the trust is indirect trust, since there are other elements involved. Indirect trust represents the social aspect of trust. We need indirect trust when systems start expanding becoming a huge interconnected network of components (it is not feasible to have a direct trust relationship among all components), and especially when autonomous agents are involved. Indirect trust is only required between actors of the system and it provides the needed mechanisms for a global, scalable and distributed security solution. We are especially focused in this indirect trust because our concern in novel applications using mobile agent technology.

Controlled / Not Controlled

We have controlled trust when there exists established entrusted mechanisms to control actions done by all actors of a system. Any actor can verify these mechanisms and the infrastructure comprises all involved parties. Because this type of trust is fair for all actors as it is assured by a common entrusted mechanism, this trust it utterly objective. Delegation and derived responsibilities are examples of this type of trust.

Non-controlled trust is the subjective opposite end of the axis representing this dimension in the trust space. It is owned by an actor of the system and can not be universally verified. There are many types of non-controlled trust mechanisms. They are only useful to its user. Reputation or royalty belong to this type of trust: they are defined by the actor itself, using its own measures based on observation and/or heuristic functions.

Other dimensions

Trust must be considered on a space-time framework. Therefore, time and space must be also considered as dimensions for trust. Any trust model has to deal with the evolution of trust in time. Many common mechanisms are inadvertently using time. For example, reputation: it is not the same the reputation of an actor on some time than after some days.

The space, understood as domain or scope, must also be taken into account. Trust is valid when involved parties share a common context, when they are close enough. To

extend trust into this dimension, different parties must join in a wider domain or scope by sharing ontologies, for example.

Dimensional interactions

The aforementioned dimensions, which are not probably all of them, are not independent. There are many interactions among them.

There are different views on how we can combine control and social (indirect) trust. They can be considered supplementary rather than complimentary. A good trust model must provide a broad viewpoint of electronic trust combining all views and where it is possible to locate all different facets of trust. That is to say that all dimensions of trust must be considered.

There are several issues here: firstly, it is almost impossible for any trusted third party to provide perfect enterprise-wide security. Secondly, a trusted third party is often used to provide one secure channel for communication between a first and second party, there often exists other channels that are not secure. Thirdly, end-to-end security often doesn't exist rather there are islands of security, point-to-point systems that have limited integration and supported limited dimensions of security. Fourthly, we cannot often adopt a revolutionary approach of designing systems for total security from the ground-up, rather we need a evolutionary approach to increase the amount of trust and security in existing systems. Finally, although a second party can be entrusting with confidential information or charged to do some action, there is often no direct control or verification that they will keep the information confidential or that they have the ability and are not otherwise prevented from carrying out the action. The second party maybe autonomous it may not be able to be directly controlled or internally monitored.

In chapter 4 we present a new trust model that can be used to find out security requirements in mobile agent applications and look for solutions.

Chapter 3

MARISM-A

In this chapter we present a new secure platform for the execution of mobile agents we have developed to answer the high demand for sea-of-data applications. It includes other desirable features on these environments, apart from the basic aspects found on most of them. Furthermore of protecting different security aspects of communication and migration, the new designed agent architecture allow the protection of agent itinerary, data and code. This scheme has lead to a new programming paradigm based on agencies. The platform itself, together with the specification of several mechanisms and an integrated development environment (MARISM-A IDE), comprise the MARISM-A project, an **A**rchitecture for **M**obile **A**gents with **R**ecursive **I**tineraries and **S**ecure **M**igration [CCD02], [RMB02]. All these characteristics make MARISM-A appropriate to develop and execute sea-of-data applications.

We will use MARISM-A as a testbed for our trust model approach in next chapter. In fact, the trust model and MARISM-A have had a symbiotic relationship, each one having profit from the other in their parallel construction. The rest of this chapter introduces the setup of the MARISM-A platform and analyses its security requirements using a variation of the Byzantine Princes problem explained in chapter 2.

3.1 Introduction

For some time now, multi agent systems have proliferated to fill the demand for applications based on this technology. In chapter 2 we have reviewed agent technology and multi agent systems. Agents make feasible the easy implementation of a wider variety of applications, hard to develop by using classic paradigms. Agents are also an enabler of new types of applications, standing out the so called sea-of-data applications. In these applications a massive amount (sea) of information is distributed among internetworked locations. This information can never leave these locations, but it is needed by a program to provide an outcome. A program can not just fetch the information and locally process it. There are several reasons preventing this, including the ratio volume of information / available bandwidth, the way this information is stored (it might be acquired by a special device), or legal reasons (medical images in hospitals). Agent technology is the basis to make these applications come true. The price for this is a new branch of open issues concerning security. This is especially tricky on mobile agent systems, where agents are not static and can migrate from one environments to another to be continue an execution.

Main advantages of using mobile agent systems include moving the code to the remote data, instead of moving the data (useful when data volume is huge), the possibility for the initial (launching) agency to be offline and access remote resources at the same time (very useful with weak or expensive network links), or the tasks parallelisation, allowing scalability in the processing of information. These advantages are very useful in the field of telecommunications and massive information process, and it results indispensable for some cryptographic applications [Rie99], [BTR+01]. MARISM-A benefits from all these intrinsic advantages, and adds some others that make it especially suitable for sea-of-data applications. Some of the new features include secure migration, secure agent communication and the coexistence of different agent architectures (including our new recursive agents) in the same heterogeneous framework. The reason for this is that the MARISM-A project is being used at the moment to implement several new technologies developed by our research group [RLN+02], [NRB02b]. All these traits confer to the platform the capability of easily creating new applications to solve difficult problems hardly faceable without a framework like MARISM-A,

FIPA/OS [FOS02], a well known multi agent system programmed in Java, has been the starting point of MARISM-A. FIPA/OS observes most of the FIPA specifications [FIP00] for agent communication and platform elements, and so does MARISM-A. Mobility mechanisms, not very concerned in FIPA specifications, have been implemented taking into account the MASIF mobility standard [MBB+98].

As a result of all of this, from MARISM-A has emerged a new paradigm for mobile agent programming. This new programming model is agency oriented and let complex applications with itinerant agents to be very easily coded. Furthermore, the new agent architectures we have designed allow to independently keep secrecy on all agent components, while preventing attacks against integrity at the same time. This is indeed one of the more novel features of our platform. This model extends the object oriented programming to location oriented in a highly intuitive fashion. This new programming model makes possible a new type of developing environment (IDE). In this IDE, the programmer graphically defines the itinerary of the agent and adds the code to be executed in the nodes representing agencies.

One of the most important things borne in mind during MARISM-A designing stage has been flexibility and scalability. Even different types of agents, with different migration patterns, can be embodied by programmers.

3.2 Platform description

The native language of the implementation of MARISM-A is Java. For the time being, all components of MARISM-A are being implemented only using this language (100% Java). This makes MARISM-A a very portable platform, essential feature in this kind of environments. MARISM-A is still on its early stages. We have implemented secure mobility (migration) and have simple applications running. The security framework in the platform has been based on a Public Key Infrastructure (PKI) (described below). At the moment we are implementing different agent architectures and mechanism to protect agents parts, such as itineraries.

3.2.1 Elements

The basic element in our platform, apart from the agents, is the agency. This is the basic environment for the execution of agents. An agency consists of a directory service, an agent manager and a message transport service. An agent system has several agencies distributed on a network. Each agency in controlled by an entity (its owner).

Agents are software units executing in the agencies on behalf of their owners. Agents in MARISM-A can be mobile or static, depending on the need of the agent to visit other agencies to fulfil its task. There are several types of mobile agents according to the characteristics of its architecture: basic or recursive structure, plain or encrypted, itinerary representation method, etc. Agents can communicate each other through the agency communication service. Figure 3.1 shows different types of agent supported in MARISM-A.

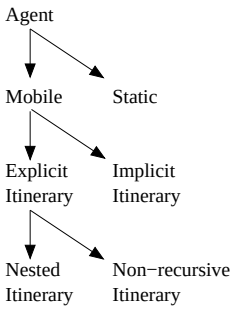

Figure 3.1: Supported agents types in MARISM-A

3.2.2 Mobility

Since mobility implementation is vaguely defined in FIPA specifications, we have designed our own migration protocols. We have tried to observe the MASIF standard as much as possible, always abiding by FIPA structures. Our mobility solution consists of a two level protocol: at message interchange level using FIPA Agent Communication Language (ACL), and at a negotiable transport level for the transmission of the agent. The migration of the agent is negotiated between agencies using ACL. Agent requirements about execution and the underlying transport protocol are dealt with at this level. Since FIPA/ACL language is supported by all FIPA compliant platforms, migration compatibility with other FIPA

platforms is guaranteed. Platforms without the agent demanded execution capabilities or without common transport protocols must refuse the migration request. This flexibility makes MARISM-A able to transparently inter-operate with other platforms.

If the agent destination agency accepts the migration request then the second level of the migration protocol will be started. At the moment, our platform is only accepting MMP (MARISM-A Migration Protocol) over TCP/IP, a basic protocol allowing the transmission of data and code of agents. The destination agency reconstructs the agent in the remote side and resumes its execution. The whole migration protocol is shown in figure 3.2.

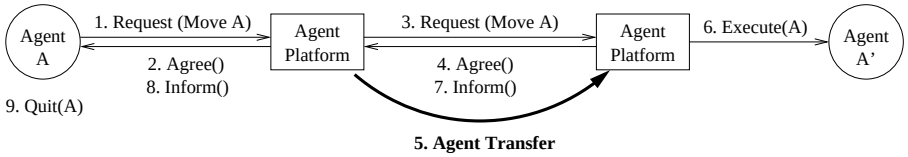

Figure 3.2: Migration Protocol

Step 5 of figure 3.2 represents the agent transfer process. To carry out this process we have used the reflection mechanisms of the Java language (getting information about classes and interacting with themselves), and its capabilities for network programming. These are the basic steps of the process:

- **Agent Serialisation** Data (objects) and code (classes) are converted into an easy to transfer format that allows to have all components of the agent on just one information structure.
- **Agent Codification** The agent is encoded according to the specified transfer protocol.
- **Agent Transmission** A connection with the remote agency is established and the agent is transfered through it.

The remote agency carries out the inverse process when it receives the agent.

We have used a classic client/server scheme to implement the information transmission task. Each MARISM-A agency has four components devoted to this process: a code server

and client and a data server and client. These elements make possible to have different instances of the same agent getting into the agency. Agent classes are stored on a class cache memory, allowing them to be downloaded only once in case or frequent use. When the agency receives a migration request it instantiates a data and a code clients to obtain the data and classes from the source agency (which is using the corresponding servers). These clients use a special class loader of MARISM-A (*AgentClassLoader*), which is a subclass of the default Java class loader. This new loader sees to download remote classes, while the default class loader is still used to load local classes.

When the required code and data is obtained, the agent is reconstructed and its execution is resumed.

3.2.3 Security Infrastructure

Security has been one of the weakest points of the FIPA standard. There is a work group now in FIPA devoted to security, but there is not yet a consensus about how security must be solved. The security solution we have implemented in MARISM-A is a proposal in this direction.

Desirable features regarding security for MARISM-A include:

Secret migration Agents must be kept secrecy while transfered from one agency to another. Only involved agencies have to have access to the agent.

Authentic migration Migration is only possible between authorised agencies.

Non-repudiation migration When the migration is over, none of the involved agencies can repudiate the transmission / reception of the agent.

Secret communications between agents Established communications between agents in the same agency have to be secret to other agents. Spy agents are not effective in this platform.

Authentic communication between agents Agents must prove their real identities to establish communications. This is indispensable in applications where agents from a same owner must interchange information.

Anonymous communication between agents Sent messages between agents may have an undisclosed source and sender, becoming anonymous messages.

Non-repudiable messages between agents After the sending or reception of a message, a sender or receptor can not deny its action of send or receive.

Some of these features are inherent in the platform itself. For example, security in agent communications. Because messages rely on an agency internal service, privacy and authentication between agents is guaranteed against other agent attacks. The communication system allows to hide sender identity and therefore source agent anonymity is preserved. A subsystem for message logging in the agency makes possible the non-repudiation functions. We plan to have some more complex mechanism to store proves of agent migrations, such as a receipt repository.

Secrecy, authentication and non-repudiation during migration are basic security requirements for most applications. We have implemented a basic security infrastructure to fulfil these requirements. Security requirements in MARISM-A are analysed and discussed later in this chapter, by using a variation of the Byzantine Princes Problem methodology. The flexible architecture of MARISM-A allows to later add specific safeguards implementations to particular security requirements. To fulfil these basic security requirements we have used a PKI shared by all agencies in the system. All cryptographic protocols used in MARISM-A use this PKI. A Certification Authority (CA) is the central trust element. All needed certificates in the system are issued by this CA and stored using a directory service (DS). Secure LDAP protocol is used to access the directory.

Secure migration protocol to transfer agents between agencies uses SSL protocol beneath. This will assure secrecy and authentication in the migration process. Required cryptographic keys are obtained from the directory service and the certification authority is used as the common trusted party. To avoid the chance of repudiation after migration some logs are kept. We plan to use a simple protocol to prevent repudiation. The aim of this non-repudiation protocol is to finish with the agent in one end and a receipt (or non-falsifiable prove of reception) in the other. This protocol is detailed on chapter 5.

The usage of this PKI, jointly with the new designed agent architectures (analysed below), allows other security PKI-based extensions for the platform, such as protection of

data, code and itinerary using asymmetric cryptography.

A secure mobile agent system based on a PKI presupposes, up to a certain level, a centralisation of trust: all actors must believe in the honesty of a common certification authority. This PKI lets have several hierarchically organised trust levels. This is not a limitation for the potential applications of our platform, since these applications already have an implicit hierarchic trust. Nevertheless, MARISM-A is not limited to this PKI and other mechanisms can be applied to provide a distributed notion of trust. For example, a Simple Public Key Infrastructure (SPKI) can be used for resource access control [NRB02a].

3.3 Agent architectures

One of the novel aspects introduced in our platform is the flexibility for agent architectures. In chapter 2 some security solutions for mobile agents were reviewed. Most of them had special agent requirements. Instead of focusing on a specific type of agent, we have created different agent architectures and, which is more relevant, we have enabled MARISM-A to accept different agent architectures. Architectures can provide their own security mechanisms since they may depend on agent structure. Even mobility is a feature of only some agent architectures. Moreover, our design allows to have several types of agents living together in a heterogeneous environment. MARISM-A provides some predefined architectures.

Most bibliographic references on agents do not make a clear distinction between different parts of an agent. Some of them suggest the need of considering independent some internal parts, especially for mobile agents. This is the case of agent data in [SR98], of agent code in [BHRS98], or agent itinerary in [BRSR99] and [KAG98]. Independence of these parts plays an important role for some agent protection mechanisms, whereas it is unnecessary for others. In MARISM-A, the architecture of the agent is an adaptable model that determines the different parts in which an agent is divided and the combination of security, integrity, and other mechanisms included in it. Figure 3.1 showed different agent architectures implemented in MARISM-A.

All mobile agent architectures share some basic aspects, such as the differentiation of internal parts and migration mechanisms. A mobile agent consists of code, data, state, and

an implicit or explicit itinerary. Code is the set of instructions describing the execution of the agent. Data is an information storage area that can be used by the agent at any moment for reading and writing and goes with it all the time. Results of executions are stored in this area, normally using some convenient protection mechanisms in such a way that only agent's owner will be able to read it. State is like the data part of the agent, but reserved to store agent information regarding its state. Explicit itineraries are structures containing the agencies that are going to be visited by the agent on its life cycle [MB02]. Just as we reviewed in chapter 2, itineraries are constituted by several basic structures: sequences, sets and alternatives. These structures can be combined to build complex itineraries. In a sequence, the agent will migrate to each agency one after the other. In a set, a group of agencies will be visited by the agent in no special order. On the other hand, only one agency of those listed in an alternative will be visited by an agent, depending on some conditions. Note that set structures is the only one involving cloning of agents. If cloning is not desired it is enough to replace sets by alternatives to all possible arrangements of the nodes, and then set the condition to a random selection. The snapshot of figure 3.5 shows an itinerary with an alternative (starting in triangle nodes), a set (semicircle nodes) and sequences (round nodes). Our itinerary allows to use a wide range of mechanisms to protect it (secrecy, integrity) and provide anonymity to involved agencies.

The implementation of agent management methods is included in the very agent. This allows, for example, to delegate specific mobility functions to the agent, freeing the agency from this responsibility. Using this implementation, MARISM-A becomes more generic, simultaneously supporting a wider range of agent architectures. The main architectures (types) of agents included with MARISM-A for the time being are described next.

3.3.1 Static

Static architectures are also supported by MARISM-A, as explained in previous sections. Many system agents do not need to migrate to other systems because of the tasks they must carry out. Whereby, they can be static. User agents can be static as well. Because agent management code is in the very agent, it is indifferent for the platform to deal with mobile or static agents.

$$\text{Agent}_i = \text{ControlCode}, \text{State}_i, \text{Code}_i, \text{Data}$$

There are not many words to say about security in static agents. Communication and interface with other agents are provided by secure services of the agency. Data protection is assured by the agency too, and there is no itinerary to protect here.

3.3.2 Mobile with implicit itinerary

This agent architecture is very similar to the static agent, except for the control code that in this case implements mobility. There is not a separate itinerary, but it is merged in its own code. A simple method will allow to migrate to other agency.

$$\text{Agent}_i = \text{ControlCode}, \text{State}_i, \text{Code}_i, \text{GlobalData}$$

The same code is executed by all agencies. Thus, there is no point in keeping it secret (assuming a secure migration, the code it is already secret to the agencies not involved with the execution of the agent). Data is shared by all agencies and can be protected since it is apart.

It might be interesting to protect integrity and secrecy of data that has been written in some agency (outcomes). In a shopping application, for instance, where agencies represent shops and agents act on behalf of buyers, it might be necessary to protect offers from competitors. This is a common situation in sea-of-data applications, where there exists distrust between agencies but honesty toward users.

Some of the data area is reserved to store results from executions (Results Data). Results can be stored plain:

$$ResultData = R_1, R_2, ..., R_n$$

where R_i is the outcome in agency i.

If the application requires confidentiality and integrity for this information, MARISM-A provides a protection mechanism based on hash chains [DH00], [HC02]. Results are firstly encrypted using agent's owner cryptographic information. Only the owner of an agent will be able to read its results. Once the result has been written, a hash of the Result

and previous hashed information is calculated, signed and written next. This hash has information about the identity of next agency in the itinerary, therefore no agency can neither modify the result area nor remove some result. Each agency verifies during immigration that all hashes in the Results Data are correct. This is the format of the Results Data area:

$$\begin{aligned} ResultsData = {} & E_o(nil, Id_1), S_o(H(E_o(nil, Id_1))), \\ & E_o(R_1, Id_2), S_1(H(E_o(R_1, Id_2))), \\ & E_o(R_2, Id_3), S_2(H(E_o(R_2, Id_3))), ... \\ & E_o(R_n, Id_o), S_n(H(E_o(R_n, Id_o))) \end{aligned}$$

where $E_o(m)$ is an encryption of m that can only be decrypted by the owner of the agent (o); $S_i(m)$ is a signature made by i; R_i is the result of agency i; Id_i is the identity of the next agency, i; and $H()$ is a hash function.

