PROVISIONING OF SECURE MULTI-AGENT SYSTEMS (MAS) BASED ON TRUST CONTRACTS

Diligent Philile Biyela

A dissertation submitted to the faculty of science in fulfillment of the requirements for the degree

MASTERS OF SCIENCE

in

COMPUTER SCIENCE

Department of Computer Science

University of Zululand

2004

DECLARATION

This dissertation represents research work carried out by the author and has not been submitted in any form to another university for a degree. All the sources I have used have been duly acknowledged in the text.

DEDICATION

I dedicate this work to my family which is always behind me especially my mother. Her support and encouragement gets me through the toughest of times. To my lovely daughter who her love inspires me to greater achievements. I also dedicate this work to my husband whom I love dearly who supports me in all the endeavors I take.

ACKNOWLEDGEMENTS

I would like to acknowledge my gratefulness to all the staff members of the Department of Computer Science of the University of Zululand. To my supervisor Prof. M.O. Adigun, I would like to express my sincere gratitude and state the I would not have done this without you, his support and wisdom which I greatly appreciate. To my colleagues, and to all who contributed to this research, their help and guidance is highly appreciated.

To God who gave me the strength and wisdom to see through this study, I am also very appreciative and acknowledge His blessings.

TABLE OF CONTENTS

List of Figures

List of Tables

ABSTRACT

This research work focuses on the development of an Interaction Security Contract (ISC) mechanism that enhances the capability of agents to protect themselves against compromised entities in Multi-Agent system. To realize the framework, two tasks were carried out namely (i) a simulation of an agent based system was created and trust accuracy levels based on the proposed mechanism were evaluated to establish the significance of the proposed security scheme; and (ii) a security-aware agent-oriented shopping mall system was designed and implemented to demonstrate the proposed contract based security model. The results obtained are threefold: (i) This study was able to establish that trust models based on reputation do not obtain accurate trust values for agents to make correct trust decisions in agent systems; (ii) it was established that it was even riskier to form collaborations without any trust model in place as agents collaborated with agents whose security status was not known to them. This clearly shows that agents were at risk of being compromised; and (iii) it was established that if agents publicize their security properties truthfully, trust in the overall agent systems will be greatly improved as agents themselves will only be collaborating with agents who meet their specified level of trust. In conclusion the study advocates the employment of the proposed Interaction Security Contract (ISC) mechanism, since it was demonstrated how knowledgeable trust relationships can be formed and improved in agent based systems.

CHAPTER ONE

1.0 INTRODUCTION

1.1 Overview

Over the years computer systems have evolved from centralized monolithic computing devices supporting static applications into client server environments that allow complex forms of distributed computing. There are three major technologies that have emerged for distributed computing. In historical order, they are the message passing systems, remote procedure call, and distributed object systems [1]. Distributed systems allow clients to access remote functions or objects. However, these functions are predefined on a server; therefore there is no room for client customization. To address this issue, software agents were introduced as software structures capable of making "rational decisions". The idea of mobile agents was introduced to increase system flexibility, scalability and reliability. However, the mobile agent paradigm is still in its infancy hence it has not yet fulfilled all of what it promises. Among the reasons for the paradigm's unmet potential are security concerns and incomplete knowledge of the possible consequences of mobile code use.

Researchers have expressed diverse views on what an agent is, Among many definitions of agents are (i) a persistent software entity dedicated to a specific purpose[2]; (ii) a computer program that simulates a human relationship by doing something that another person could do for you [3]; (iii) the integrated reasoning processes [4]. Furthermore agents have been conceptualized either with focus on negotiation and coordination of information transfers [5], or even with emphasis on the autonomy of the agent [6]. Franklin and Graesser [7] give a comprehensive review of a variety of agent definitions. There seems to be some characteristics that are broadly accepted by many as representative of the key qualities that can be used to assess agency. These characteristics are incorporated in the agent definition by Weiss [8], where agents are viewed as software entities that are capable of flexible autonomous migration from one platform to another in order to fulfill their design goals, and flexibility means three things, namely:

- reactivity: ability of agents to perceive their environment, and respond in a timely fashion to changes that occur in it in order to satisfy their design objectives;

- pro-activeness: ability of agents to exhibit goal-directed behavior by *taking the initiative* in order to satisfy their design objectives;

- social ability: capability to interact with other agents (and possibly humans) in order to satisfy their design objectives.