3.3.3 Mobile with explicit nested itineraries

Agent code is split into several pieces when using this architecture. There is a main code that will be executed in all agencies (Common Code), and as many code fragments as agencies are in the itinerary, each one to be executed in a particular agency (Local Code). This feature makes MARISM-A very useful in some types of application where execution is dependent on the context. The agent changes after a migration (the used part is removed). This agent aspect dynamism allows several security mechanisms to be applied. In this architecture the agent has a recursive structure as follows.

Agent$_i$ =	ControlCode, State, CommonCode, GlobalData
	Itinerary$_i$
Itinerary$_i$ =	LocalCode$_i$, Data$_i$, Agencies$_i$, Itinerary$_{i+1}$ | *Nil*

Agencies$_i$ is the agency (or agencies, depending on the type of itinerary) the agent is going to visit (migrate) next from i. The agent that is sent to the next hop of the itinerary (Agent$_{i+1}$) has the same structure. The last host is identified with a *Nil* next agent. CommonCode is executed by all agencies when the agent immigrates and before the specific LocalCode. Programming is simplified by using this common code to include the agency

independent code only once. The control code in the agent deals with the functions of agent management, in this case extracting the relevant parts of the agent.

This architecture can only be used with tree type itineraries. When the itinerary contains cycles it is not feasible to do the recursive encapsulation. If a tree type itinerary is required then some other architecture must be used, for instance the explicit non-recursive itinerary (see next section).

This architecture allows to implement several protection mechanisms to protect results data, in addition to the mechanism presented in the implicit itinerary architecture. Furthermore, code and itinerary are also protectable for agents using this architecture. The idea is to take advantage of the nested structure of the agent and make available (possible to decrypt) to an agency only the portion of the agent needed for the local execution. This is the structure of the agent when using data, code and itinerary protection:

Agent_i = ControlCode, State, CommonCode, GlobalData
Itinerary$_i$, $S_o(H(ControlCode, CommonCode))$
Itinerary$_i$ = $E_i(LocalCode_i, Data_i, Agencies_i,$
$E_{i+1}(Itinerary_{i+1}))$ | *Nil*

3.3.4 Mobile with explicit non-recursive itineraries

This architecture is similar to the previous one, but reduces its complexity by using a non-recursive structure. Instead of a nested agent, the local code and data are at the same level here. All parts of the agent are exactly the same.

Agent_i = ControlCode, StateData, CommonCode, GlobalData,
Itinerary
Itinerary = (LocalCode_1, LocalData_1, Agencies_1), ... ,
(LocalCode_n, LocalData_n, Agencies_n)

Main differences between these last two architectures (recursive and non-recursive) lie in the way a protection method for data, code and itinerary is applied. An agent of this type

can be unaltered during all his time life, whilst the recursive agent makes necessary to send a modified version of itself to the next agency. To avoid agencies attacks, agent integrity must be guaranteed through some hash verifications:

$$
\begin{aligned}
\text{Agent}_i = \quad & \text{ControlCode, StateData, CommonCode, GlobalData,} \\
& \text{Itinerary, } S_o(H(\text{ControlCode, CommonCode })), \\
& S_o(H(\text{Itinerary})) \\
\text{LocalStructures} = \quad & E_1(\text{LocalCode}_1, \text{LocalData}_1, \text{Agencies}_1), \ldots , \\
& E_n(\text{LocalCode}_n, \text{LocalData}_n, \text{Agencies}_n)
\end{aligned}
$$

There are at least two variants of mobile agents with explicit non-recursive itineraries: ordered and scrambled. In the scrambled non-recursive agent, the list of information for agencies ($Itinerary$) is scrambled. This makes it not possible to know which is the part of the agent that will be executed on some agency. Whereas in the ordered itinerary, all parts are arranged in execution order.

This architecture allows to use itineraries containing cycles, unlike recursive architectures. This is because structures forming the itinerary can be reused here. This architecture is one of the most versatile because of its flexibility and protection capabilities. The only drawback is that makes the number of agencies known.

All security solutions provided with agent architectures are based on the aforementioned Public Key Infrastructure (PKI). Moreover, we assume that agencies distrust each other, and therefore they will try to modify results carried by the agent, or to gain knowledge about its itinerary, to favour themselves to the detriment of the rest. We also assume that agencies are not malicious and they do not seek to adversely affect the owner of the agent, or the agent itself.

3.4 Implementation

The implementation of MARISM-A has been utterly done in Java and consists of three different parts: agencies, agents, and an set of tools for developing agents, including an IDE.

Different agent architectures are represented with a hierarchy. Although MARISM-A provides some architectures already implemented, this hierarchy may be extended with new user-defined architectures, as long as root classes are kept. Each class implements specific features of the architecture. Classes implementing basic security mechanisms for each architecture have also been implemented in MARISM-A basic classes.

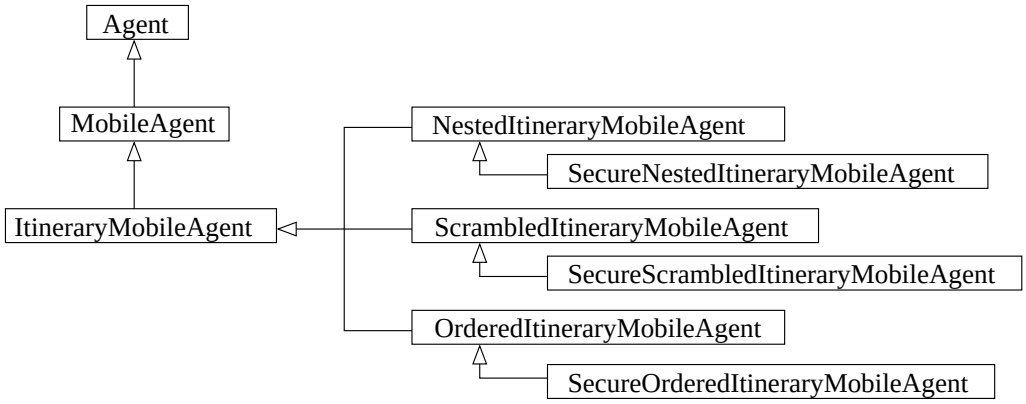

Figure 3.3: Agent classes in MARISM-A

Below we provide some implementation details of one architecture as an example.

Mobile agent with nested itinerary

This architecture is implemented in MARISM-A class *NestedItineraryMobileAgent*. This class extends *ItineraryMobileAgent*, as shown in the basic agent classes diagram (figure 3.3), and inherits its mobility behaviour and itinerary components. All the agencies in the itinerary are represented with the class *Node*. Specific methods of these classes are shown in figure 3.4.

A main method in the control code of the agent has the basic algorithm of itinerary management:

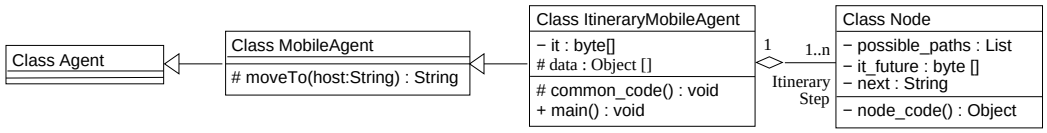

Figure 3.4: Class diagram

```
Agent current = decode( encodedAgent );
current.executeCommonCode();
current.executeLocalCode();
data.add( current.result );
moveTo( current.nextNode, current.nextAgent );
```

After agent decodification is done all components are ready to be used in this agency. First thing to do after decodification is to execute the common code and local code, in this order. Result of the execution is stored next in the data area. The agent is now ready to migrate to next agency. The *moveTo* method has conveniently been overwritten in this class to create the new agent (inner in the nested structure) which will travel to the next hop in the itinerary.

In the secure version of this type of agent (subclass *SecureNestedItineraryMobileAgent*), *nextAgent* is encrypted in such a way that only next agency in the itinerary will be able to decrypt it. Methods to encode and decode carry out cyphering at the same time to protect agent privacy.

Integrated Development Environment

The implementation of the Integrated Development Environment (IDE) has also been done in Java. In this IDE, the programmer graphically defines the itinerary of the agent in the Itinerary Creation Tool and adds the code to be executed in the nodes representing agencies. This description of the agent can be saved (and loaded) to disk using XML. When the programmer has finished the design the agent (java code) can be generated and used. For the time being, we have developed a first version of this IDE and we are now integrating cryptographic functionality. Figure 3.5 shows a screen-shot of the Itinerary Creation Tool,

part of the IDE. The drawn itinerary represents the classic example [SRM98] reviewed in chapter 2, where an agent must buy some flowers and either buy a ticket for theatre and go to a restaurant, or buy a ticket for a museum and go to other restaurant.

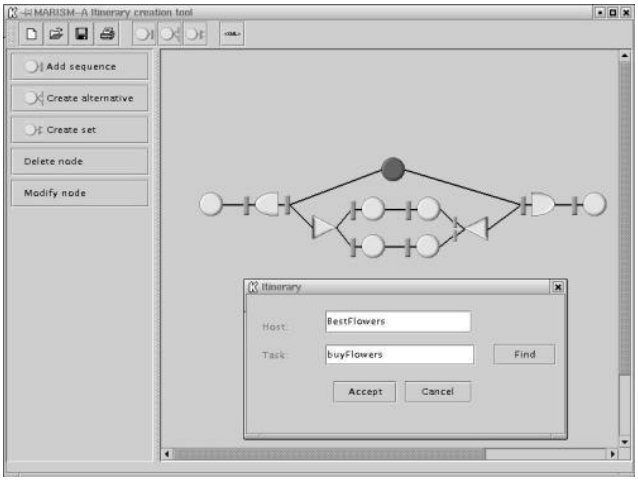

Figure 3.5: Itinerary Creation Tool, part of MARISM-A IDE

Each node represents a host with an specific code. The same agency can be represented through several nodes if the agent has to execute there more than once (different code each time). The same node is used for the same agency if the agent executes the same specific code.

3.5 Security requirements using the Byzantine Princes Problem

[DM01] proposes a problem, the Byzantine Princes Problem (BPP), consisting on a novelistic setup and a non-exhaustive list of security issues that can be projected to mobile agents applications. The BPP setup and issues are listed on chapter 2. This has been a good first approach to determine security issues in this type of applications from a list of well known problems on a real world scenario. Providing safeguards for this problems guarantees the

solutions for most common security issues in the mobile agents application. Nevertheless, the problem is not broad enough to be used for any type of mobile agent application.

It is not easy to find a proper problem to do this. Requirements of a useful problem include:

- Resources are distributed.
- It is hard, or not feasible, to move required resources.
- Users do not need to be on-line during the solution of the problem.
- Tasks may need results from previous tasks.
- Itinerary is explicit and can have alternatives.
- Some locations may need specific tasks.
- Security mechanisms are not inherent in the problem.

BPP does not fulfil all these requirements. Below we have analysed the BPP in MARISM-A and drawbacks of the problem are highlighted afterwards. We have designed a new problem to better represent security problems of any type of mobile agents application, including sea-of-data, through a real world problem based on the BPP. The new approach takes into account the problems in BPP to formulate a more convenient scenario.

This is the best correspondence between BPP elements and MARISM-A parts that we have achieved:

King	User / Authority
Castles	Agencies
Princes	Mobile Agent (Specific Code) / Agencies
Caravan	Mobile Agent (Architecture)
Caravan Personnel	Mobile Agent (Control Code)
Chess Board and Pieces	Mobile Agent (Results)
<Caravan Boot>	Mobile Agent (Data)
<Chess Rules>	Mobile Agent (Common Code)
<inherent in the problem>	Mobile Agent (Itinerary)
–	Agency Resources

Note that some of the elements in MARISM-A correspond to one or several elements of the BPP setup, or even to none.

Below follows the correspondence of security issues in MARISM-A with the list of problems in [DM01] (reviewed on chapter 2).

1. **A caravan may be destroyed while in transit.** An agent may be destroyed during its migration between agencies. The correspondence is not direct: destroying the caravan still preserves some parts of the agent, such as specific and common code.

2. **The board position carried by a caravan could be changed while in transit.** Some result data carried by the agent could be modified or removed during migration. In this case there is a straightforward correspondence. However, the limitation of being in transit obviates more realistic *ad-hoc* situations where result modification is more likely to happen.

3. **The caravan personnel may be exchanged with people with a different mission, including ones who wish to misuse the access rights of the caravan, or want to modify the legitimate outcome of the game.** A third party may intercept the agent and replace the control code to take profit from the access rights of the agent, or to modify carried results.

4. **The caravan personnel could be bribed to act improperly.** The analog problem in MARISM-A is not clear. Agent modification is considered in the previous issue, including control code. To bride someone to do something is equivalent (but not feasible on a real world scenario) to replace her with an exact replica person and new mission. In MARISM-A this problem is the same as the previous one.

5. **Imitators of the caravan may arrive at a castle.** Imitators (clones?) of the agent may arrive to an agency. This problem is similar to the previous two except the original agent is still moving around.

6. **A prince may have the caravan destroyed while it is at his castle.** An agency may destroy an agent when it is inside.

7. **A prince may attempt to change the board position before his move.** An agency may attempt to change or remove result data obtained in other agencies before inserting its own.

8. **A prince may attempt to make an illegal move.** This is an ambiguous issue. In one hand it can be the addition of a wrong result. In the other hand, it can be the wrong execution of agent's code.

9. **A prince may attempt to never make his move.** An agency refuses to execute the agent, blocking it. This is a denial of execution attack. It is not clear if the prince (agency) let the agent migrate to next castle (agency).

10. **A prince could send a imitation caravan ahead to see what the next round of moves would be, and then give his illegally wiser chess move to the real caravan, and send it on.** In the case of an itinerary with some loop, an agency may modify the agent the second time it migrates to it to pretend it is the first time and take advantage of previously collected results. If the itinerary has no loops, this is not applicable.

11. **The caravan could mistakenly record a prince's move.** An agency could mistakenly create the local result information. This is not likely to mistakenly happen but deliberately, and in the latter case the situation is covered elsewhere.

12. **Imitation castles may be set up to deceive the caravan into visiting the wrong prince, or someone who isn't a prince at all.** Not registered agencies may spoof other agencies to let them think who they are not. Also agencies falsifying identities.

13. **A prince could send out a duplicate caravan as well as the real one, so that multiple games are being played. At a later move, the prince or his brother may destroy the caravan with the less favourable chess position.** Agencies could send out agent clones. In case of itineraries containing loops the agency could destroy the less convenient replica.

14. **A prince may falsely claim that an illegal move has been made to try to force a number of moves to be taken back.** Agencies may falsely claim an agent has been modified to try to force a number of hops to be taken back.

15. **A prince's subjects may lose confidence in the prince if his ineptness were revealed by disclosure of the board's status.** Reputation of an agency may lose confidence if the results of agent executions are publicly revealed by disclosure of agent information.

16. **The caravan personnel harm the prince.** This is ambiguous. The agent harm the agency or the agent is attacked..

As we have seen, BPP is not easily applied to MARISM-A and to any sea-of-data application in general. Many important elements in this type of application do not correspond to one element in this setup or even have no representation at all, such as the itinerary (in the BPP the itinerary is inherent in the problem) or resource management.

Next we propose a novel problem, based on BPP, in which a wider number mobile agents application can be identified. This is the Byzantine Princes Problem Sequel (BPP-S).

Byzantine Princes Problem, the sequel

The Byzantine Princes Problem Sequel (BPP-S) is based on BPP and shares its setup. Let us remember that the problem description finished with a chess game being played.

The chess game is played during months and finally a pair of siblings win the disputed challenge. Before the princes take up the throne, their ruthlessness manners make them constantly quarrel. The old king soon realises the impossibility of a joint reign. After having slept on it, and advised by the wisest royal counsellors, the king resolves to organise a new and definitive challenge to decide which of the princes would become the next sovereign of the kingdom in his succession.

The new challenge consists of solving a very hard enigma, that can be only solved after seeking information in the depths of the realm, kept in the knowledge of witches and wizards, looking up strange books hidden deep down lost monasteries, and consulting far off oracles, spread out all over the ends of the kingdom. All these information sources will cooperate with both princes to provide required answers, expecting some favour from (or be ignored by) the future king. However, they will try to unveil as little as possible of their secrets and knowledge and will distrust each other because of rivalry. Some of

these knowledge custodians may demand some items from other sources. For example, a witch might need a special ingredient, only available in dragon's lairs, to make a magic sleep potion; or a library guardian / keeper monk may *need* a magic sleep potion to let the emissary visit the library.

Although the enigma is the same for both princes, there is not a unique way to solve it. Each prince makes a detailed plan to obtain the required information to solve the enigma, helped by their own trusted advisors. As seen in the BPP, princes will not leave their castles for fear of being killed by their own subjects. Thus, each prince will send an emissary with explicit instructions to find the valuable information. These instructions contains the exact whereabouts and the required interaction needed to obtain different clues that will allow the prince to solve the enigma. The emissary is trained to be able to interact with all information sources and to strictly and doubtlessly obey the prince's instructions.

Princes will try to intercept, thieve from, and sabotage emissaries from his counterpart. Therefore, emissary protection must be provided. Witches, monks, or oracles might betray a prince by disclosing emissary instructions to the other prince, or even be bribed to do it.

Analysing MARISM-A with the novel BPP-S problem

BPP-S provides a more suitable scenario to be projected on more types of mobile agent applications. All aforementioned requirements for a problem like this are met in BPP-S. Below we try to look for the correspondence with MARISM-A (assuming agents with explicit itinerary) and how it could be done with other applications.

King	Authority
Prince in a Castle	User
Emissary	Mobile Agent
knowledge places[1]	LocationAgencies
knowledge[2]	Resources
knowledge custodians[3]	Resource Manager Agents (in Agencies)
Enigma	Problem to solve, aim of the application
Clues carried by emissary	Results
Emissary Instructions (whereabouts)	Mobile Agent (Itinerary)
Emissary Instructions (what to do)	Mobile Agent (Specific Code)
Emissary Training	Mobile Agent (Common Code)
Emissary Pockets	Mobile Agent (Data)

[1] Monasteries, Caves , *etc.*

[2] Libraries, Oracle, Ancient Knowledge

[3] Monks, Wizards , *etc.*

With this new scenario all parts of a mobile agent are kept together. This allows a better projection onto a real mobile agent application, without artificial elements and unnecessary situations. Different components of code and data are intuitively found in the character of the emissary. The architecture of the agent, which is flexible in MARISM-A, corresponds to the way an emissary is organised, which is also flexible. Instructions can consist of different numbered papers with a place and a list of actions to do on that place on each (nested architecture), or just a piece of paper with text containing both places and actions (implicit itinerary architecture). Data is also easy to distinguish between relevant clues required by the prince (results), hidden somewhere in the emissary, and information needed during the journey on her pockets, or hidden. Direct itinerary representation and new resource related elements make it easier to find and to analyse new branches of security requirements.

Security issues arising from the new BPP-S problem are quite different from the BPP. Some unreal situations (in mobile agent applications) have been removed, such as those regarding caravan personnel having different positions at the same time (in favour and

against the prince). New security questions come from considering issues ignored in BPP and very important in the sequel, for example regarding the itinerary of the mobile agent or the resources. The corresponding and non-exhaustive list of situations we have found for this new problem is set out below.

1. **An emissary may be killed while in transit.** An agent may be destroyed during its migration between agencies. All its components will be lost.

2. **The clues born by the emissary could be changed or removed while in transit.** Some result data carried by the agent could be modified or removed during migration.

3. **The emissary may be replaced with an other with a different mission, or bribed to act improperly.** A third party may intercept the agent and replace the code to take advantage from the access rights of the agent, or to modify carried results.

4. **Imitators of the emissary may arrive at a knowledge place.** Clones of the agent may arrive to an agency. This problem is similar to the previous two except the original agent is still moving around.

5. **A monk, witch or oracle (knowledge custodian) may kill the emissary while she is at a knowledge place.** An agency may destroy an agent when it is inside.

6. **A knowledge custodian may attempt to change or remove the results carried by the emissary.** An agency may attempt to change or remove result data obtained in other agencies before inserting its own.

7. **A knowledge custodian may attempt to wrongly execute an emissary's instructions or provide a wrong clue.** A Resource Manager in an agency can deliberately provide a wrong result or wrongly execute agent's code.

8. **A knowledge custodian may deny to provide the required clue.** An agency does not insert the result of the execution in the agent.

9. **If an emissary comes again to visit some knowledge custodian, the latter can change her previous result providing now less information, depending on the**

results of other parts (carried by the emissary) and pretend the emissary never left her place before. In the case of an itinerary with some loop, an agency may modify the agent the second time it migrates to it to pretend it is the first time and take advantage of previously collected results. If the itinerary has no loops, this is not applicable.

10. **Imitation information places may be set up to deceive the emissary of the prince into visiting the wrong knowledge place.** Not registered agencies may spoof other agencies to let them think they are who they are not. Also agencies may falsify identities.

11. **A knowledge custodian may falsely claim that the emissary is not admissible (saying that her credentials are not correct, for instance) to try to force some results to be rejected.** Agencies may falsely claim an agent has been modified to try to force a number of hops to be taken back.

12. **A source of information may lose prestige, reputation or importance if its information were revealed by disclosure of the results.** Reputation of an agency may lose confidence if the results of agent executions are publicly revealed by disclosure of agent information.

13. **The emissary may harm a knowledge custodian.** The agent attacks some agency.

14. **Itinerary of an emissary is analysed by master's brother's henchmen, in order to get to the places first or to sabotage her mission. This may happen while the emissary is in transit or in collusion with some knowledge places.** Agent itinerary is totally or partially unveiled during migration or while in some agency. Other agencies or users might take profit from this knowledge.

15. **The emissary may attempt to access forbidden knowledges.** Agent may attempt to access agency resources it is not allowed to access.

Although the number of questions in the original problem is similar than in the sequel, they are quite different because of the context. Since all elements from the BPP-S problem

and from the mobile agent application correspond through a one to one relation, security issues do not concern multiple issues at the same time. If it were, as in the BPP, solutions would not be independent.

BPP-S suggests a better analogy to abstract out security requirements more suitable for a variety of application domains, but it is not possible to claim that this list is complete.

Nevertheless, it is highly complex to find out all security requirements in this way, even if we have a good non technological representation of the problem, like the BPP-S. It is not feasible to make sure that the list of problems is complete. Other approaches result more convenient to analyse security in mobile agents applications, such as our trust model (chapter 4).

Security mechanisms in MARISM-A implemented as of today do not fulfil all these requirements. Some described attacks are avoided by the use of cryptographic protocols framed on a PKI, and some other are thwarted due to specific mechanisms, such as itinerary protections. Issues number 1, 4, 5, 7, 8, 11 and 13 are not yet considered in MARISM-A. Some of these problems are easy to take into account (problem 13, for example, can be solved by using a *sandbox* execution model), while other situations described in these latter problems are not concerning the applications in MARISM-A, and this is one of the problems with this methodology. Chapter 4 describes the usage of our trust model in MARISM-A to find out relevant (real) security requirements by means of trust.

At the moment of this writing, MARISM-A is not yet finished. Many agent architectures are already working and some complex schemes are planned to be added soon, such as the resource access control achieved through agent certificates over a SPKI. We are developing a first version of the IDE in which we are now integrating cryptographic functionality and agent generation.

Chapter 4

Trust Model

New approaches are needed to face the problem of security in multi-agent systems, for which traditional solutions are inefficient and expensive as it has been highlighted in chapter 2. This is specially convenient for recent applications using distributed architectures and socialised behaviour, such as those based on mobile agents. We have analysed security requirements of MARISM-A in chapter 3 by means of the Byzantine Princes Problem Sequel. This method guarantees neither that all requirements are found nor that the problems are relevant to an specific application. In this chapter we show a new viewpoint of the problem, considering trust as a social dimension of security and describing a model for it. This model allows to transform the security problem in these systems through the use of trust and easily find scalable, flexible and distributed solutions. Using trust instead of a straightforward security approach will allow us to deploy different mechanisms, from different nature, to obtain more secure systems not *per se* but for the members of the system and for its actors. These solutions fit specially the new security requirements in mobile agent applications and pervasive computing.

We are not trying to find a generic definition for modelling trust in a deductive fashion and specifically defining applicable trust solutions afterwards. Rather, we seek to focus on an inductive approach of the model from basic components of trust enabling it to be used in mobile agents systems [RB02b].

Our approach consists of a descriptive or conceptual model that focuses on the identification of trust problems and a methodology to derive a trust solution to apply to solve

these problems. Within this theoretic scheme trust concept appears clear and abstract which makes easier its understanding. It also results easier to model trust requirements and look for solutions to increase security. We introduce a plane of trust in which it is easy to represent the relationships between all members of the system. Within this plane of trust it is easy to show the real trust requirements of the system. A methodology will be essential to allow us to choose and combine different mechanisms to find trust solutions and finally have security solutions. The transformation from the security plane to the trust plane allows to screen the irrelevant issues concerning security from the most important in the specific scenario of mobile agent applications. In this trust plane the solutions are quite easier to find. The transformation of the results back to the security plane is trivial since the trust solutions directly are security solutions. By using this technique, found solutions will fulfil scalability, distribution and integration requirements within an unique and holistic security/trust scheme.

Traditional security solutions are not incompatible with our model. These solutions will be mechanisms in our model that could be used it a trust relationship requires it to be fulfilled.

The methodology we have defined to apply our Trust Model on a system is threefold. Firstly, we need to define the trust requirements and trust vulnerabilities though the identification of the elements of the model within the system. Secondly, we obtain or create mechanisms fulfilling some of the trust relationships previously identified. These mechanisms are full or partial solutions to trust requirements of the system. And thirdly, we choose and combine some of the these mechanisms to compose a trust solution that covers the whole set of trust relationships for the system. If more than one combination of mechanisms exists, only one of them will be selected as the trust solution for the system. This selection might take into account complexity (based on the number of involved agents or external parties), required infrastructures, etc. Basically, the final solution will consist of a collection of interrelated mechanisms to provide trust enablers for actors as safeguards to prevent attacks.

First versions of the trust model were based on static system situations[RBB$^+$01], [RPBB01b], [RPBB01a]. The model has now evolved with the replacement of situations by trust relationships, which is a more sensible approach considering the nature of the problem

to solve. This change has simplified and improved the model.

The rest of this chapter describes the elements of the model and describes the methodology to apply it. An example based on a scenario in MARISM-A will help to illustrate the use of the model.

4.1 Elements

Our model is oriented to systems using mobile agents and, more generally, to open systems with multiple participants. If consists of five basic elements: entities, actions, relationships, messages and verification functions.

Entities The entities in our scheme are all these components involved with trust. Actors (agents, users and resource providers) are entities that actively act and are bearers, transmitters and receptors of trust. Resources and agencies, final objects for trust, are also entities in the model.

Actions Actions are the events done by the actors of the system that are direct object of a trust relationship. An action is always related with an entity at least. An example of action is a resource access made by a mobile agent.

Relationships A trust relationship is a connection between systems entities to represent the mutual placed trust for the accomplishment of an action. An example for this is the relationship between two agents to access a resource owned by one of them.

Messages Communication is also a key element in our model. This communication is shaped through the interchange of specific messages between system actors to manage and handle trust aspects related with actions. Main classes of messages are appointment, action delegation, obligation and prohibition.

Verification functions This element includes control functions for objective trust. The control on subjective trust is carried out by the actors themselves, so it is not included here. Management of delegation mechanisms is included in this part of the model. Time dimension of trust is analysed during this verification.

Our model follows a non-centralised scheme based on system's actors actions and communication. The model shows several semantic domains of trust and it is open, providing a flexible framework to represent all different associated concepts of trust without bindings to any pre-established infrastructure. It also provides a high freedom degree for formalisation.

4.1.1 Methodology

We apply the theoretical model into practise through the design of a methodology.

The first stage of this methodology is the identification and characterisation of the elements in the model: entities, actions, relationships, messages and verification functions.

Afterwards, required trust relationships are pointed out in the Trust Requirement Diagram. Trust dimension of security requirements of the system will be represented through the trust relationships between entities.

A collection of mechanisms are selected from a mechanism repository (or especially designed) to satisfy trust relationships.

Finally, a subset of these mechanism satisfying all relationships is chosen by solving the Set Covering Problem. This will be the trust solution, a set of mechanisms that will totally cover the needs of trust and will consistently be a security solution for the system. These mechanisms must be adapted to the model to compose .

Trust Requirements Diagram

The Trust Requirements Diagram (or equivalently, the Trust Requirements Graph, TRG) will allow to find trust requirements for a system.

Complexity of the traditional analysis of security is usually due to the risk analysis which is done. Some authors find in risk analysis the only way to solve security problems [RHG99]. In our proposal we are using the trust plane, therefore focusing on trust relationships between actors of the system and leaving less relevant aspects aside.

To find the TRG we define a directed graph with a vertex for each actor and trust objects (entities). Edges in this graph are trust relationships required between entities.

TRG(E,R)

E={*System entities,* $(e_1, e_2, ...)$}
R={*Trust relationships between entities* $(r_1, r_2, ...)$}

Each one of the elements in R has the source and destination entity of a trust relationship.

$$r_i = (e_j, e_k), e_j, e_k \in E, r_i \in R$$

All these relationships have an associated action and some attributes for that action. The assignment is established through function A:

$$\begin{array}{rccc} A: & R & \rightarrow & Actions \text{ x } Attributes \\ & r_i & \rightarrow & (a_i, t_i) \end{array}$$

The action is the object for trust on a relationship. Examples of these actions are permission, obligation, designation or prohibition. Actions may have some attributes to indicate the scope or other relevant information. The object of permission or prohibitions might be the attributes for the respective actions.

Trust relationships in which there are more than two entries are involved can be slip up into several one to one relationships and therefor all type of trust relationships can be represented in the TRG.

Mechanisms

A mechanism is a a partial or complete solution to a trust requirement. It may include cryptographic protocols, control functions or infrastructures.

Mechanisms have some common components that allow the designer to characterise and identify them. These basic components allow to classify mechanisms, to store them on a mechanism repository and to look up for them later.

Identification:	A name to refer to the mechanism.
Description:	A description of its operation.
Relationship:	The type of trust relationship the mechanism is designed to solve.
Type:	The type of mechanism indicating whether its nature is *deterrent*, reducing the likelihood of the risk or the impact. *preventive*, protecting, blocking the impact of a possible attack. *corrective*, reducing the effects with respect to the relationship.
Dependencies:	A list of required mechanisms.

Trust Solution

When all elements of the trust model have been determined and characterised we will look for a set of mechanisms whose implementations meet the requirements imposed by trust relationships represented in the Trust Requirements Diagram.. Each of the mechanisms has to satisfy one or more of the edges of the Trust Requirements Graph.

In order to obtain the Trust Solution we need the symmetric bipartite Solutions Graph. This is created using available mechanisms as vertex of one side, and indirect trust relationships as vertex of the other:

SG(V,A)

V=(V_1 = {*Mechanisms*, $(m_1, m_2, ...)$ }) $\cup$ (V_2 = {*Relationships*, $(r_1, r_2, ...)$ })
A={ $a_{ij}|m_i$ satisfies trust relationship r_j, $m_i \in V_1, r_j \in V_2$ }

Edges of this graph may have a cost if the implementation complexity of the mechanism to satisfy the relationship is known.

The Trust Solution is the set of mechanisms represented by vertex from V_1 that provide a solution for the covering problem on graph SG. If edges have a cost, then the Trust Solution coincides with the SCP (*Set Covering Problem*) problem [CFT98].

The final stage of the methodology, once we have the set of mechanisms to use, is to adapt them to use messages described in the model to manage trust relationships and integrate them into the system. When these mechanisms are adapted, they can be added to

a library and be used again in subsequent analysis. Some examples of messages are action delegation request and delegation grant.

4.2 Practical application of the model: MARISM-A

As we have seen in previous chapters, MARISM-A is a novel mobile agents platform especially conceived to develop secure sea-of-data applications. But far from being too dependant to some type of application, it is very flexible and can be qualified as general purpose. Thus, MARISM-A is a perfect scenario to put into practise our trust model. The use of the trust model in MARISM-A let us to validate the model and add security to the platform at the same time. Both MARISM-A platform and the trust model have been designed in parallel. This symbiotic relationship that has made possible each part to profit from the other, detecting drawbacks of the model in advance and adjusting it to a real scenario.

Along next sections we analyse how the trust model can be applied to determine trust requirements and security safeguards in sea-of-data applications deploying MARISM-A. As all applications of this type have common scenarios, the example shown below is applicable to most of them. Different applications are often simpler that the case of sea-of-data applications, and therefore the way of applying the trust model in the first is included in the provided example.

We have chosen a simplified version of a real application to make a clear application of our approach and get the essence of it.

4.2.1 Description of the scenario

Along previous chapter we have defined sea-of-data applications. A common set up often consists of a user requiring some results that must be extracted from the analysis of a huge amount of distributed information (data). Data providers and the user have access to a shared network. Due to network constraints, it is not possible for the user to fetch all the information and process it locally. Furthermore, user has a limited link with the network and she is just intermittently on-line.

Let us consider our user to be a researcher on image classification. She is trying to

evaluate an algorithm she has designed to classify tumours on medical images. In order to do this, the algorithm has to be executed over thousands of images and the results must be verified against traditional human classification for assessing purposes. Several hospitals spread over the world have databases with that type of images and medical diagnosis. As both, hospitals and the user, are connected to the Internet it seems feasible to carry out the experiment.

Nevertheless, there are some problems that must be arranged first. The user has to face two initial problems: her connection to the Internet is though a not very reliable 56Kbps modem link, and hospital head office departments refuse to allow an external person to see medical images alleging legal issues. Moreover, they are afraid of executing alien programs due to virii threat.

Even though it seems there are no solution to the situation, mobile agent technology may help here by means of a Mobile Agent System like MARISM-A. Low quality connection of the user, legal issues regarding medical images, and the threat of executing external programs, all are irrelevant problems if MARISM-A is deployed. Hospitals accept to help the user since images will not neither leave the hospital nor be unveiled to external persons. They accept to install a MARISM-A platform because programs (agents) will be identified with their owner, agents can not execute arbitrary code in the host computer, and the system will allow the hospital to participate in other research projects including their own.

Thus, the user programs her algorithm on an agent. The itinerary includes the collaborating hospitals arranged according to the set of experiments she wants to carry out (in the best case only some of the hospitals will be visited). All hospitals have installed a MARISM-A platform and have connected it to the Internet. Data (medical images) are Resources for MARISM-A, and some agents are created on each platform to allow and control the access to these resources (Resource Provider and Resource Controller Agents).

Fortunately, MARISM-A has provided a better prospect to solve the problem. But far from solving all the problems, a long list of them begins to worry both user and hospitals: Is the incoming agent owned by the user it claims? Is the algorithm going to be executed exactly as it was programmed? Are hospitals going to modify results carried by the agent and obtained elsewhere (for instance to increase their own importance in the experiment)? Is the itinerary going to be modified in such a way that some hospital will be skipped? Next

section will apply the trust model to find out trust / security requirements and a methodology to provide solutions.

4.2.2 Trust Requirements Diagram

A sea-of-data application implemented in MARISM-A is a complex system regarding security requirements. Last section pointed out some of them. In chapter 2 and 3 we analysed how resemblances with a real world problem could help to find most common security requirements. As highlighted before, our approach proposes the use of trust to find these requirements. In the example, hospitals, the user, resources and alien persons are the leading actors in the scenario and seems sensible to analyse their mutual trust needs in order to block security attacks. Therefore, as a first step toward this goal we need to identify the entities of the model in the example.

It is clear that the user (e_1) and hospitals are entities. Hospitals are hosting agencies. We assume for simplicity that there are only two agencies in the system: Agency 1 at user's home (e_3) and a hospital (e_4). The program with the algorithm of the user is a mobile agent (e_2). Medical images are a resource of an agency (e_7) and there is a resource provider agent (e_6) to allow its accessing. This access is controlled by a resource controller agent (e_5). In order to make the example more general we will not refer to the entities with the names in the specific example, but with the names of the parts of MARISM-A (agency, mobile agent, resource, etc).

Figure 4.1 shows the Trust Requirements Diagram corresponding to the scenario of the example. Entities are arranges as vertex in this graph and trust relationships are represented with edges.

The mobile agent (e_2), acting on behalf of user (e_1), is executing on agency (e_3). The agent have to trust in the correct execution of its code in this agency. After this, it migrates to another agency, (e_4). This migration must be done securely, with a secret and authentic transmission. Subsequently, the agent will try to access resource (e_7), owned by resource provider (e_6). (e_2) must have an access permit obtained from the resource controller agent (e_5), acting on behalf of (e_6).

There are ten trust relationships (r_1 and r_{10}) between these entities. Actions associated

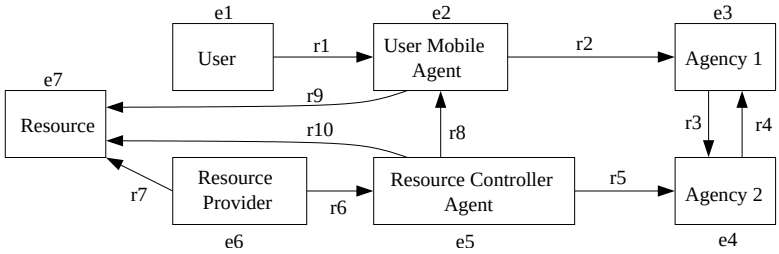

Figure 4.1: Trust Requirements Diagram

to these relationships are assigned by the function A in the model. In this case, these are the actions assigned with each relationship:

r_1 : Code execution
r_2 : Code execution
r_3 : Secure and authentic transmission
r_4 : Secure and authentic transmission
r_5 : Code execution
r_6 : Resource access permission
r_7 : Access
r_8 : Resource access permission
r_9 : Access
r_{10} : Access

Two from these ten relationships are direct trust: r_1 and r_{10}. A special mechanism to fulfil trust requirements derived from these relationships is no needed. If we analyse these two cases we notice they correspond to the special relationship between an agent and its owner, and between a resource and its provider. In both situations each part entrusts the other and therefore we have direct trust.

4.2.3 Trust Solution

Once we have obtained the trust requirements diagram of the problem, we have identified all entities of the system and the trust relationships between them. Next step is to select a set of mechanisms which could provide a way to fulfil the relationships.

If there exists a repository of mechanisms it is easy to select some from there following some basic search criterion, such as the type of relationship they cover. If not, we need find them by other means. We can look for mechanisms providing solutions to those specific

relationships in the bibliography or create our own. When these mechanisms are found they must be adapted to be used under the model.