One of the issues holding the mobile code paradigm back from maturing is security. There are at least four prominent security threats affecting multi-agent systems which are: (i) the security threat of malicious hosts; (ii) the security threat of malicious agents against an executing platform; (iii) the security threat of malicious agents against other agents; and (iv) the security threat of the network infrastructure.

The reasons for the above security threats are obvious when the working mechanism of a mobile agent is considered; this is depicted in Fig. 1.1. A mobile agent firstly resides on a home machine; then it is dispatched to travel autonomously within a specified itinerary. Next, it executes itself in an executing platform, during which it collects host-specific information, and then generates runtime states and variables. The foregoing is an iterative process which continues until the agent returns home with useful information from the last host in its itinerary.

An agent may have incomplete capabilities for accomplishing its goals, in which case it needs other agents to interact with to achieve its design objectives. Hence a Multi-Agent System (MAS) is a grouping that relates mobile agents with mobile agent platforms and the interactions that takes place between them [9]. A MAS exhibits the following characteristics [10]:

- Each agent has incomplete information and capabilities

- There is no global system control
- Data is decentralized, and
- Computation is asynchronous

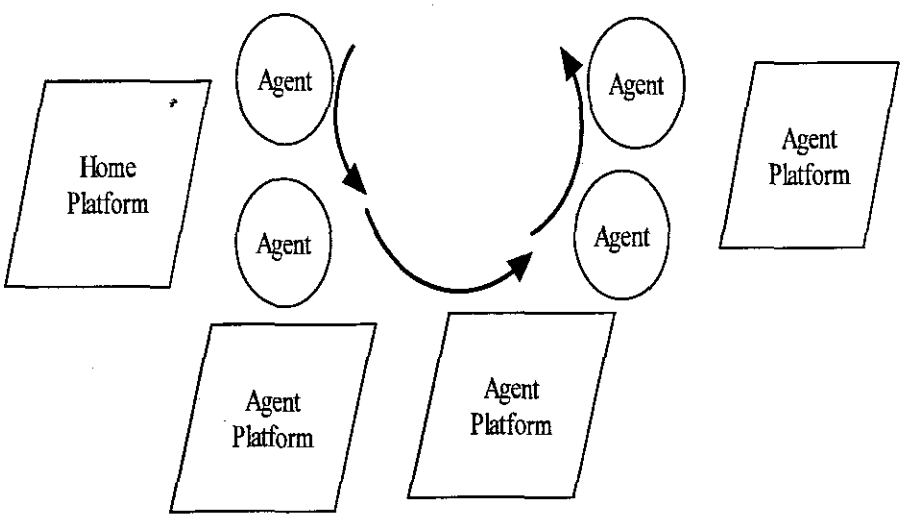


Fig. 1.1 The working mechanism of an agent (taken from 9)

Therefore, multi-agent systems have been described as a social organization of autonomous agents that can flexibly achieve their design goals by interacting with one another. The security of a MAS is threatened by the fact that agents are constantly coming in and going out of the executing platforms. This fact then raises the following security concerns in multi agent systems.

- There are concerns about agents being attacked by malicious platforms [9]. The challenge in agent-based systems is that as agents execute, the executing platform has complete control over the agents that are executing, since they expose their state and data. If a platform is malicious and access control mechanisms for an agent are not strong enough, the platform can easily spy on the state and the data exposed by the agent when executing. The following are the security threats associated with this kind of an attack: masquerade, denial of service, eavesdropping and alteration.