We have selected three candidate mechanisms to be used as part of the Trust Solution in the example:

m_1 : *Identification:* STS
Description: Secure Transmission Protocol [DOW92].
Relationship: Regarding secure transmission.
Type: Preventive.
Dependences: N/A.

m_2 : *Identification:* SPKI Certificates
Description: Delegation System based on *Simple Public Key Infrastructure* (SPKI) [EFL$^+$99]. Certificates are used to prove authorisation to access resources.
Relationship: Regarding authorisation, delegation.
Type: Preventive.
Dependences: Depending of the use it may require a PKI.

m_3 : *Identification:* PKI
Description: Public Key Infrastructure (PKI) [MvOS96], including a Certification Authority to issue certificates and a Directory Service.
Relationship: Regarding identification, encryption, signing
Type: Preventive.
Dependences: N/A

From these mechanisms and the indirect trust relationships in the Trust Requirements Diagram, we create the Solutions Graph. Edges of this bipartite graph connect a mechanism with a covered trust relationships (figure 4.2).

To obtain the Trust solution for this scenario it is enough to solve the Set Covering Problem for the solutions graph. In this case the solution is easy to find: m_1 and m_2 mechanisms are sufficient to cover all relationships. There are well known algorithms in graph theory to solve this problem even if it has not trivial solutions [CFT98]. These two mechanism are being implemented in MARISM-A [NRB02a], [NRB02b], [RLN$^+$02].

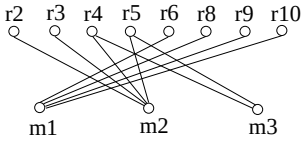

Figure 4.2: Solutions Graph

Last steps, when trust solution methods are already obtained, are to adapt them to the model and the integration with the system. To adapt the mechanisms to the model, they must use messages to manage different trust relationships. These messages will be the links among all mechanisms and system parts to achieve a global trust. Mechanisms implementing controlled objective trust must provide the correspondent verification functions.

Chapter 5

Ad hoc security solutions

In this chapter we will review three security solutions designed before year 2000. In the time these proposals were written, our trust model did not exist yet or was still in a very early stage. MARISM-A platform was also created after all of these proposals. This initial work showed us the problems arising from security designing and lead us toward the definition of a general model using trust.

Our approach to find security solutions to mobile agent systems (shown in chapter 4) is based on a trust model and a methodology. Some mechanisms providing specific solutions to cover trust relationships have to be selected as part of the methodology. A repository is mentioned to make this selection. This repository stores known mechanisms that can be used in later analysis. The solutions described in this chapter can be converted in mechanisms that can be part of this repository. Therefore, they can be used in the designing of complex security solutions using our methodology.

Apart from the use as mechanisms for our model, this former work have let us to acquire the required background in security and applications to face the development of the trust model and the MARISM-A platform.

Each security solution is on a separate section in this chapter and is self-contained. First section is devoted to a fault-tolerant mobile agent based system to perform large scale electronic voting over the Internet [RB02a]. The whole system was entirely oriented to the e-voting application.

Second section describes a non-repudiation protocol to secure the itinerary of mobile

agents [BRSR99]. This mechanism will be part of the MARISM-A default migration method.

Last section shows a set of mechanisms to avoid security attacks on a multi agent system implementing a marketplace for the selling of bandwidth in 3G telecommunication networks [RBB+01], [BTC+00].

5.1 A Fault-tolerant Voting Scheme based on Mobile Agents

Proposed electronic voting schemes have traditionally focused on security requirements. For instance, [RB99] fulfils the most widely accepted security requirements. However, most of these proposals suffer from an important lack of safety as they are not fault-tolerant.

In this section we introduce a novel fault-tolerant voting scheme. To achieve this goal we have redesigned the whole scheme of [RB99] by using mobile agents. These agents are organised into two levels. The agents in the lower level are related to electoral functions and the agents in the higher level to coordination tasks. We provide mechanisms to assure both electoral agents reliability and coordination agents persistence. With this scheme the election can be easily reorganised and resumed in case of host failure.

5.1.1 Introduction

The objective of electronic voting schemes is to allow elections to take place securely over general-purpose and open computer networks. During the ballot collecting process, a set of eligible voters use the computer network to cast their ballots. After some time, the system stops accepting ballots. The counting process is initiated and, finally, the tally is published.

Different security requirements for voting schemes have been proposed. The following list includes the most widely accepted ones [CC96, FOO92]:

Accuracy: A voting scheme is accurate if (1) it is not possible for a validated ballot to be altered, (2) it is not possible for a validated ballot to be eliminated from the final tally, and (3) it is not possible for an invalid ballot to be counted in the final tally.

Democracy: A voting scheme is democratic if (1) it permits only eligible voters to vote, and (2) each eligible voter can vote only once.

Privacy: A voting scheme is private if (1) neither ballot collecting authorities nor anyone else can link any ballot to the voter who has cast it, (2) no voter can prove that he or she voted in a particular way, and (3) all ballots remain secret while the voting is not completed.

Verifiability: A voting scheme is verifiable if voters can independently verify that their ballots have been counted correctly.

The first privacy property, known as anonymity, is probably the cornerstone of secure voting schemes. There are no obvious solutions to anonymity since in order to preserve the democracy requirement, voting authorities responsible for collecting ballots should have assurance of the identity of voters who contact them. The most widely accepted solution to this problem found in the literature consists in assuming the existence of an external anonymous communication channel [Cha81]. Even though current implementations of this channel can realistically be used for anonymous e-mail communications, they have disadvantages when used between voters and ballot collecting authorities in voting schemes.

In [RB99] we faced up with the anonymity problem by introducing mobile agents in the design of the voting scheme. This voting scheme is based on the concept of electronic Electoral College (EC) introduced in [RBR97]. Moreover, this voting scheme does not need any independent anonymous channel. Voters cast the desired ballots, specially enveloped, to the respective ECs during a single non-anonymous voting session. In this way, every EC accumulates enveloped ballots in its own ballot box. The desired anonymity is provided by shuffling ballot boxes a number of times. The process of shuffling ballot boxes is implemented at the end of the election by a set of mobile agents (shuffling agents).

Although [RB99] fulfils all commonly accepted security requirements, it has an important lack of safety: it is not fault-tolerant. If a host running the ballot receiving process fails, the whole voting process results interrupted and it is not possible to resume it. Voters who have not yet voted cannot do it after this interruption and all ballots received are lost. The shuffling process of ballot boxes done by shuffling agents is neither fault-tolerant. Shuffling agents move following a prefixed static itinerary of hosts. If one of these hosts fails, the shuffling processes using that host cannot continue, and their ballot boxes cannot

be tallied. Recent proposals of anonymous communication channels that do not need a prefixed itinerary (such as [Abe98]), may be used as a possible solution to this problem. Other proposals that try to protect this prefixed itinerary ([BRSR99]) may be also used. However, on both cases the number of cryptographic mechanisms is considerably increased.

We propose a protocol to solve the lack of safety described above. We redesign the voting scheme of [RB99] extending the use of agents to all voting phases. We break off the connection between software agents performing the voting process and the physical systems hosting them.

We reach fault-tolerance by using two levels of agents. The agents in the lower level are those related to electoral colleges (Electoral College Agents, ECA). The relationship between voters and the voting scheme does not depend on one host. In our scheme, each ECA, responsible for receiving ballots, can be executed on different hosts. If a host fails, the voting operation is not suspended, but the ECA is re-launched to a new host. The scheme is reconfigured and the voting operation resumes. A similar mechanism allows a fault-tolerant shuffling process under all circumstances. The agents in the higher level (Coordination Agents, CA), allow scheme reconfiguration. We adapt the persistence method introduced in [SRM98] to assure the continuous running of coordination agents.

5.1.2 Sketch of our Previous Voting Scheme

In [RB99] we propose a large scale voting scheme based on the concurrent operation of multiple electronic electoral colleges, with no need for independent anonymous channels. Anonymity of ballots is assured by shuffling ballot boxes by a set of mobile agents acting on behalf of a central voting authority (figure 5.1). The voting scheme is totally self-contained and suitable to be implemented over large internets.

Ballot collecting capabilities are distributed among a set of hosts called Electoral Colleges. These ECs are on the bottom of a functionally hierarchical structure of elements. Each EC host has a ballot box to store ballots casted by their voters. The Counting Centres (CC), on a higher level, reconstruct the global tally from ECs' local tallies. The Electoral Authority (EA) is located at the top of the scheme. The EA is in charge of the whole election. An Information Directory (ID) stores the electoral roll as well as other information

and keeps it always accessible.

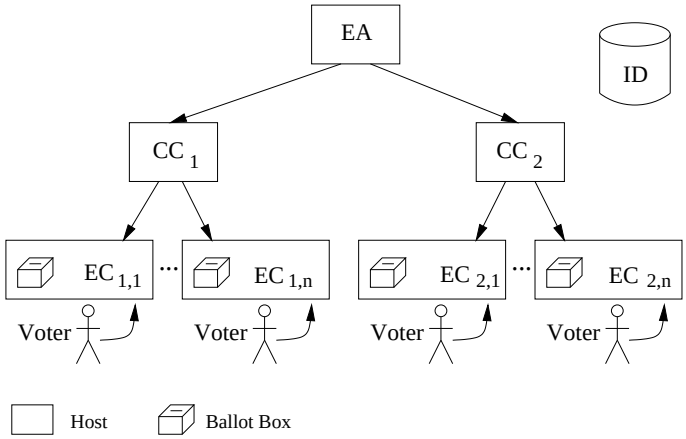

Figure 5.1: Voting scheme of [RB99]

Operation

Operation of [RB99] is divided into three phases: preliminary phase, voting phase and shuffling phase. In the preliminary phase all initial information is generated, including the electoral roll and all the needed cryptographic keys. Electoral Colleges get their keys during this phase.

During the voting phase, each EC is responsible for collecting ballots from a reduced set of registered voters. Every voter contacts his or her corresponding EC to cast the desired ballot in a single session. A five-step protocol is performed between the voter and the EC to assure the security requirements. Ballots are stored in the ballot box.

Finally, during the shuffling phase a set of mobile agents (Shuffling Agents) are launched by the EA. These agents have a static list of ECs. They go through this prefixed sequence of hosts shuffling ballot boxes to keep anonymity. When all ballots are shuffled, ECs decrypt ballots and tally them. Results are sent to the CCs in order to reconstruct the global tally. The final election results are published using the Information Directory.

Safety Issues

Although [RB99] fulfils all commonly accepted security requirements, there are two critical situations where it is specially weak: when a host fails during the voting phase, and when a host fails in the shuffling phase. Both result in the failure of the whole voting process and the losing of all the ballots in the ballot box.

- **Host failing in the voting phase**

  When an Electoral College host fails, voters cannot interact with it any more. The static structure of the scheme makes this situation irretrievable and ballots contained in the ballot box of this EC are definitely lost. Even if the host is repaired, the election cannot be resumed since private information of the EC has also been lost. The only solution is to restart the voting process because if not it would be blocked forever.

- **Host failing in the shuffling phase**

  Shuffling agents (SA) in the scheme of [RB99] follow sequential itineraries, beginning and ending in the initial host they have been launched to. These itineraries are permutations of the ECs list, settled by the Electoral Authority before the shuffling phase starts. This fixed sequential list of hosts makes the system weak in case of host failing: only the host running the SA is able to know the next hop in the itinerary.

  When the shuffling phase starts, SAs are launched to ECs. If an EC is not responding, the SA cannot start. Moreover, all SA involved in a shuffling sequence which contains the failed EC will be stopped, and the voting scheme is blocked. Although the failed EC may resume its operation, the ballot box and private information have been already lost and the voting process cannot continue.

  If an EC host fails during the shuffling phase, the Shuffling Agent executing on it disappears. The ballot box being shuffled by the SA is also lost, as well as the shuffling sequence of the SA.

Most of this safety problems are related to the static structure used in such scheme. This lack of dynamism unavoidably involves a fail vulnerable system.

Some problems arise when using static Electoral Colleges. The most important one is that the entire system depends on each EC, and they cannot be moved to another hardware in case of failure. Before any fault-tolerant mechanism could be used, we need to obtain EC/hardware independence.

5.1.3 A Safe Structure

To solve these safety problems, we apply agent technology to the global design in order to achieve a totally dynamic structure, as shown in figure 5.2. Our solution is compounded of three elements, which are fully described in next sections.

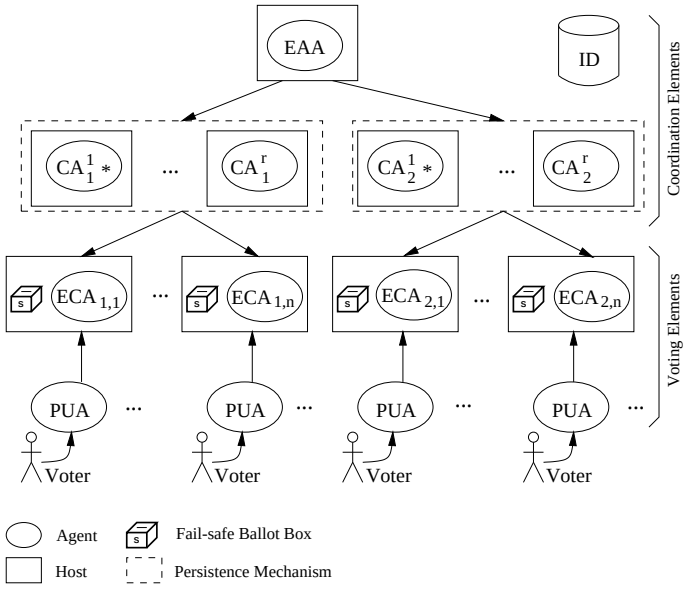

Figure 5.2: Fault-tolerant Voting Scheme

First, we need a set of hosts and agents. All agents are logically arranged in a hierarchy over a network of hosts. Some of these agents are dedicated to ballot receiving tasks. These are the “Voting Elements”. Other agents do coordination tasks to assure that the voting scheme will always be working. These are the “Coordination Elements”.

The second component of our solution is the Information Directory (ID). This directory, based in ITU-T X.500 standard ([ITU93]), stores the information related with all elements

mentioned above, as well as the electoral roll. This directory is accessed through LDAP [YHK95].

Last component of our scheme consists of two fault-tolerant mechanisms. The first mechanism provides persistence of coordination elements and it is based on [SRM98]. The second mechanism provides reliability of voting elements and it is first introduced in this paper. With these mechanisms, the voting scheme uses the low level network of hosts to manage a high-level dynamic fault-tolerant network of electoral agents.

Each host is made up by an execution environment that allows such host to execute mobile agents. For instance, [SBH96] or [IKV00] agent systems can be used to achieve this.

Voting Elements

Voting Hosts. There is sensitive information in an EC, such as private keys or ballot boxes, that cannot be moved from one host to another. This information seems to bind the host with the EC, and therefore with voters. Our goal is to break this binding. In order to have a dynamic voting scheme we need a persistent information system in voting hosts. All voting hosts have a Ballot Box. This Ballot Box must be stored in a persistent device since it contains sensitive information (voter's ballots) irretrievable in case of host failure. RAID technology [PCGK89] can be used to provide this persistent information device. We assume that all information in the Ballot Box is cryptographically protected to preserve its secrecy and authenticity.

Proxy User Agent (PUA). This agent acts on behalf of the voter. It knows voter's choice and interacts with an Electoral College Agent. This agent is always on the voter side.

Electoral College Agent (ECA). ECA stores encrypted ballots from PUAs in the ballot box. Each ECA is linked to a host. It keeps a unique cryptographic key that remains still in the host. This agent provides an interface between the ballot box contained in the persistent information device of the host and other agents, as only such ECA knows how to handle this information. This agent gets the ballot from the Proxy User Agent and puts it in the ballot box. ECA tallies the final shuffled ballot box and sends the result to a Coordination Agent.

Shuffling Agent (SA). This agent moves from one host to another performing ballot box shuffling. There is a SA for each ballot box in the system, and they are the core mechanism of the anonymity system used.

Electoral Colleges in the original scheme have been replaced by a host, an Electoral College Agent, and a Tallying Agent. This dynamic version of the EC provides more flexibility to the system as described in section 5.1.4.

Coordination Elements

Electoral Authority Agent (EAA). This is a global authority of permanent nature. Functions of this agent are similar to those of the Electoral Authority in [RB99], such as launching shuffling agents. In this new scheme it has a more active role, since new agents may be launched at any time. The EAA has an abstracted view of the physical network, using Coordination Agents and Information Directory (see subsection 5.1.3) as interface.

Coordination Agent (CA). The Coordination Agent (CA) is the key element in the new scheme. It controls the connection between the low level network of hosts and ECAs. It coordinates reliability mechanisms to deal with host failure and to assure Information Directory coherence. It also updates the Information Directory when the information of an ECA changes. Finally, it also coordinates global tally reconstruction.

All agent activity is monitored by the CA, so it knows when an ECA or an SA does not respond.

Information Directory

All elements in the system have an entry in the Information Directory. The basic structure of this directory is the same as that of the directory used in [RBR97] to allow large scale elections by coordinating electoral colleges. In this section we only describe the differences from [RBR97], introduced to provide fault-tolerant mechanisms in the voting scheme.

The entries in the Information Directory are hierarchically stored, corresponding to the functional dependences in the scheme as shown in figure 5.3. The EAA is on the top of this

structure. Under this node, CA entries are found. ECAs are in a lower level. A field in a CA entry indicates which ECAs is controlling.

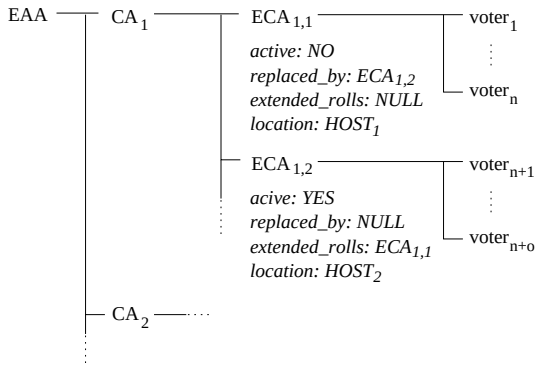

Figure 5.3: Information Directory

The leaves of this tree-like structure are the voters. The set of all voter entries makes the electoral roll. A subset of this electoral roll is assigned to each ECA, so only voters assigned to a specific ECA could vote using it.

ECA's entry has four fields used by fault-tolerant mechanisms: *active*, *replaced_by*, *extended_rolls* and *location*. The *active* field is a flag indicating whether the ECA is running or not. In case of host failure this flag is set off. *replaced_by* contains the name of the substitute ECA in case of ECA not active. The *extended_rolls* field is a list of all subsets of the electoral roll that is managing the ECA. The fourth field on the ECA entry is *location*. This field holds the physical location of the element. In the original scheme each Electoral College was on a fixed host. We break now this hard link between hardware and software, being the *location* field the only connection between a host and an ECA.

Figure 5.3 shows part of the directory after a host has failed: $EC_{1,1}$ is set as *no active*, as this agent disappeared when the host failed, and the other fields are fulfilled to allow $ECA_{1,2}$ to replace it.

Note that because of the hierarchical arrangement of elements, each voter can be identified with a path from the top of the tree to his/her corresponding leaf. This path, called Distinguished Name, contains the name of the ECA assigned to the voter, the name of the CA and the name of the EAA.