- Some concerns exist about the security of the executing platforms as they host a number of agents from different platforms. The agent platform provides an execution environment for mobile agents. It provides access to the file systems, local executable code, peripherals, system memory, and CPU cycles to mobile agents. However with malicious agents roaming in the network whose intentions may be to spy, corrupt other agents, and/or compromise executing platforms, there is a need to protect platforms from attacks launched by these malicious agents. A lot of work has been done to identify security threats posed by compromised agents to platforms [11, 12, 13, and 14]. Malicious code poses a major threat in the computing world and that is even worse for the mobile code paradigm since these systems are hard to control. Without an effective

safe mechanism to verify, authenticate, authorize, and execute the mobile agent, the host is probably at stake of being attacked. A malicious agent can launch the following attacks on a platform: masquerade, denial of service, unauthorized access.

- There are also concerns about malicious agents launching attacks against other agents executing in the same platform [13]. Numerous agents are executing on platforms, good agents and malicious agents. An agent's security weakness can be exploited by a malicious agent and cause an attack. Threats associated with this kind of attacks are masquerade, unauthorized access, denial of service, and repudiation.

- Finally, there are also concerns of the protection of agents as they are transferred from one platform to the other [13].

This work reports on an approach to address the protection of agents against other agents in agent systems.

1.2 Statement of the Problem

It is fascinating to note the work of Khan et al [15] on how to compose security-aware software. The work raises the important questions of trustworthiness of components and proposes a security property characterization framework for use in component assembly situation. Should components be able to publicize their

security properties at runtime, much more will an agent system benefit from its entities (both agents and platforms) having this capability. In fact a security-aware agent will be more successful in its itinerary than the one that is only protected from outside.

It is therefore required in this research to follow the approach proposed by Khan et al [15] to find a security property characterization for an agent system; formulate a security contract scheme that can be used when agents are collaborating with other agents; and demonstrate that the mechanism works in a multi-agent system situation.

1.3 Research Goal and Objectives

The main goal of this work is to provide a mechanism that makes enforcing a security contract negotiation possible between two interacting agent entities. The main goal is synthesized as an equivalent of some low level objectives which are to:

- Formulate a model to enforce security contract negotiation between two interacting agent entities at run-time.

- Simulate and implement the model developed using the appropriate programming language and software tools.

1.4 Research Methodology

The research methodology includes:

i. Definition of a specification mechanism for an agent interface based on the Compositional security Contract (CsC) concept;

ii. Development of a model of a security-aware agent based system using the Gaia methodology;

iii. Provision of access to security properties through the agent's active interface;

iv. Implementation and simulation of an agent based system to demonstrate the contract-based security model.

1.5 Organization of the Dissertation

The rest of the dissertation is organized as follows. Chapter two presents the literature review conducted, covering from general concepts to the existing state-of-the-art security schemes for agents and platforms in agent based systems. The subject of discussion in chapter three is threefold: first is the analysis of the security characterization framework; secondly is the design of a shopping mall system which serves as a case study for the proposed security framework; and

thirdly this entails the design of a simulator used to evaluate the proposed mechanism. Chapter four reports on the simulation and implementation carried out to prove the workability of the ISC mechanism. Finally, the conclusion and envisaged further work are covered in chapter five.

CHAPTER TWO

2.0 BACKGROUND CONCEPT AND LITERATURE REVIEW

2.1 Introduction

This chapter is divided into two sections in section 2.2, explores the security related concepts. This section review existing security schemes that have been proposed for mobile agents. These mechanisms are categorized according to the security properties they are designed to achieve. Agent based systems methodologies are presented in section 2.3. In section 2.4, trust models are explored. Section 2.5 presents the proposed model. The chapter ends with a summary in section 2.6.

2.2 Security Concepts

Distributed object systems such as CORBA [16], DCOM [17], and RMI [18], enable remote access to objects. However, this remote access introduces new complexity to the security of these systems. Therefore there is a need for the implementation and deployment of proper security policies to protect these systems. These policies should ensure that security in these systems is pervasive, or the security of the entire system cannot be easily compromised. Hence the above mentioned technologies provide the underlying communication infrastructure along with the security subsystem.

Distributed systems security can be roughly divided into two aspects. The first aspect concerns the communication between users or processes possibly residing on different machines. The second aspect concerns authorization which deals with ensuring that a process only gets those access rights to the resources in the system it is entitled to [19]. Kerberos [20, 21] is one distributed security system that is based on shared secret keys to assists clients in setting up a secure channel with a server. CORBA security service is based on the Kerberos, DCE security model to provide comprehensive security services to clients. Another system that bears resemblance to Kerberos is the SESAME (Secure European System for Application in a Multi-vendor Environment) security system [22]. However this system uses public key cryptography combined with shared secret keys. The security architectures of these systems have been proved to work and therefore they form a solid foundation for further development and their approach have been extended to devise security solutions for agent based systems, since agents are an extension of distributed computing.

Agent-based systems tend to be open, dynamic, and unpredictable environments. The openness of these systems is advantageous because it allows a broad range of users to have access to a broad range of services by different competing service providers. However, this openness means that these systems lack global system control and that the information in general is highly decentralized. Needless to say that this poses major security threats as malicious agents may be present in the system trying to spy, or modify, and or corrupt other agents in the system. Existing agent based systems still exhibit security challenges that need to be addressed, if mobile code is to be used to develop mission-critical, real world applications. In Chapter one, an attempt was made to list four threat taxonomies in agent systems. These were identified as: the threat of a malicious agent compromising a platform, the threat of a

malicious agent compromising another agent, the threat of a malicious platform compromising executing agents, and the threat of an insecure communication infrastructure.

There are a number of mechanisms that have been introduced in order to mitigate security threats in agent systems. Several effective solutions have been prescribed for the protection of the platform and the communication infrastructure. However due to the fact that platforms have complete control over agents, most mechanisms that are tailored specifically for agent protection use detection mechanisms as a deterrent. This is influenced by the fact that when an agent is executing it moves beyond the safe boundaries of its home platform, therefore it is exposed to malicious agents and platforms which may compromise its integrity, confidentiality, availability, and anonymity. Otherwise agent systems must be implemented in such a way that they always return to the home platform after executing on a foreign platform, but this does not realize the notion of loose roaming and its advantages.

A lot of work [23,24,25,26,27,28,29,30,31,32,33,34,35] has been done on devising security mechanisms for agent based systems. Different approaches have been prescribed for the different security properties. These security properties can be classified as: integrity, accountability, confidentiality, anonymity, and availability. The next section reviews existing security mechanisms in agent based systems as prescribed to accomplish specific security goals.

2.2.1 Schemes Tailored for Integrity

Partial Result Authentication Code (PRAC) is a scheme that uses cryptographic checksums formed using secret key cryptography [35]. This scheme is meant to provide forward integrity of the partial result obtained by the agent. Forward integrity means that the results obtained in the previous hosts cannot be modified. The forward integrity property ensures that if one of

the platforms is malicious, the results obtained from previous platforms are still valid. However, this scheme has its limitations. The worst case is when a malicious platform retains copies of original secret key of the agent or the key generating functions of an agent. The other weakness of this scheme is that colluding attacks may be possible. Since this scheme is oriented towards integrity and not privacy, the accumulated results can be viewed by any platform, so there is still a need to employ cryptographic primitives to supplement this scheme.