Fault Tolerant Mechanisms

In order to have a fault-tolerant voting scheme, we need to assure CA persistence and ECA reliability. The mechanisms we use to achieve both properties are described below.

CA persistence. A fault-tolerant protocol is used to assure CAs persistence. This protocol was first introduced in [SRM98]. With this protocol, agents are performed in so-called stages. Each stage consists of a number of hosts all having images (replicas) of the agent. One of this hosts executes the CA while the other hosts in the stage monitor its execution. In case of host/agent failure, a previously chosen host in its stage executes its CA replica. The stage used for each CA is decided by the Electoral Authority Agent. Stages are represented by slashed boxes in figure 5.2. Replicas are denoted by super-indexes. On each stage the active replica is pointed out by a star (*).

With this scheme, hosts on a stage are not dedicated to it. Hosts may be on several stages at a time, even executing several Coordination Agents.

ECA reliability. When a voting host fails, the ECA agent suddenly disappears. When this situation is found out, CA redirects voters to other ECA. We describe this process in section 5.1.4. The ballot box is safely stored on the persistent data device of the host, so it can be retrieved later. The system transfers the functionality of the failed ECA to an existing (or new) ECA on other host.

With this mechanism, voter has a new ready ECA to vote after a short time, all previous casted ballots are safely stored and, therefore, the election can continue.

We use the same mechanism to deal with host failure in the shuffling phase. In this case, when a host fails its ECA moves to other host as described in section 5.1.4.

5.1.4 Voting Scheme Operation

We have redesigned the operation of the voting scheme described in [RB99] to include safety enhancements. The operation of the voting scheme is divided into three phases. We describe these phases below.

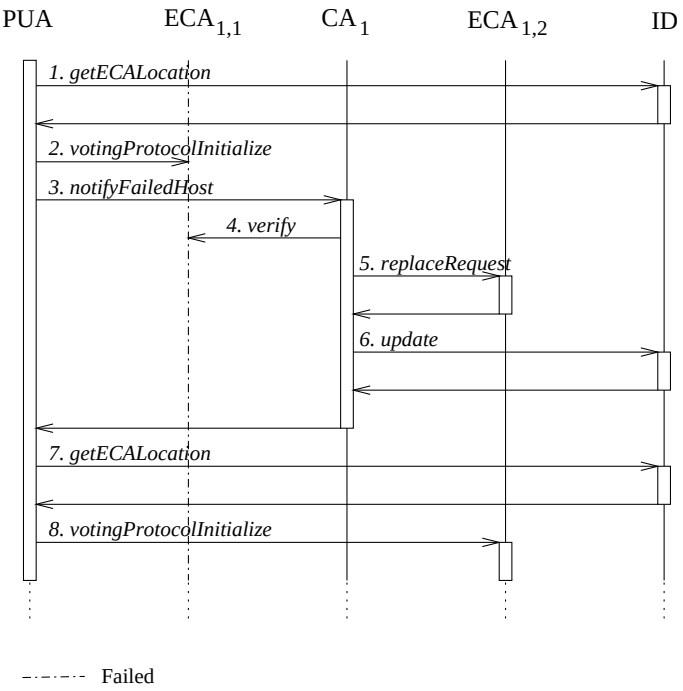

Figure 5.4: ECA Replacement Mechanism in Voting Phase

Preliminary Phase

We assume that Electoral Authority Agent and Coordination Agents are already running when the voting starts.

The first step to be undertaken during the preliminary phase consists in the creation of the Information Directory previously described. It is the task of the EAA to decide how many CAs the system will have, how many ECAs will be operated, and which ECA each voter will be registered to.

When the Information Directory is ready, the EAA enquires CAs for the most suitable set of hosts to run Electoral College Agents. Since CAs monitor their adjacent part of the net, they know exactly where is the best place for ECAs. Then, EAA launches ECAs altogether with their private keys to their respective hosts. If a host fails these keys are lost although a copy always remains in the EAA. These keys are used to establish secret and authentic communications with PUAs and to protect anonymity. These mechanisms are

fully described in [RB99].

Voting Phase

During the voting phase, each voter contacts the host running his/her assigned ECA. This information is retrieved from the Information Directory (ID): the identification of the voter contains the name of his or her ECA, and the host executing this agent is stored in the directory. After this, the voter casts the desired ballot to the ECA in a single session. The voting protocol executed during these voting sessions is described in detail in [RB99]. Ballots are stored in the Ballot Box.

If a host executing an ECA fails, ECA private keys are lost and the host is no longer available. Figure 5.4 is an UML sequence diagram showing how the ECA replacement operation is performed.

When a new PUA cannot connect to a host (step 2 in figure 5.4), it informs his/her CA about the situation (step 3). The CA verifies the situation and (step 4) it rearranges the voting elements, so it is possible to continue the normal operation. First, CA checks by polling if there is an active host running a new ECA (ECA_n) to substitute the failed ECA (ECA_f) (step 5). If a suitable host is found, the Information Directory is modified (step 6) and the voters will find ECA_n (steps 7 & 8): *active* flag in the directory entry of ECA_o is set off. Its *replaced_by* field is fulfilled with ECA_n name and the *extended_rolls* field of ECA_n is fulfilled with the name of ECA_o. Thus, *extended_rolls* allows the system to merge existing subsets of the electoral roll. Host of ECA_n is found in the *location* field. Figure 5.3 shows ECAs entries after an ECA replacement.

When there is not a host that can carry out the job of the failed one, a new ECA is created. To do this, CA informs the EAA who sends an ECA to the specified host. A new ECA entry is added in the Information Directory. The *replaced_by* field of the failed ECA is set off and its *replaced_by* field is fulfilled with the new ECA. The name of the failed ECA is stored in the *extended_rolls* field of the new ECA.

With this updates in the Information Directory the voters of the failed ECA that have not voted yet are redirected to the substitute ECA, and this will accept them.

Private keys of the ECA in the failed host are lost. Note that these keys are not needed by its substitute: if an existent ECA replaces the failed agent, it uses its own keys; and

when a new ECA is created to replace the former, it gets new keys from the Electoral Authority Agent. Since the ballot box remains in the failed host and it will not be used till the shuffling phase, private keys are only needed by the Electoral Authority Agent, who already owns them.

Shuffling Phase

At the end of the voting phase, EAA launches ECAs to those hosts that have failed during the voting phase and have been re-established. Then, EAA launches Shuffling Agents (SA) to all hosts running an ECA. SAs carry private keys using the cryptographic scheme described in [RB99], so they can decrypt and shuffle ballot boxes to assure anonymity.

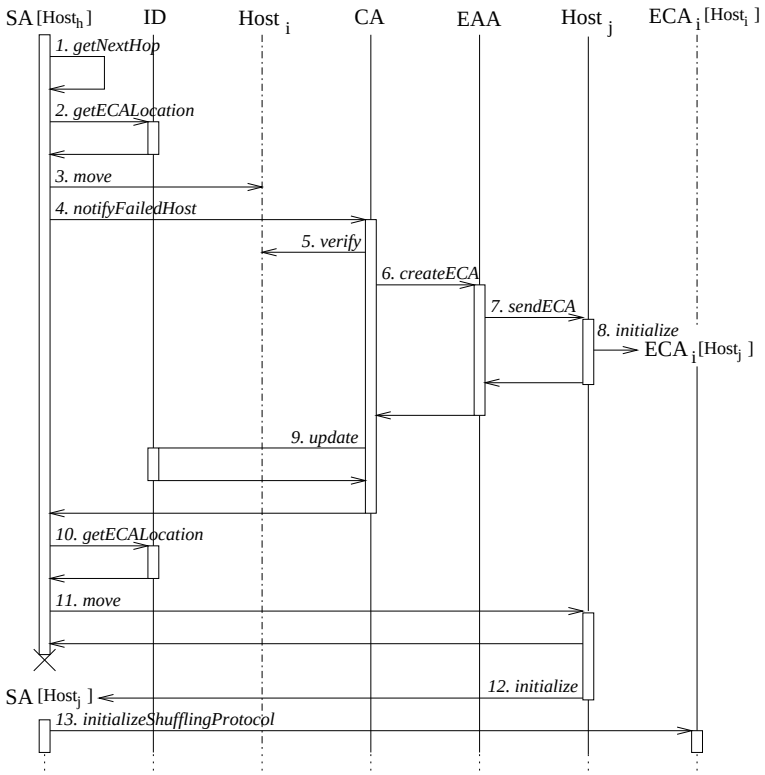

Figure 5.5: ECA Replacement Mechanism in Shuffling Phase

Those hosts that have failed in the voting phase and are not re-established by this phase

are specially handled. Human operation is required to recover their ballot boxes and tally the ballots. These results will be merged in the global tally reconstruction.

As described in section 5.1.2, SAs follow static itineraries. Whilst this static itinerary meant a static sequence of hosts in [RB99], now we have broken off this connection. If the next ECA in the itinerary does not respond because the host has failed, an ECA replacement mechanism is used. Figure 5.5 shows an UML sequence diagram of this mechanism. In particular, it shows the case of SA trying to move to $Host_i$ when it has failed (also ECA_i). We have slightly modified the standard diagram to allow the picturing of a multi-agent system. *SA[*$Host_h$*]* stands for a Shuffling Agent running on host $Host_h$.

SA gets the next hop of the itinerary (step 1 in figure 5.5) and then it obtains the location (host) of the ECA by consulting the Information Directory (step 2). Then, SA tries to move to the new host. If this host has failed (step 3), SA informs the CA (step 4). After verifying that the host has really failed (step 5), the CA informs the EAA about the situation (step 6). In this step, CA gives EAA a suitable location to host the failed ECA. A replica of the missing ECA is sent to the new host by the EAA (step 7) and it is initialised (step 8). CA updates then the ID (step 9). After this, SA tries again to get ECA location from the ID (step 10). As the Information Directory has been updated, SA obtains the new location of the ECA. SA can move now to the host (step 11 and 12) and initiate the shuffling protocol with the correct ECA (step 13).

If a host fails while a SA is running on it, CA notices it and notifies EAA about this. Then, A new SA is launched to the original ECA. Note that this mechanism is feasible since each original ballot box if preserved by ECAs.

At the end of the shuffling phase, when SA is again with its initial ECA, the ballot box is decrypted and stored. Finally, each ECA tallies its ballot box and sends the result to the CA. CAs coordinate tally reconstruction and final tally is stored in the Information Directory.

5.1.5 Discussion

In this section we have presented a secure and fault-tolerant voting scheme. We started from [RB99], a voting scheme that fulfils the most widely accepted security requirements

for a secure election. We have redesigned this scheme by using mobile agents to achieve a dynamic structure. Finally, we have added a persistence mechanism to Coordination Agents and a reliability mechanism to Electoral College Agents. The result is a secure and fault-tolerant voting scheme that allows an election to be easily reorganised and resumed in case of host failure.

Persistence mechanism for CAs uses the stage concept introduced in [SRM98]. A set of hosts is needed for each persistent agent to implement a stage. Although it seems that the number of required hosts has been increased, these hosts can be shared by different stages. On large schemes, differences on number of hosts between a non-fault-tolerant version and our proposal are not significant. The only extra overhead in these hosts is the number of running processes: for each Counting Centre in the original scheme there are n Coordination Agents in the fault-tolerant proposal.

Regarding communication costs, both persistence and reliability mechanisms require monitoring. Since this monitoring can be achieved by using a simple polling protocol over UDP, network overloading is negligible. A more complex protocol is used to perform the agent reliability mechanism, as it involves communications between all scheme components. This protocol is only used when there is a host failure and it has seven steps in the voting phase and twelve in the shuffling phase, which is a reasonable cost for resuming the election.

Mechanisms used in this paper to achieve a fault-tolerant scheme can be also used to solve similar problems in other applications. For instance, we are using the ECA/CA safety architecture (persistence and reliability) to implement fault-tolerant Connection Agents in connection admission to telecommunication networks schemes([BCH$^+$00]). These mechanisms can be considered to be added into a mechanisms repository to be used using our trust model based methodology of chapter 4 to provide security solutions to mobile agent systems.

5.2 A non-repudiation protocol to secure the itinerary of mobile agents

Mobile agents are software units basically consisting of code, data and itinerary. Mobile agent systems are platforms that allow mobile agents to autonomously migrate between different hosts. These hosts offer the runtime environment for the mobile agents. All these concepts are introduced in chapter 2.

5.2.1 Our proposal

To protect the itinerary of mobile agents we propose a cryptographic solution which assumes the existence of a Public Key Infrastructure (PKI) and a Trusted Authority (TA). Every host H_i is provided with an asymmetric key pair with the public component adequately certified. We denote host H_i's public and private keys as, resp., P_i and S_i. The public certificates (and the certificate revocation lists) are hold in a suitable directory service. Figure 5.6 depicts this environment.

The TA launches a number of agents that have to be executed at several hosts, following a prearranged sequence. This sequence constitutes the agents' itinerary. In addition, every agent carries a set of data that may be manipulated by the hosts of the itinerary.

We consider therefore each mobile agent as a triplet:

mobileAgent = (code, itinerary, data).

The itinerary is a *sequence* of n encrypted simple entries E_i, randomly sorted. Following the notation introduced in previous section, the itinerary would be denoted as:

$$i = [E_k, E_l, \ldots, E_t], k, l, t \in (1, \ldots, n)$$

where

$$E_i = (\text{host}_i, \text{private information for host}_i, \text{host}_{i+1})^{P_i}$$

The TA launches a mobile agent to the first host in the itinerary. The host retrieves the agent's itinerary and tries to decrypt each of the entries until its own identification is

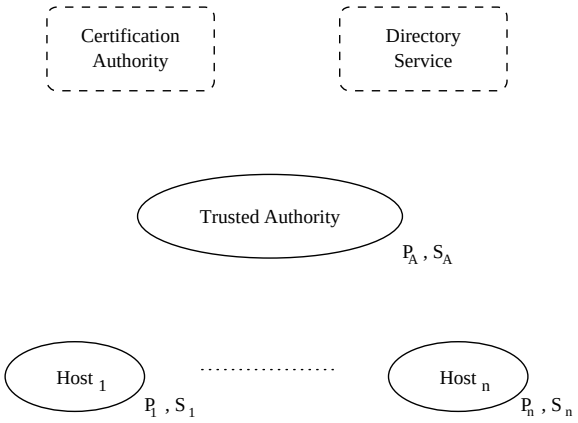

Figure 5.6: Environment

found in one of the decrypted entries. From this entry, the host is able to read its private information together with the address of the next host to be visited by the agent. The private information may contain a private key needed to perform some operation on the data carried by the mobile agent. This process takes place at each visited host. The last host of the itinerary is responsible for delivering the agent back to the TA, together with the results obtained at all visited hosts.

To configure the itinerary of each mobile agent as a set of encrypted entries in random order, has the advantage of preventing every host from knowing the identities of other hosts of the itinerary, except previous and next hosts. This reduces drastically the chances of fraudulent collusions of malicious hosts.

Whenever a mobile agent is passed from one host to the next, a non-repudiation protocol [ZG96] is executed between both hosts. Non-repuditation of origin and non-repudiation of receipt services can be used to determine the origin of an attack if the itinerary of the mobile agent is altered. Figure 5.7 shows this non-repudiation protocol. In the figure, P_R and S_R are an asymmetric key pair randomly generated by host H_i. Also, K is a symmetric key randomly generated by host H_i.

An accurate discussion of the operation of this protocol can be found in [RBR98]. Briefly, host H_i forwards the mobile agent to host H_{i+1}, encrypted with key K. If and only if H_{i+1} replies with the receipt of reception in step two of the protocol, host H_i discloses

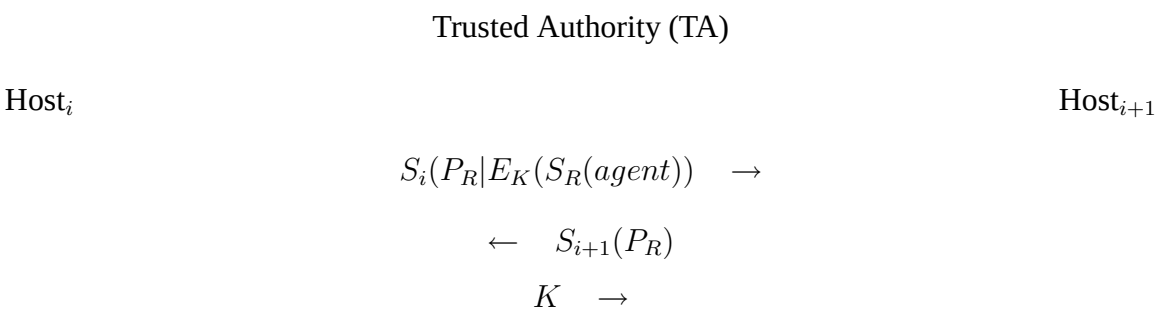

Figure 5.7: Non-repudiation protocol

key K in step three. All interchanged information has to be securely stored by both hosts.

The TA is not involved in the execution of the protocol, unless this is required by one of the hosts. In such case, the TA is able to determine which host is attacking the itinerary of the agent. Either the attacker is H_{i+1}, by stopping the protocol at step two, or the attacker is H_i by stopping the protocol at step three (or by not starting the protocol at step one).

Moreover, the TA can eventually take the initiative of verifying the whole itinerary of the mobile agent, or some of its steps. Since each step is individually verifiable and the responsibilities regarding the agent moves are clear in all steps, hosts are strongly discouraged to disrupt the agent's itinerary.

In this section we have described a non-repudiation protocol for protecting the itinerary of agents. This protocol can be used in the MARISM-A platform as a mechanism to provide itinerary protection during agent migration.

5.3 Design of a Secure Multi-Agent Marketplace for 3G telecommunication networks

In this section we describe a security framework for a multi-agent system designed to manage resources in future mobile communications networks. This framework is based on a very simple general trust model. The multi-agent system is being developed as part of the IST SHUFFLE project. A business model appropriate for selling of bandwidth resource and

services is investigated and mechanisms to achieve a Global Trust Model to support User Agent to Service Provider Agent and Service Provider Agent to Network Provider Agent negotiation is outlined. Our trust model for the marketplace is based on concentric spheres structure. The core of this onion-like model will be physical security and it is assumed that this is provided. A security infrastructure is located in middle spheres: the internal and the external security infrastructure. In outer spheres we will use complex aspects of trust such as fairness, reliability, reputation and loyalty to provide a complete model of basic trust for marketplaces. Many aspects of the model generalise to a common secure auction model to be used by e-Business and business to business applications.

5.3.1 Introduction

The use of agent systems to support the management of resources in telecommunications networks was first investigated in the IMPACT and FACTS projects [BHBR01]. The former successfully demonstrated the use of agents to perform real time connections to an ATM network in the context of multiple service providers and user specified (explicitly or implicitly) QoS requirements. Security issues associated with the IMPACT business model have been addressed in [BCH$^+$00]. However, the business models associated with managing future mobile communications networks raise other issues and these are the subject of this paper. A model for the creation of a secure marketplace for bandwidth transactions is outlined, and how this generalises to service oriented transactions in a generic marketplace is indicated. The marketplace, as described, is in itself a multi-agent system providing a secure e- trading and e-commerce centre oriented to business to business activities.