Another scheme Message Authentication Code (MAC) [26] has been proposed to extend the above PRAC mechanism, this scheme goes further due to the fact that apart from encapsulating results at each host, it suggests to associate results with the identities of the previous platforms and the subsequent platforms

Mutual Itinerary Recording [29] is meant to ensure the integrity of agents. The integrity is guaranteed by identifying trusted hosts. This scheme allows for the itineraries of the agent to be recorded and tracked by another cooperating agent and vise versa in a mutual supportive agreement. However this scheme also has its drawbacks, firstly the cost of setting up the authenticated channel is high, and secondly the inability of the peer to determine which of the two platforms is responsible if an agent is killed.

Itinerary Recording with Replication and Voting [30] uses a similar concept with the mutual itinerary recording scheme. This scheme ensures that the agent reaches its destination safely. The main difference with the mutual itinerary recording scheme is that with replication and voting rather than a single copy of an agent performing computations, a number of copies of an agent are used. However, the major drawback with this scheme is the additional resources consumed by replicate agents.

2.2.2 Schemes Tailored for Accountability

To solve the problem of non-repudiation in agent systems a number of schemes have been suggested; three of which are covered in the following below.

Execution tracing [32] is a technique for detecting unauthorized modifications of agents through the faithful recording of the agent's behavior during its execution on each agent platform. The scheme requires each host to construct an execution logging, or tracing, when it executes the mobile agent. This approach also has its limitations, the most obvious being the size and the number of logs to be retained, and the fact that the detection process is triggered occasionally, based on suspicious results or other factors. Another drawback is the lack of accommodating multi-threaded agents and dynamic optimization techniques. While the goal of this scheme is to protect an agent, the scheme does not afford some protection for the agent platform, providing that the platform can also obtain the relevant trace summaries and traces from the various parties involved.

State Appraisal [23] is the mechanism that is aimed at ensuring that an agent has not been compromised due to the alteration of its state. The success of the scheme depends on the extent to which harmful alterations to an agent state can be predicted and mitigation mechanisms in the form of appraisal functions can be prepared before using the agent. The drawback of this technique is that some state alternations cannot be easily foreseen and detected. It has been indicated that it may not always be possible to distinguish normal results from deceptive alternatives.

The Trusted Agent Proxy Server (TAPS) [24] appears to be the most comprehensive scheme. Giansiracusa proposed Trusted Agent Proxy Server (TAPS) architecture to lower the risk

concerns of agent code executing on hostile platforms. This architecture is based on the notion of a trusted proxy server host, which acts as a strong deterrent against malicious behavior from potentially hostile agent platforms.

2.2.3 Schemes Tailored for Confidentiality

Computing with Encrypted Functions [30] is a cryptographic method to protect mobile agents from eavesdropping. This approach falls under the blackbox category. The approach is to encrypt the functions in the mobile agent. The main difference from the traditional encryption technique is that with this scheme the functions after encryption are still usable. This is a promising technique since it applies the cryptographic primitives systematically on an agent. However, this approach currently supports polynomials and rational functions only, if the program implements security-sensitive functions other than polynomials and rational functions, the functions cannot be encrypted.

2.2.4 Schemes Tailored for Anonymity

A few schemes have been derived to provide for anonymity as compared to other security requirements. In [33] Westhoff et al proposed an onion-like data structure that is used to protect the agent's itinerary from wholly known to the remote hosts. Without any protection the itinerary is in its atomic version that is the concatenation of IP addresses of the remote hosts. With the onion-like structure, the itinerary is encrypted layer by layer. The next host information is revealed as long as the remote host has the correct secret key to decrypt. Another anonymity scheme in [34] uses hardware to protect the whole agent, including the itinerary.

2.2.5 Schemes Tailored for Availability

Several schemes are derived to ensure that the platform will allocate resources and allocate with quality of service to the mobile agents. One of those schemes is the CPU Resource Control approach presented in [27]. This scheme assumes that the hosts follow the resource allocation scheme faithfully, without mechanisms to prevent hosts from being malicious. Mobile agents specify their constraints when requesting services from platforms. The platform adjusts with its own constraints to allocate CPU resources to the mobile agent. With both constraints set by the agent and the platform, this scheduling scheme addresses security and the quality of service. However, this scheme does not detect platforms that do not achieve a certain quality of service when an agent is returned.

2.3 Agent-Oriented Software Development Methodologies

Agents are used to understand, model, and develop an important class of distributed systems. If agents are to realize their potential, there is a need to develop software engineering techniques that are specifically tailored for them. Traditional object-oriented software development techniques fail to capture an agent's flexible, autonomous problem solving behavior. Hence methodologies such as the Gaia methodology [36], AUML [37], Tropos [38], and MASE [39] have been introduced to provide abstractions that are specifically tailored for agent based system analysis and design. However this work adopts two of these methodologies and also employs an ACL (Agent Communication Language) message structure from FIPA (Foundation for Intelligent Physical Agents) specifications [40] to model the demonstration system. This is elaborated on overleaf.

2.3.1 The Gaia Methodology

MAS according to Gaia are viewed as being composed of a number of autonomous interacting agents that live in an organized society in which each agent plays one or more specific roles. In the Gaia design process the first step is to map roles into agent types and to create the right number of agent instances of each type. The second step is to determine the services model needed to fulfill a role in one or several agents. Finally, the last step is to create the acquaintance model for the representation of communication between the different agents.

2.3.2 FIPA Modeling Abstractions

AUML (Agent-based Unified Modeling Language) is an initiative by the FIPA Modeling TC (Foundation for Intelligent Physical Agents Modeling Technical Committee). AUML is an extension of UML [41] that is aimed at providing and capturing features that are unique to agent-based systems. AUML defines agent-based modeling abstractions such as capability and service for agent class modeling. These abstractions are employed in this work to model the demonstration system. Furthermore, this work has employed some FIPA specifications [42,43,44] to model the workings of the system.

2.4 Trust Models

Trust has been a subject of research in both sociology and computing. Trust in Multi-agent systems has always been considered from the view point of reputation rather than an experience by the concerned agent itself [45]. Trust has always been central to effective interactions in open multi-agent systems. A number of reputation models have been proposed

for online environments and agent system in general [46], [47], [48], and [49]. The review of these models is presented below.

Sporas [49] is a reputation model that only considers the most recent ratings between two users. Sporas does not store all the ratings, but rather updates the global reputation value of an agent according to its most recent rating. It also introduces a reliability measure based on the standard deviations of the rating values

Histos [49] was designed as a response to the lack of personalization that Sporas reputation values have. On the contrary to Sporas, the reputation value is a subjective property assigned particularly by each individual. The treatment of direct interaction in this model is limited to the use of the most recent experience with the agent that is being evaluated.

Abdul-Rahman and Hailes [48] defined a trust model that uses four degrees of belief to typify agent trustworthiness: *vt* (very trustworthy), *t* (trustworthy), *u*(untrustworthy), and *vu*(very untrustworthy). For each partner and context, the agent maintains a tuple with the number of past experiences in each category. From the point of view of direct interaction, the trust of a partner in a given context is equal to the degree that corresponds to the maximum value in the tuple.

Yu and Singh [46] defined a model where the information stored by an agent about direct interactions is a set of values that reflect the quality of those interactions. Only the most recent experiences with each concrete partner are considered for the calculations. By using the historic information together with the *Dempster-Shafer* theory of evidence, an agent can calculate the probability that its partner gives a service ascribed to each one of these groups.

Agents have been providing recommendations to other agents to weed out bad agents within a community of transacting agents. E-commerce sites such as e-Bay [50], Amazon.com [51], and Bizrate.com [52] have been known to employ reputation based trust models to assess the trustworthiness of buyers and sellers interacting in the system. The problem with building trust blindly based on the recommendations of others is due to the subjective nature of trust [53]. Thus to assume that all agents cognitively process trust in the same way and then to go and define a universal fixed trust algorithm is not a reasonable approach. What is required is the flexibility to allow agents to participate in the trust decision making process. For example when an agent A informs agent B that agent C is untrustworthy, agent B simply takes agent A word for it, without trying to establish whether agent A itself can be trusted about the statement. Therefore, agent B cannot truly say it has the correct trust information about agent C. Thus there is a problem of processing second order beliefs: beliefs about others and beliefs about us. Therefore, a better approach to solving this trust complexity is to enable agents to find out the trustworthiness of other agents themselves.