The resource management system can be viewed as three layers, namely the facilities layer, the negotiation layer and the resource layer. At the base is the resource plane. Since resource management is a real time application survivability and reliability are an important design goal. In the resource plane all the network provider agents in an interaction belong to the same network provider. Agents that communicate with agents not owned by the same entity are in the negotiation plane. These autonomous agents communicate with each other using the trusted services provided by the facilities plane. These services

allow agents to perform e- trading between themselves as well as establishing indirect trading relationships through the facilities in the top layer, like auctions. In next section the concept of marketplace and instances of a marketplace is introduced. The layers in each marketplace instance are elaborated. A method through which trust can be built into the system using the facilities layer is described. Possible business models for the provision of bandwidth and services in a multi service provider and network provider environment are given. Section 5.3.3 is devoted to security issues. In section 5.3.4 the negotiation plane is described in detail. Whilst many of the facilities plane concepts apply to many application domains, the negotiation plane described is particular to the domain of resource management for mobile communications networks. The need for fairness is introduced and an illustration of the use of method through which the facilities layer can support fairness in resource allocation is given confidence.

5.3.2 Marketplace

The purpose of this section is to briefly describe a non- discriminatory, secure environment in which an multi-agent system could operate. This system, which we call the marketplace exists to provide a secure trading environment in which agents may trade. In our example of the marketplace, the commodity being traded is bandwidth. Due to the distributed nature of this commodity, the marketplace is instantiated thousands of times. This section will first describe the business environment motivating the marketplace and then provide a more detailed description of it. Finally, it will address both the issues of trust and security within the marketplace

Business Environment

As in any business arrangement, each party has its' own interests in mind. Resources (typically bandwidth) are provided by Network Providers (NPs) who own the physical network. The resources are sold to customers through Service Providers (SPs). A Network Provider may also act as a Service Provider in its own right. Such a SP is much like any other but, depending upon regulatory constraints, may have more control options than other

SPs. The liberalisation of telecommunications networks is expected to lead to an "any-to-any" scenario with any customer being free to buy services from any SP, who in turn could buy network capacity from any NP.

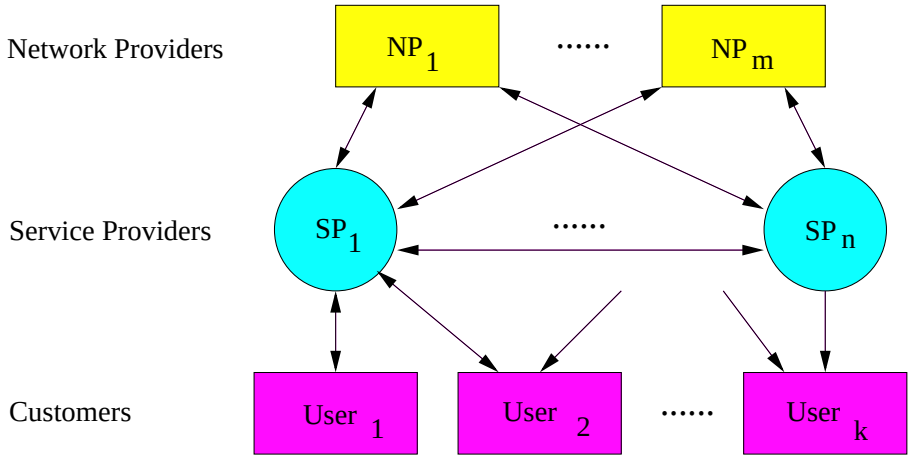

Figure 5.8: Scenario

The scenario shown in figure 5.8 is the most general case and particularly relevant for the future evolution of third to fourth generation mobile networks. Whilst the current scenario is less flexible, mainly allowing customers access to only one SP (on one terminal) who in turn buys network capacity from one NP. We feel that the "single SP per customer" model is on the decline. Examples of this trend are that with the fixed telephony service, customers can select a different SP on a call by call basis, and WAP services are available from any service provider, just as the Internet user is free to browse any web site. SP to SP exchange is also possible. This has the advantage of allowing SPs to offload excess bandwidth to others who may want it and another way to buy bandwidth other than directly through a NP. A mechanism similar to that used in bond markets has been proposed by Bourne [Bou00] for SP-SP trading. Such medium to long-term contracts bring predictability, stability and scalability to the market.

One method for meeting the objectives of the different parties is for the parties to enter into a contractual arrangement called a Service Level Agreement (SLA). The areas a SLA could cover are dependent on both the applicability and importance of a certain topic to the

two parties bound by the agreement. Two kinds of SLA are relevant, namely the Customer-SP SLA and the SP-NP SLA. For brevity we concentrate on the SP-NP interaction (i.e. business to business) though much of what is said applies to both. Factors that could be defined in a SLA as well as sample SLAs and some URLs of some publicly available SLAs are given in [SHU01] deliverable 4 of the project SHUFFLE. Currently, the criteria used in defining a SLA is subjective and lacks clarity. For agent interpretation, more careful and detailed specification is required. It is assumed here that SLAs will be agreed between trading parties, that they have a standard form and that they can be downloaded and individual parts interpreted.

Marketplace Model

The introduction of agents goes beyond the interaction between the customer and the service provider. As shown by the European Union ACTS projects IMPACT and FACTS, there is a significant role for agents to play in managing the resources within and between SPs and NPs. Combining the control of the resources of both the SP and the NP leads us to propose an outline three layer marketplace architecture:

- The facilities plane houses the entities that ensure secure interactions in the negotiation plane. Issues such as SP registration and key distribution feature in this plane. Confidence (as determined by the facilities provided) is an important concept at this level. Facilities are provided to allow the agents in the negotiation plane to participate in indirect trading relationships, such as blind auctions, as well as infrastructure services. We will reserve trust to refer to the whole model built up from confidence, reputation, loyalty, fairness, reliability and survivability.

- The negotiation plane is where all interaction between the customers, the SPs and the NPs takes place. In this plane the service provider winning the business sets up the connection using a network operator of its choosing. This layer offers an interface for the agents to communicate with each other as well as with the services offered by adjacent layers. Issues such as business models and SLAs feature in this plane. Reputation and the maintenance of statistics for measuring reputation is an important concept at this level to provide fast decision making.

- The resource plane is where the network operator manages its resources both across and within individual radio cells. Reliability and survivability are important operational considerations at this level. This layer lodges all low level network services including handover and cell selection management, cell size management and offloads to other networks. It provides an interface to the other layers. Since the internals of this layer are out of the scope of this paper, we will not focus on it, but depend on the interface to its services. This layer represents the "goods" layer of the marketplace.

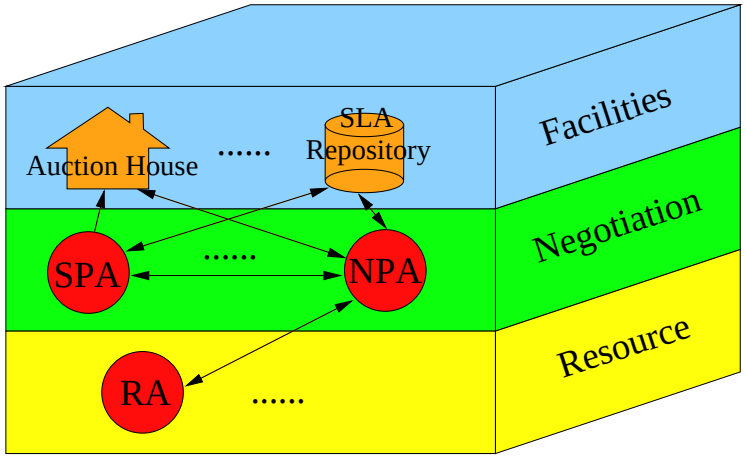

Figure 5.9: Marketplace

Splitting the system into these three planes supports the definition of well-defined interfaces. Different services are then more independent providing flexibility (as behaviour of services may change), and adaptability (new services can be added without modifying the design). Figure 5.9 shows the set up of the marketplace.

Multi-instanced marketplace

The marketplace analysed in this document can be a single marketplace or a network composed of instances of a marketplace template. In the telecommunications scenario described, the latter is the case. Each marketplace instance is situated in a different base station. Thus, the marketplace in this case consists of a set of independent synchronised marketplace instances (synchronised in the sense that if a norm is changed, each marketplace agent has its norm changed), sharing all the basic norms defined by the institution

and the members. A clear analogy exists with the object oriented paradigm terminology: the marketplace is like a class and the instances like objects of that class. Each Regional Network Controller (RNC) owned by a NP has an instance of the marketplace (we will also refer to this instance as the marketplace for the rest of the paper when meaning is clear) with the same members (buying and selling agents and facilities). As the example of a customer request later shows, adding a new agent or facility involves an interaction with the marketplace as a whole, not only with one instance. This cloning of the marketplace ensures location independence of the institution and fair threading of members (e.g. the agent provided by a SP will have the same set of capabilities in every instance of the marketplace in which it exists.

The aforementioned concept of marketplace with instances can be implemented in two ways: distributed or centralised. In the distributed approach each instance is considered to be part of the marketplace, and several mechanisms allow information to be spread and operations to be performed (for instance, adding a new member). Using the second option, a centralised version of the marketplace (as an institution, not as an instance) exists. Both alternatives have pros and cons. The main advantage of using the distributed version relies in the decentralised control of marketplace external operations, such as the inclusion of new member or norms, and the absence of a special location. A main disadvantage is the lack of control of the private information about the outer PKI. Although several mechanisms exist to distribute private keys [BCH+00] [Rie99], there is the possibility of collusion of NPs owning various RNCs. This situation can be detected a posteriori [Rif93] which allows legal action to be undertaken, but this makes the system more complicated, unnecessarily. On the other hand, an advantage of using a centralised marketplace is that all private information can be in one controlled place (perhaps replicated for robustness). Disadvantages include the need of a special location for the marketplace and the bottleneck of external operations, if these are frequent.

Trust Model

Marketplaces in highly competitive business models should be based on a strong trust model. Trust is recognised by many to be cardinal to information security, security policies, accountability, reliability, business relationships, etc. At this time, there are no satisfactory

answers as to what trust is, no consensus and no well- defined models. However, some references related to trust theory show some results linked to cryptography and certification. The trust model we use, will allow us to represent our understanding of the word "trust" not only in reference to the trust between agent entities, but also "trust" in regards to the system itself. Our trust model for the marketplace is based on a concentric spheres structure. The core of this onion-like model will be physical security and it is assumed that this is provided. We deal with this aspect in section 5.3.3, where security issues surrounding the marketplace are discussed. A security infrastructure is located in middle spheres: the internal and the external security infrastructure. In outer spheres we will use complex aspects of trust such as fairness, reliability, reputation or loyalty to provide a complete model of basic trust for marketplaces. Figure 2 shows a synoptic cube of the trust model.

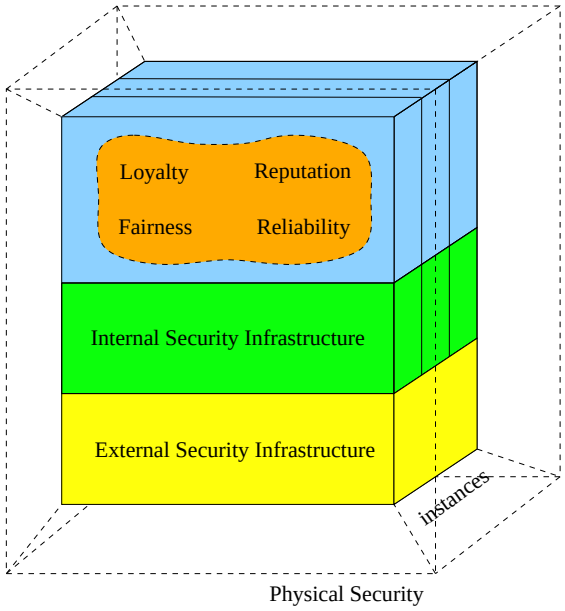

Figure 5.10: Synoptic cube

External Security Infrastructure

The inner sphere of the trust model is the external security infrastructure. This infrastructure is only needed in multi- instanced marketplaces. Since a Public Key Infrastructure (PKI) is a well-known security infrastructure and many security protocols are designed for

it, it can be used as the external security infrastructure.

A PKI basically consists of a set of algorithms to implement public key cryptography, such as RSA, a Certification Authority (CA) to issue certificates, a certificate standard, such as X.509v3, and an information directory to place and distribute public keys, such as X.500 with LDAP access. [Sch96] describes all these PKI elements.

The use of a PKI brings new issues to discuss. Probably, the most important issue is the distribution of certificates: a trade-off must be found between an easy spreading system and a fast mechanism for revoking certificates.

Security can be established through the presentation of third party certificates. This assumes the existence of a publicly known and trusted third party as well as the need to know other agents public keys. In mobile communication networks, a telecommunication regulatory body can be easily imagined as a trusted authority. However, the consequent requirement that a buying or selling agent reveals its identity to participate in trading activity violates what could be considered an agent's right for privacy (for itself and for the user of the agent). [MS00] describes approaches where identity disclosure is not required for trust establishment.

Nevertheless, we will adopt the centralised certification mechanism of a globally known trusted party as it fits well with our mobile communications business environment. In most highly competitive business environments a natural trusted third party can be identified and thus the model of the PKI is suitable. As presented in [BCH$^+$00], a Certification Authority (CA) might be the weak point of the whole PKI based security system. The owner of this CA can break all security in the system, since it is the trust in the infrastructure. Even if we consider a presumed impartial third party, such as a telecommunication regulatory body holding this responsibility, the institution does not perform the administrative procedures directly, but employs technicians to do it. These persons can be bribed to reveal the secret compromising the whole trust model. In previous work, [Rie99] [BCH$^+$00], we have presented a distribute-controlled CA based on a secret sharing scheme. In this scheme it is possible to detect cheaters [Rif93].

How can coherence of the institution, integrity of code and trust be kept if the marketplace is cloned in many places? The external security infrastructure will provide answer for this. A party submitting its agent to the marketplace has to be confident that the agent is not

going to be cheated in the marketplace, but also that the integrity of the agent will persist in all the instances of the marketplace. The solution to this problem is to have a simple PKI to keep the cohesion, in terms of trust, of the marketplace. Who is this trusted third party all marketplaces trust in? In the particular scenario we are managing, the telecommunications regulatory body (e.g. Oftel in the UK) is a good candidate. Members (agents and facilities) can enter particular instances of the marketplace (placed in different RNCs) only from this trusted third party. An implication of this is that all instances must have the public key of the external security infrastructure to verify signed elements and send back encrypted information.

Internal Security Infrastructure

Every instance of the marketplace has its own internal security infrastructure. A Public Key Infrastructure (PKI) like the one presented in the previous section is also used here. As a requirement, the keys of the internal PKI used in every instance must be unknown to the external security infrastructure. This avoids the owner of the marketplace controlling (read cheating) specific instances. This is an added feature of this model of the marketplace with instances comparing to the unique marketplace approach: if the marketplace code is accepted all instances are trustworthy since they control themselves with private PKIs. The Certification Authority of the PKI used in the internal security infrastructure is a key element in the trust model since it is the hard link between this layer and upper layers.

High level trust

In the top layers of the trust model, more complex and high level concepts of trust are managed. These concepts are described in sections about the marketplace architecture. The main aspects include reputation, loyalty, reliability, fairness and confidence. Some of these aspects are related to different layers of the marketplace. For example, the concept of reliability is linked with the resource plane, while reputation or loyalty are linked with the negotiation plane.

The implementation of the trust model in these upper layers of the marketplace (in negotiations, selection processes, auctions, etc.) must be embedded in the language used, since

there exists a high dependency between both. The language proposed as a normative in the FIPA specification [FIP00], namely the FIPA Agent Communication Language (ACL) for inter-agent communication, can be used as a base of such language, but the specification is presently incomplete.

5.3.3 Security issues

The implementation of the marketplace involves the use of a multi-agent system (MAS). We are using FIPA OS [FOS02] as the multi agent system for implementing the marketplace. FIPA, the open standard for heterogeneous agency interaction, has defined a trust model. This model is very weak and is thus inadvisable for use in a real application. The lack of defences against impersonation, faults and bottlenecks are evidence of the early stage of the FIPA trust model.

FIPA also has a standard security model [PC00]. It is based on a PKI, but it is incomplete, not accurately specified and with security faults such as the duplication of private keys, that makes attacks on the system easier. No current FIPA based agent systems report the use of this model. Furthermore, the current FIPA specifications do not provide any specific mechanisms or policy for secure ACL communications. The basic problem in FIPA is the implicit assumption that agents are cooperative and trustworthy and that security is performed elsewhere. This is a common problem [BCH$^+$00].

We have used FIPA-OS to implement the marketplace, but instead of using FIPA security and trust model, we use our own.

Security Issues Surrounding the marketplace

Due to the high competitiveness of usual scenarios of this kind of marketplace (such as the telecommunication scenario described above), its placement is a very sensitive issue. Security issues in applications using multi agent systems (MAS), as the marketplace, are twofold. Firstly, there is the problem of agents attacking other agents of the system. This is a resolved problem and many MAS include solutions to this. Secondly, there is the problem of malicious hosts. Prevent agents from host attacks is still an open problem. Examples of

security issues in the marketplace concerning this are NP owning the MAS of the marketplace (NP has access to the internals of the system as it is placed in the RNC of the NP) and modifying SP agents to spy or manipulate its code, denying services to specified agents or removing some competence agents. Many references relate both problems [Hoh98a]. The malicious host problem is especially relevant in the marketplace: It is very hard to prevent the owner (someone that has access to the system) from manipulating the marketplace, so who owns the MAS of the marketplace? Below we analyse different possibilities to choose the best implementation approach. In a first approach (internal agent architecture) the agents are all placed within a segregated area of the RNC of a NP. The interests of the SP are represented by its Service Provider Negotiating Agent (SPNA) and the interests of the NP by its Network Provider Negotiating Agent (NPNA). The functionality of these agents is outside the scope of this document, but can be found in [SHU01]. The segregation in the RNC is done to ensure that the SPNAs (an example in this domain of a competence agent) do not have any means of accessing restricted areas of the NP. However, this layout does not provide any form of protection to the SP. In this model the SP must trust the host (the NP) as it is easy to see that if the host is malevolent, it could easily alter the memory space of the SP's agent, or any of the aforementioned attacks.

A different approach is the external distributed agent architecture. Each SP provides it's own "agent attachment" to the RNC. This second architecture provides not only segregation, but protection of the SP from the potentially malevolent NP. However, in this layout, it is required that every SP attach it's agent to every RNC of every NP it wishes to do business with. Thus this layout could very easily prove anti-competitive. Imagine the entry of a new SP to the network market. This SP would have first find the capital to physically attach to many RNCs of at least one NP which could prove very costly (production of agent attachment, placement of agent attachment, maintenance of agent attachment). Furthermore, due to hardware limitations there would only be a limited number of agent interfaces at each RNC, thus preventing entry into the market once the capacity of the RNC has been reached. Finally, the external centralised agent approach provides an architecture which protects both the SP's and the NP's interests. In this architecture, an "agent platform" attachment controlled by a trusted authority is placed on each RNC of any NP. In

this manner both the NP and the SP ensure that others only have access to limited information. Furthermore, the SP is protected from malevolent NPs, as the NP does not control the attachment.

5.3.4 Marketplace internal architecture

Facilities plane

The facility plane houses all facilities used by agents to achieve indirect agent to agent trading relationships. We call a facility any entity in the marketplace that provides a set of similar services. A facility also provides a well-defined interface and a set of protocols to be used by agents. Some facilities control the infrastructure of the marketplace. These form the institution of the marketplace. Examples are the registration office and the trusted authority. Other facilities provide the system with basic trading services, such as the services of an auction house. Finally, there are facilities to help agents to perceive the environment better and thus allowing complex architectures to have better decision- making capabilities. Examples are the internal information office or the external information office. Some of the possible facilities on this plane are:

- **Auction House.** This allows agents to participate in auctions. Different types of auctions are supported (Dutch, English etc.) as well as different operation modes (blind seller auction, blind buyer auction etc.). A defined auction protocol and an objective auctioneer keep the auctions fair and safe (No seller refusing to sell, no buyer abandoning after being awarded, etc.)

- **SLA Repository.** Since agreements between agents are a fundamental part of the marketplace in the telecommunications scenario, an agreement (SLA) repository is needed. This institution facility keeps all agreements safely stored and allows them to be consulted in case of dispute. This avoids an agent failing showing its SLA to other agent or institution facility. Keeping all SLAs in the same place makes them easy to maintain as they are stored and accessed more efficiently.

- **Trusted Authority.** This is the central part of the trust model in the marketplace. As described previously, a PKI has been chosen because of the special conditions

in the telecommunication scenario. The Certification Authority (CA) of the internal security infrastructure issues all certificates needed and keeps the public information in a directory. In our marketplace model, the Trusted Authority will be the same entity as the CA. Agents can query public keys from it and other facilities can request certificates.

- **Registration Office and Directory.** The marketplace is a closed environment. Only authorised agents or facilities can be lodged. The registration office registers authorised agents into the marketplace and requests a certificate (to the trusted authority) for them. The registration office can also remove an agent, for instance, when it has expired. Facilities and network services can also be added or removed. The registration office also takes care of name spacing. It provides a directory so it is possible for an agent to know about the other agents currently in the marketplace.

- **Legal Service.** Agents can modify SLAs or even create new agreements. To give legal validity to these documents a notary facility exists also in the facility plane. When the agreement is signed by the notary and by the concerned parties, it becomes a legal document in the marketplace. Before signing an agreement the notary reviews it and verifies if norms are observed. All agreements, including SLA are written in the same agreement flexible language (perhaps XML). Failing to comply with a notary recognised agreement can result in penalties.

- **Internal and External Information Service.** This facility provides the marketplace with a reliable information service that can be used by agents. The internal information may include social indicators. As this facility belongs to the institution, its signed information is reliable for agents. Thus, this information can be used as parameters in different selection functions.

- **Connection Centre.** All traded connections could be submitted to the connection centre. At this point, the network connection could be verified with the norms of the marketplace and with the agreements and finally it is rejected or accepted. In the last case, the connection is made in the network layer and the economic transaction is sent to the financial office. In the case of many small value trades (as in mobile

connections) such a validation, transaction by transaction, is not appropriate. Periodic retrospective monitoring with respect to the SLA and other norms of the market place are necessary, supported by an intrinsically fair mechanism for trading. This would be the role of the connection centre here.

- **Financial Office.** This is where economic transactions are executed. By using this gateway it is possible to use external economic services.

All these facilities are regulated by a set of norms. The norms of the institution are set before any marketplace activity and depends on the scenario.

The marketplace, as described above, is in itself a multi-agent system and a secure e-trading and e-commerce centre oriented to business to business activities.

Negotiation plane

Reputation

Agents that communicate with another entity's agents are in the negotiation plane. These autonomous agents can communicate with each other using the services of the facilities plane. These services allow agents to perform e-trading between themselves as well as establishing indirect trading relationships through the facilities in the top layer, like auctions. The system needs to provide complex protocols to achieve this services. Agents in the system can have complex architectures that will be able to benefit from the rich environment offered by the marketplace. In other application domains, this layer is equivalent to the buyers/sellers layer.

The ability to negotiate in multiple several timescales is essential in many environments, such as in management of telecommunications networks.

Some negotiations will have very tight real time constraints (e.g. SP-NP) and others have more relaxed constraints (e.g. SP-SP). To provide mechanisms that support fast real time decisions we will concentrate on the use of the concept of reputation allied to negotiation.

In a market where there are many small value transactions, there needs to be a method of managing risk when there is not the time or the information to be sure of achieving the

very best deal for each transaction. One tactic is to monitor services over a given time period. During the monitoring period each party will depend on reneging penalties and proportions specified in a SLA to manage services. The outcome of these transactions will feed directly into the building of a reputation model of the other parties. A Customer will have a reputation model of each SP. A SP will have a reputation model of each NP and vice versa. The dual of reputation can be thought of as loyalty. Each party continuously updates its reputation model through inputs such as compliance or lack there of, with the SLA. In the simple procedure described below, reputation of a SP is used by a Customer to select a SP and reputation of a NP is used by a SP to select a NP. If the NP cannot provide what is requested then subsequent connection time bilateral negotiation may be performed to see if some satisfactory agreement can be reached at a compromise QoS. The NP can be bold or cautious in its negotiations with an SP depending on compliance with the relevant SLA to date. Similarly the SP can be bold or cautious depending on the same SLA and its believed reputation with respect to the Customer and the Customer-SP SLA. A SP is unlikely to know exactly its reputation with a Customer. So the SP needs to model what it believes its reputation to be with respect to a Customer. Reputation can also affect NP behaviour if the non-provision penalties are not too high, the tolerance specified in the SLA is within bounds and the SP has a low reputation with respect to the NP then the NP may consider taking a chance of offering the QoS without reserving the resource pending confirmation of agreement. If the agreed penalties for non-provision is too high then still many possible connections will not be bid for, or wasteful reservations made, under utilising the resource, so overall performance of the system depends on the contents of the SLAs.

The selection by the SP is based on maximising utility over time, but its' perceived reputation with the Customer will drive it to use the best performing NP, i.e. the NP with the best reputation. There is nothing to stop a service provider who is owned by a specific network provider (NSP) from always choosing its owning NP first. As a business decision a NSP may select its owning NP first and then if the required QoS cannot be offered immediately without negotiation, selecting a NP with the best reputation based on its models. It could be itself. This could also improve the stability of the system as discussed later.

Negotiation has value as it is in the interest of both parties to compromise when resource is very scarce. This is because the cross correlation of demands for service at different SPs

and different NPs over time is high. So the probability of succeeding with another SP (or NP) when demand is so high that resource is a problem (the only time of interest) is not great. With negotiation a SP (or customer) makes a connection that could not otherwise be made. The SP makes concessions in the context of the Customer- SP SLA and service required (e.g. it may accept a reduced quality video link in an audio visual connection for a period of time.) A NP uses bandwidth that may not be otherwise sellable because it is not sufficient for full QoS. It can offer lower bit-rates, make changes in modes of hand-over, etc. Agreement in negotiation leads to rewards in terms of reputation that can be better than no agreement.

For example, a SP will select a NP based on the reputation of that NP with the SP. If the NP cannot provide the requested QoS then either the SP selects another NP or the SP and NP use what time there is to negotiate genuine trade-offs. The first step, selection by the SP, uses information that is not dependent on the resources at this instant. The information used includes static information and recent performance information of the NP. This information can be updated continually by the SP. The selection gives initial commitment to the chosen NP. In fact if the SP is to have adaptive behaviour then the needs to be some element of randomisation in the selection so that alternative NPs are continually explored, even though the bulk of the traffic may be with one or two NPs. (An alternative is for the SP to periodically to have a kind of auction).

In a very simple model the reputation of NP j with SP i at time t is denoted by a real number $r_{ij}(t)$, $0 < r_{ij} < 1$. If the NP grants the request at time t, $r_{ij}(t-1)$ is incremented by $\delta r_{ij} = \alpha r_{ij}(t-1)$ where $t-1$ is the previous time r_{ij} was computed. Initially for each SP, all NPs are allocated an identical reputation, i.e. for each i, e are all the $r_{ij}(0)$ are all the same, $0 < \alpha < 1$. When resources are in high demand then some concessions have to be made. Strategies and mechanisms to perform these relaxations to improve service levels are out of the scope of this paper, but assume that analysis has provided some relaxation policy. (The policies adopted in SHUFFLE are outlined in [SHU01].) If an agreement cannot be reached the SP may start again by selecting a different NP, or may just not offer a connection to the customer and suffer its penalty in terms of the customer-SP SLA. If no agreement can be reached then the adjustment is $\delta r_{ij} = \beta r_{ij}(t-1)$, $-1 < \beta < 0$. If negotiation reaches an agreement then the adjustment is $\delta r_{ij} = \gamma r_{ij}(t-1)$ where, $\beta < \gamma <$

α and so lies between the positive increment for complete instantaneous satisfaction and the negative decrement for dissatisfaction, depending on how the SP values the agreement. This simple model does not recognise that some NPs may be more reliable for some kinds of request than others or that coverage is not uniform for a NP. A richer model involving request context could be considered, but that is not done here. A richer model of selection and subsequent negotiation would more than one NP to be selected and competing bilateral negotiations take place between different NPs. An offer from a NP has to be accepted there and then, or a counter proposal made. So a SP can agree with an NP (to secure the resource, and pay the corresponding charge from the time of agreement and the time of any renege) and later renege if other concurrent negotiations find a better deal. A penalty for reneging so quickly may have to be paid. Alternative protocols will be investigated.

Selection based on reputation is faster and less wasteful in network bandwidth than an auction or call for proposals, but is it reasonable? Certainly an SP can use NP reputation as feedback to use in NP selection. In this case it pays a NP to perform well and not just perform minimally. Is the customer being treated well? The interests of the SP will possibly bias negotiation by the SP on the customer's behalf, however the SP has to be mindful of its reputation with the Customer not just the constraints of the customer-SP SLA. Importantly, whilst selection does not of course solve the problem of not being able to deliver when resource is not available, it is potentially more efficient than an auction mechanism. In the simplest protocol described it allows the NP to manage its resource on the assumption of award if it meets the required QoS. The dangers of overselling when making offers to many parties concurrently is circumvented and the equal danger of reserving what will be unsold resource is also avoided. Radio is such a scarce resource that this is a real problem. Suppose an auction mechanism were used, then reserving bandwidth in case an offer to sell is accepted is almost out of the question. Not reserving could lead to large numbers of reneges. Finding the balance is difficult because of the short time scales.

Fairness

Fairness can be an issue. For example, since one of the SPs is likely belongs to the a NP (i.e. be a NSP), a SP could choose its owning NP, almost irrespective of current performance, hence providing the customer with (probably) a performance adequate to meet the

Customer-SP SLA, but not the best service possible, or even a fair allocation of available resource. Equally, a NP may prefer to give concessions to its NSP and preferentially reject other SP requests for relaxations.

Given that a supplier's performance meets the agreed SLA does a buyer have a right to expect fairness? Arguably if a buyer is unhappy with the SLA they should negotiate another SLA. However a problem with such an approach is that the ability to negotiate a strong SLA depends on the relative power of the negotiators. This imbalance means that small customers are often given take it or leave it option. Formation of co-operatives of customers is one option, protective regulation another. Yet another is for the customer to monitor the performance. (The monitoring and selection procedures could in practise be software provided by another party.) The use of feedback statistics, through for example the reputation model outlined previously, of recent NP performance with respect to the SP-NP SLA, other SP-NP SLAs and with ideal performance (no resource constraints) and other NPs gives the end customer the potential of a more zealous performance, rather than the minimum specified in the SLA. Choosing a SP with the best reputation will force the SP to choose the NP with the best performance, so there is a knock on effect. The reputation mechanism is however not adequate to prevent collusion by all SPs or all NPs.

What if all the NPs collude in treating one SP unfairly, but still keep within the terms of the SLAs? For example too many SPs could drive down prices as they each chase the fixed demand, and so a cartel looking for comfortable margins may want to limit entry of new SPs. More specifically a regulatory board may insist on the grounds of openness of the market that NPs sell bandwidth to SPs that are not NSPs. A cartel of NPs could favour the NSPs when resources are scarce providing poorer service to the non- NSP, but not affecting their relative reputation. For blatant denial of service at all times, such collusion is not stable, as if one NP breaks rank and picks the outcast SP then that NP will have a competitive advantage. However, if service is denied to the outcast SP only when resource is scarce and what is left is reserved to give QoS for NSPs then the commercial advantage to the NP to stay in the cartel could exceed that of breaking it. Could such a situation be recognised by calculating a fairness metric or circumvented by some mechanism? Notice that this applies even if we do not work in the any to any market. If a SP is committed to a NP it still would like to be assured of fair treatment.

Different fairness metrics can be considered. When resource is scarce and a SP is requesting from the NP additional resource (beyond for example what may have been purchased in a "block" agreement) then the SP wants to be sure that it is not being unfairly discriminated against. It would like equal access to resource that is available under the same mechanism. This could be a small portion of resource managed by the NP or all of it depending on the business model of the NP. However it is an important part when demand is high. (Notice that we anticipate that normally no resource will be 100% guaranteed rather a probability of allocation will be guaranteed. The simpler model can be accommodated.) It is possible to construct a statistic that can be used to determine whether a SP has been provided a fair allocation. For example if we assume that units bandwidth is allocated independently (which they are not as some connections use more bandwidth than others) then a simple exact likelihood ratio test can be constructed or an approximate Chi squared statistic computed.

More sophisticated statistics could be derived to allow for the fact that connections are of different bandwidths. The key feature of such a statistic is that it can only be computed by the NP or a third party acting as a monitor. A SP cannot compute its own statistic because it does not know the total bandwidth demand for the NP. The statistical approach also depends on having reasonable probabilistic model under the null hypothesis of fairness in order to derive a sensible statistic. So whilst feasible and potentially discriminatory to degrees of unfairness, this approach is complex and requires a third party.

Should a Customer be able to impose the wish not to use a particular SP or NP because, for example, the NP exploits child labour? There is no problem with this even if the SP or NP selections are blind, as long as the required attributes are visible and not used for collusion.

Resource plane

As previously stated, the Resource Plane is the domain of the NP. The Network Provider must manage its' resources in order to accommodate the various SLAs that it has with all the SPs. Depending on the NP's model of the SP the NP may be willing to go to varying lengths to appear "reliable" to that SP. If the NP wishes to appear very reliable to a SP, but does not have the dedicated resources to provide the SP at that moment, the NP has a

variety of options. The first type of operation the NP can undertake in order to free up some resources for the SP are called cell-scale operations. An example of a cell-scale operation might be that at call select, a different type of cell is chosen (e.g, macro- cell rather than a pico-cell), probably providing a lower QoS to the user. Another example of a cell-scale operation is to forcibly handover a connection to a different type of cell (using a different frequency band), also providing a lower QoS. In cases where cell-scale manipulations are not adequate the NP may go one step further and perform radio resource operations. In Shuffle the operations are classified as planning and reactive and the MAS will include a layered model.

Examples of operation

New SP agent enters the marketplace. Who will act on the behalf of a SP when a SP wants to register its agent in each marketplace? The SP cannot do this by itself since there is not a trusted relationship between SP and the marketplaces (it is not feasible). The trusted third party does, since all the instances of the marketplace trust it (have its public key). The registration office of a marketplace is queried about the operation and starts the process validating the origin and integrity of the new agent (e.g. digest of code signed by the owner of the marketplace). If the agent passes these tests, the office assigns an original name to the agent and a certificate is issued by the trusted party. The agent, with its certificate and name, enters the marketplace and stays in the agent layer. The directory of agents, facilities and network services is updated in the registration office and this new is made public throw an information service. This is repeated for every instance. Customer request. We suppose that a customer request arrives at a Service Provider Agent. An information service facility is involved in this operation in order for the SP to know that a user wants a connection. Each SP agent is different from other SPs: the SP owning the request has encoded a specific private architecture so it offers unique behaviour. Let SPAa be the SPA assigned to the user in this example. SPAa has some previous experience dealing with network and service providers. Suppose SPAb, a new Service Provider, has been announced. in the marketplace. SPAa experience says NPa is the best option for the customer, but it does not fits completely the query of the customer. Observing the social indicators provided by a Domestic Information Service, SPAa realises that the new SPAb

is trading successfully with many others SPAs. SPAa has an agreement, notary-signed, with SPAc that allows at this moment to have a similar offer to that of NPa. A parallel trading is started by SPAa with NPa, SPAb and SPAc. Trade ends with SPAc offering a better deal if they sign a loyalty agreement for a period of time. They use the notary facility to make the agreement legal in the marketplace and submit the connection query to the connection centre. The connection centre validates the query (norms are observed and SLAs fulfilled) and submits the query to the network layer where resource agents will establish the connection. In this case the validation is very light as the full validation is done by periodic retrospective monitoring. This prevents the agents to operate directly network connections, avoiding the chance of cheating, misuse of resources or "illegal" operations (such as establishing a connection not associated with a SLA).

5.3.5 Synopsis

A simple trust model for a multi-agent marketplace has been presented, where several mechanisms have implemented several security requirements. We have analysed a business environment for selling of bandwidth resource and services.

The trust model is based on concentric spheres structure, with physical security in the core, a security infrastructure in middle spheres and complex aspects of trust in outer spheres, such as reputation, fairness and reliability. Benefits of the partition of the trust model into shells are adaptability, flexibility and scalability. For example, it is very easy to add new facilities or norms or use the model in a large scale marketplace.

Chapter 6

Conclusions

Mobile Agent technology, as we have seen in chapter 2, has reduced complexity of designing of complex applications with special requirements (intermittently off-lined users, bandwidth limitations, etc.), inherently providing scalability. It has enabled new types of applications, for example sea-of-data applications, where code execution at data location is a requirement. One of the main aims of this dissertation was to analyse the problem of security in Multi Agent Systems, and especially in mobile agent environments. In chapter 2 this technology has been reviewed. We have seen how agents can be used to implement complex systems, but at the same time they introduce new security issues, with very difficult solutions. Through chapter 5 we have shown some specific security solutions we have designed to solve particular security problems on some applications: fault tolerance in electronic voting system based on mobile agents, non-repudiation protocol for mobile agent migration, and security infrastructure in agent based marketplace for the selling of bandwidth. Some of the described mechanisms can be added in a repository for later use within a more general scheme to find security solutions (such as our proposal in chapter 4).

Although mobile agent technology allow the implementation of sea-of-data applications, none of the available platforms is capable of doing this fulfilling some basic requirements of security and easy to program. Even more, none of these platforms is flexible enough to allow easy extensions to implement this. In chapter 3 we have presented the start of the implementation of MARISM-A, a novel Mobile Agent System especially designed to easily develop secure sea-of-data applications. MARISM-A is being designed to observe

both FIPA and MASIF standards. Most novel features in this platform include the flexibility of agent internal architecture (programmers can add their own agent architectures in run time), and the protectability of agent itineraries and collected results. These characteristics lead to define a new programming paradigm based on location. We have outlined major security requirements of sea-of-data applications by using the Byzantine Princes Problem Sequel, a problem we have designed based on the BPP problem [DM01]. With this traditional method for security requirements analysis it is not possible to assure that there are not more requirements or even that the requirements found are relevant for the application. MARISM-A is still at very early stages and has many needs. It is not fault-tolerance yet, for instance, and assumes agencies can not act maliciously. Although having malicious agencies is not the normal situation (we provide for the most common scenario of agencies distrusting each other and cooperating with users), we plan to do some more research in this direction. The Integrated Development Environment of the platform is also in its beginnings, but it is already able to generate agents with complex itineraries. Next future road-map includes continuing the implementation of both MARISM-A internals and API, fitting new research results in the field of security, and a user-friendly IDE to allow rapid application developing.

As an alternative to traditional security requirements analysis methods we have proposed to use an approach based on trust. Trust provides a new prospect for the analysis and solution of the security problem, especially in mobile agent systems. Since trust is not a simple concept, we have characterised the concept in chapter 2, providing different points of view and finding out its basic elements. Next step has been to define a model for trust and provide a methodology to use it to find trust/security requirements in mobile agents applications and point out security solutions. Our trust model has tried to fill up the alarming gap between policies and practise that it is commonly found on similar approaches. This has been achieved by means of structures and a methodology. The model and the methodology have been defined in chapter 4. An application using MARISM-A has been chosen to show how this methodology can be applied to achieve security solutions. Nevertheless, our trust model is just described and further research must be done in this direction. The novel idea here has been to describe a trust plane, parallel to security, in which represent trust requirements as a transformation of security requirements in highly socialised scenarios.

This trust model, however, does not provide security solutions for all requirements. More mechanisms are still needed to implement security solutions and be used to fulfil requirements. We plan to make use of adapted role-based access control mechanisms using SPKI certificates to solve specific requirements of sea-of-data applications and ubiquitous computing. [SA02] points out new directions toward security solutions in ubiquitous computing using a new security policy model known as Resurrecting Duckling.

We have joined *iTrust*, a working group on trust management in dynamic open systems with European funding (IST-2001-34910). In this working group we expect to contribute with our experience in the MARISM-A project and in the computer security area and get valuable feedback to continue the developing of our practical trust model and the implementation of different security solutions on our platform.

As shown along this dissertation, security in complex mobile agent applications, such as those based on the sea-of-data model, does not admit an easy solution. Trust has provided an alternative to traditional methods to find a global solution for security issues. However, it has not solved the entire problem. Trust clearly enables a better model to find security requirements in highly socialised applications, as shown in this dissertation. Nevertheless, further research in the application of a trust model is still required. The seek on this topic will lead our research in the next future. MARISM-A project has resulted very helpful for the cohesion of several research lines. We hope it becomes a useful platform for MAS programmers and researchers in the foreseeable future, bolstering up the developing of new applications and security proposals. Simultaneously, MARISM-A applications will be used as testbeds to our trust model methodology.

Bibliography

[Abe98] M. Abe. Universally Verificable Mix-net with Verification Work Independent of the Number of Mix-Servers. In *Proceedings of Eurocrypt*, LNCS 1403, pages 437–447. Springer-Verlag, 1998.

[AC01] Agentcities, 2001. `http://www.agentcities.org`.

[AR97] A. Abdul-Rahman. *The PGP Trust Model*. EDI-Forum, 1997.

[ARH00] A. Abdul-Rahman and S. Hailes. Supporting Trust in Virtual Communities. In *33rd Hawaii International Conference on System Sciences*, 2000.

[Bae98] J. Baek. A Design of a Protocol for Detecting a Mobile Agent Clone and its Correctness Proof Using Coloured Petri Nets. Technical Report TR-DIC-CSL-1998-002, Kwang-Ju Institute of Science and Technology, 1998.

[BAN90] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*, 8(1), 1990.

[BBK94] T. Beth, M. Borchedring, and B. Klin. A Logic of Authentication. In *European Symposium on Research in Computer Security*, 1994.

[BCH+00] J. Bigham, L.G. Cuthbert, A. Hayzelden, Z. Luo, J. Borrell, and S. Robles. Distributed Control of Connection Admission to a Telecommunications Network: Security Issues. In *4th Conference of Systemics, Cybernetics and Informatics*, Florida, 2000.

[BCS99] P. Bellavista, A. Corradi, and C. Stefanelli. A Secure and Open Mobile Agent Programming Environment. In *Proceedings of the Fourth International Symposium on Autonomous Decentralized Systems (ISADS '99)*, pages 238–245, Tokyo, Japan, 1999. IEEE Computer Society Press.

[BHBR01] J Bigham, A.L.G Hayzelden, J. Borrell, and S. Robles. Distributed control of connection admission. In A.L.G. Hayzelden and R.A. Bourne, editors, *Agent Technology for Communication Infrastructures*, chapter 6. Wiley, 2001.

[BHRS98] J. Baumann, F. Hohl, K. Rothermel, and M. Straßer. Mole - Concepts of a Mobile Agent System. *Special Issue on Distributed World Wide Web Processing: Applications and Techniques of Web Agents*, 1(3):123–137, 1998.

[Bou00] R. A. Bourne. A Quote-Driven Market for Service Providers in Telecommunications Networks. In *Proceedings of 8th International Conference on Telecommunication Systems Modelling and Analysis*, pages 205–209, 2000.

[BRSR99] J. Borrell, S. Robles, J. Serra, and A. Riera. Securing the Itinerary of Mobile Agents through a Non-Repudiation Protocol. In *IEEE International Carnahan Conference on Security Technology*, pages 461–464, 1999.

[BTC+00] J Bigham, L. Tokarchuk, L. Cuthbert, S. Robles, and J. Borrell. Towards a Global Trust Model for a Multi-Agent System to Manage Transactions in Future Mobile Communication Networks. In *3rd UK Workshop of the UK Special Interest Group on Multi-Agent Systems*, 2000.

[BTR+01] J. Bigham, L. Tokarchuk, D.J. Ryan, L.G. Cuthbert, J. Lisalina, M. Dinis, and S. Robles. Agent-based resource management for 3G networks. In *Second International Conference on 3G Mobile Communication Technologies*, pages 236 – 240, London, UK, March 2001. IEEE, IEEE Press.

[CC96] L.F. Cranor and R.K. Cytron. Design and Implementation of a Practical Security-Conscious Electronic Polling System. Technical Report WUCS-96-02, Washington University. St. Louis, 1996.

[CCD02] Combinatorics and Digital Communication Research Group (CCD). An Architecture fo Mobile Agents with Recursive Itinerary and Secure Migration (MARISM-A). `http://www.marism-a.org`, 2002.

[CFT98] A. Caprara, M. Fischetti, and P. Toth. Algorithms for the Set Covering Problem. Technical Report OR-98-3, DEIS-Operations Research Group, 1998.

[Cha81] D. Chaum. Untraceable Electronic Mail, Return Addresses and Digital Pseudonyms. *Comm. ACM*, 24:84–88, 1981.

[Che97] Y. Cheng. A Comprehensive Security Infraestructure for Mobile Agents. Master's thesis, Universitat d'Estocolm/KTH, 1997.

[DH00] J. Domingo and J. Herrera. Enhanced Hash Chains for Efficient Agent Route Protection. *Information Processing Letters*, April 2000. submitted.

[DM01] D. L. Drake and K. L. Morse. Analyzing the Byzantine Princes with Trust Models. In *Proceedings of the Fourth International Conference on Electronic Commerce Research (ICECR)*, volume 2, pages 390–396, Dallas, USA, November 2001.

[DOW92] W. Diffie, P.C. Van Oorschot, and M.J. Wiener. Authentication and Authenticated Key Exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[EFL+99] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI Certificate Theory. Request For Calls 2693, 1999.

[FIP00] Foundation for Intelligent Physical Agents. FIPA Specifications. `http://www.FIPA.org`, 2000.

[FOO92] A. Fujioka, T. Okamoto, and K. Ohta. A Practical Secret Voting Scheme for Large Scale Elections. In *Proceeding of AUSCRYPT'92,* LNCS 718, pages 244–251. Springer-Verlag, 1992.

[FOS02] Emorphia. FIPA-OS. `http://fipa-os.sourceforge.net`, 2002.

[Gam90] D. G. Gambetta. Can We Trust Trust? In D. G. Gambetta, editor, *Trust: Making and Breaking Cooperative Relations*. Basil Blackwell, Oxford, 1990.

[GH99] M. Geier and F. J. Hauck. Scalable Migration for Mobile Agents. In *MOS'99*. INRIA, 1999.

[GH00] M. Geier and F. J. Hauck. Fragmented Objects for the Implementation of Mobile Agents. Technical Report TR-I4-00-05, Friedrich-Alexander University, Erlangen-Nürnberg, Germany, 2000.

[Gon97] L. Gong. *Java Security Architecture for JDK 1.2*. Sun Microsystems, Inc., Mountain View, 1997.

[GV97] C. Ghezzi and G. Vigna. Mobile Code Paradigms and Technologies: A Case Study. In *Proceedings of the First International Workshop on Mobile Agents*, LNCS 1219, pages 39–49, Berlin, Germany, 1997. Springer-Verlag.

[HC02] J. Herrera and J. Castellà. Expanded *Hash* Chains for n Values Compact Commitments. In *Proceedings of the 7th Spanish Meeting about Cryptology and Information Security.*, Oviedo, 2002. To appear.

[HCK95] C. Harrison, D. Chess, and A. Kershenbaum. Mobile Agents: Are they a good idea? Technical report, IBM, IBM T.J. Watson Research Center, 1995.

[Hoh98a] F. Hohl. A Model of Attacks of Malicious Hosts Against Mobile Agents. In *Proceedings of the ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations*, pages 105–120, INRIA, France, 1998.

[Hoh98b] F. Hohl. Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts. In *Mobile Agents and Security*, LNCS 1419, pages 92–113. Springer-Verlag, 1998.

[HR99] F. Hohl and K. Rothermel. A Protocol Preventing Blackbox Tests of Mobile Agents. Article acceptat pel 11. Fachtagung "Kommunikation in Verteilten Systemen" (KiVS'99), 1999.

[IKV00] IKV++. Grasshopper. The Agent Platform. Technical report, IKV++ GmbH, Berlin, 2000. http://www.ikv.de/products/grasshopper.

[ITU93] ITU-T. *Recommentadion X.500 (11/93) – Information technology – Open Systems Interconnection – The directory: Overview of Concepts, Models and Services*, 1993.

[Jøs96] A. Jøsang. The Right Type of Trust for Distributed Systems. In *New Security Paradigms Workshop*, 1996.

[JW98] N. R. Jennings and M. J. Wooldridge, editors. *Agent Technology*. Springer-Verlag, 1998.

[KAG98] G. Karjoth, N. Asokan, and C. Gülcü. Protecting the Computation of Free-Roaming Agents. In *Proceedings of the Second International Workshop on Mobile Agents*, LNCS 1477, pages 194–207. Springer-Verlag, 1998.

[Mar94] S. Marsh. *Formalising Trust as a Computational Concept*. PhD thesis, University of Stirling, 1994.

[Mau96] U. Maurer. Modeling a Public-Key Infrastructure. In *European Symposium on Research in Computer Security*, 1996.

[MB02] J. Mir and J. Borrell. Protecting general flexible itineraries of mobile agents. In *Proceedings of ICISC 2001*, LNCS. Springer Verlag, 2002. To appear.

[MBB+98] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, et al. MASIF: The OMG Mobile Agent System Interoperability Facility. In *Proceedings of the Second International Workshop on Mobile Agents)*, pages 50–67, Stuttgart, Germany, 1998.

[MS00] Y. Mass and O. Shehory. Distributed Trust in Open Multi Agent Systems. In *Proceedings of Autonomous Agents 2000' workshop on Deception, Fraud and Trust in Agent Societies*, pages 81–86, 2000.

[MvOS96] A. Menzes, P. van Ooschor, and Vanstone S. *Handbook of Applied Cryptography*. CRC Press, 1996.

[NL96] G. C. Necula and P. Lee. Proof-Carrying Code. Technical report, School of Computer Science, Carnegie Mellon University, September 1996.

[NRB02a] G. Navarro, S. Robles, and J. Borrell. Adapted Role-based Access Control for MARISM-A using SPKI Certificates. In *2nd. International Workshop on Security of Mobile Multiagent Systems*, Bologna, July 2002. submitted.

[NRB02b] G. Navarro, S. Robles, and J. Borrell. SPKI for Resource Access Control in Mobile Agents Environments. In *Proceedings of the 7th Spanish Meeting about Cryptology and Information Security*, Oviedo, 2002. In Spanish. To appear.

[Pal94] E. Palmer. An Introduction to Citadel - a Secure Crypto Coprocessor for Workstations. In *Proceedings if the IFIP SEC'94*, 1994.

[PC00] S. Poslad and M. Calisti. Towards Improved Trust and Security in FIPA Agent Platforms. In *Proceedings of Autonomous Agents 2000' workshop on Deception, Fraud and Trust in Agent Societies*, pages 87–90, 2000.

[PCGK89] D. A. Patterson, P. Chen, G. Gibson, and R. H. Katz. Introduction to Redundant Arrays of Inexpensive Disks (RAID). In *IEEE COMPCON 89*, San Francisco, 1989.

[RB99] A. Riera and J. Borrell. Practical Aproach to Anonymity in Large Scale Electronic Voting Schemes. In *Network and Distributed System Security, NDSS*, pages 69–82. Internet Society, 1999.

[RB02a] S. Robles and J. Borrell. A Fault-Tolerant Voting Scheme based on Mobile Agents. In *6th Conference of Systemics, Cybernetics and Informatics*, Florida, 2002. To appear.

[RB02b] S. Robles and J. Borrell. Trust in Mobile Agents Environments. In *Proceedings of the 7th Spanish Meeting about Cryptology and Information Security*, Oviedo, 2002. To appear.

[RBB+01] S. Robles, J. Borrell, J. Bigham, L. Tokarchuk, and L. Cuthbert. Design of a Trust Model for a Secure Multi-Agent Marketplace. In *5th International Conference on Autonomous Agents*, pages 77–78, Montreal, May 2001. ACM Press.

[RBR97] A. Riera, J. Borrell, and J. Rifà. Large Scale Elections by Coordinating Electoral Colleges. In *IFIP SEC'97, Information Security in Research and Business*, pages 349–362. Chapman&Hall, 1997.

[RBR98] A. Riera, J. Borrell, and J. Rifà. An uncoercible verifiable electronic voting protocol. In *IFIP SEC'98*, pages 206–215. Austrian Computer Society, 1998.

[RHG99] L. Rosenberg, T. Hammer, and A. Gallo. Continuous Risk Management at NASA. In *Proceedings of the Applied Software Measurement / Software Management Conference*, 1999.

[Rie99] A. Riera. *Design of Implementable Solutions for Large Scale Electronic Voting Schemes*. PhD thesis, Universitat Autònoma de Barcelona, 1999.

[Rif93] J. Rifà. How to avoid the Cheaters succeed in the Key Sharing Scheme. *Desings, Codes and Cryptography*, 3:221–228, 1993.

[RLN+02] S. Robles, I. Luque, G. Navarro, J. Pons, and J. Borrell. Secure Platform for Mobile Agent Execution with Protectable Itinerary, Code and Data. In *Proceedings of the 7th Spanish Meeting about Cryptology and Information Security*, Oviedo, 2002. In Spanish. To appear.

[RMB02] S. Robles, J. Mir, and J. Borrell. MARISM-A: An Architecture for Mobile Agents with Recursive Itinerary and Secure Migration. In *2nd. IW on Security of Mobile Multiagent Systems*, Bologna, July 2002. (submitted).

[Rob99] S. Robles. Applying Mobile Agents Systems into Large Scale Voting System Designing. Master's thesis, Universitat Autònoma de Barcelona, 1999. In Catalan.

[Rot98] V. Roth. Secure Recording of Itineraries through Cooperating Agents. In *MOS'98*, pages 147–154. INRIA, 1998.

[RPBB01a] S. Robles, S. Poslad, J. Borrell, and J. Bigham. A Practical Trust Model for Agent-Oriented Electronic Business Applications. In *4th International Conference on Electronic Commerce Research*, volume 2, pages 397–406, Dallas, USA, November 2001.

[RPBB01b] S. Robles, S. Poslad, J. Borrell, and J. Bigham. Adding Security and Privacy to Agents Acting in a Marketplace: A Trust Model. In *Proceedings of the 35th Annual IEEE International Carnahan Conference on Security Technology*, pages 235–239, London, October 2001. IEEE, IEEE Press.

[SA02] F. Stajano and R. Anderson. The Resurrecting Duckling: Security Issues for Ubiquitous Computing. *Computer, IEEE Computer Society*, Security & Privacy supplement, April 2002.

[SBH96] M. Straßer, J. Baumann, and F. Hohl. Mole - A Java Based Mobile Agent System. In *Workshop Reader ECOOP'96*, Dpunkt, 1996.

[Sch96] B. Scheneier. *Applied Cryptography. Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, New York, 1996.

[SHU01] Shuffle. `http://www.ist-shuffle.org`, 2001. IST European Project.

[SR98] M. Straßer and K. Rothermel. Reliability Concepts for Mobile Agents. *International Journal of Cooperative Information Systems (IJCIS)*, 7(4):355–382, 1998.

[SRM98] M. Straßer, K. Rothermel, and C. Maiöfer. Providing Reliable Agents for Electronic Commerce. In *Proceedings of the International IFIP/GI Working Conference*, LNCS 1402, pages 241–253. Springer-Verlag, 1998.

[ST97] T. Sander and C. Tschudin. Towards Mobile Cryptography. Technical Report 97-049, International Computer Science Institute, Berkeley, 1997. `www.icsi.berkeley.edu/ sander/publications/tr-97-049.ps`.

[ST98] T. Sander and C. Tschudin. Protecting Mobile Agents Against Malicious Hosts. In *Mobile Agents and Security*, LNCS 1419. Springer-Verlag, 1998.

[Vig98] G. Vigna. Cryptographic Traces for Mobile Agents. In *Mobile Agents and Security*, LNCS 1419, pages 137–153. Springer-Verlag, 1998.

[Whi94] J. E. White. Telescript Technology: The Foundation for the Electronic Marketplace. White paper, General Magic Inc., 1994.

[Whi95] J. E. White. Mobile Agents. Technical report, General Magic Inc., Octubre 1995.

[WPW+97] D. Wong, N. Paciorek, T. Walsh, J. DiCelie, M. Young, and B. Peet. Concordia: An Infrastructure fo Collaborating Mobile Agents. In *Proceedings of the First International Workshop on Mobile Agents*, LNCS 1219, pages 86–97, Berlin, Germany, 1997. Springer-Verlag.

[WSB98] U.G. Wilhelm, S. Staamann, and L. Buttyán. Protecting the Itinerary of Mobile Agents. In *MOS'98*, pages 135–145. INRIA, 1998.

[WSB00] U.G. Wilhelm, S.M. Staamann, and L. Buttyán. A Pessimistic Approach to Trust in Mobile Agent Platforms. *IEEE Internet Computing*, pages 40–48, September-October 2000.

[Yee97] B. Yee. A Sanctuary for Mobile Agents. In *DARPA Workshop on Foundations for Secure Mobile Code Workshop*, March 1997. `www.cs.nps.navy.mil/research/languages/statements/bsy.ps`.

[YHK95] W. Yeong, , T. Howes, and S. Kille. *Lightweight Directory Access Protocol*. RFC 1777, 1995.

[YKB93] R. Yahalom, B. Klein, and T. Beth. Trust Relationships in Secure Systems - A Distributed Authentication Perspective. In *Symp. on Research in Security and Privacy*, Oakland, 1993. IEEE.

[ZG96] J. Zhou and D. Gollmann. Observations on Non-Repudiation. In *Asiacrypt'96*, LNCS 1163, pages 133–144. Springer-Verlag, 1996.

Sergi Robles

Bellaterra, July 2002