The Compositional Security Contract (CsC) framework [15] enables trust decisions to be made at runtime, based on the results obtained from the security information negotiated by components. A component will therefore not trust another component unless there is an explicit expression established from the security tests specifying that components can trust each other during a composition. The CsC adopts the existence of a logic rule that requires a positive trust expression to be satisfied before a component can be considered to be trustworthy enough for composition. The scheme defines trust as a binary value, such that an agent either has complete trust in other agent or no trust at all.

2.5 The Proposed Model

This research work proposes a security mechanism called the Interaction Security Contract (ISC). This mechanism is a derivative of the Compositional security Contract (CsC) proposed by Khan and Han in [15]. The ISC is a trust guaranteeing scheme that empowers agents to make crucial trust decisions based on the trust values obtained by the agent itself, rather than through third party intervention. This scheme advocates the publication of the agent's trust attributes through its interface so that agents who are anticipating collaboration can be able to reason about each others trustworthiness. These trust attributes of an agent are identified as the agent identity, origin, and the functionality-specific security properties. The security properties are specified as the required and the ensured properties, where the required properties refer to preconditions that other agents interested in a collaboration should fulfill. The ensured properties refer to the post conditions that guarantee the security service once the precondition is met. Each agent exposes its required and its ensured security properties, this allows agents to reason about each others security properties before the actual collaboration can take place. This study argues that trust based on reputation obtained from third party entities does not reveal accurate trust values. As a result agents should be able to assess the security status of other agents by themselves. This is beneficial since the result obtained from these assessments bears more credibility than the one obtained from third party entities. Furthermore, the proposed approach is cost-effective as it eliminates costs incurred through third party intervention.

The ISC represents trust as a binary value computed as either conformity or non-conformity of an agent's required security properties to another agent's ensured security properties. Therefore agents are able to reason about other agent's trustworthiness, without the

intervention of third party entities. It should be noted that in this study it is assumed that agents publicize their security properties truthfully.

2.6 Summary

In this chapter a number of schemes have been reviewed that are aimed at protecting agents and platforms from malicious entities in the system. It should be noted that our approach deviates from the approaches presented above, in a sense that it facilitates the enforcement of a runtime contractual agreement between two entities anticipating a collaboration. Furthermore, this work does not employ detection as a deterrent, since it is believed that by employing the detection approach, damage will have already been done on an agent. Therefore, as a way of preserving the system's resources, the negotiation approach is the ideal approach to achieve this goal. Moreover, agent based systems methodologies are explored. Finally trust frameworks for agent based systems are reviewed.

Next, the security contract framework is expatiated. This defines how to protect agents before they are involved in collaboration with other agents.

CHAPTER THREE

3.0 SECURITY FRAMEWORK ANALYSIS AND DESIGN

3.1 Introduction

This chapter is divided into four sections. Section 3.2 reports the security characterization framework employed in this work in detail. The next section (section 3.3) covers the design of a security-aware agent-oriented system. The simulation model used for evaluating the proposed mechanism is presented in section 3.4. The chapter ends with a summary in section 3.5.

3.2 The Security Characterization Framework

3.2.1 The Security Model

The proposed security model emanates from the model proposed by Khan and Han in [15]. The model requires the publication of the security functions of agents and hosts as part of their interface. The attributes that affect a security trust relationship are not properly published in agent systems. If these attributes are not known to the software developer, the agent's cannot be trusted completely. These trust attributes are identified as *identity*, *origin*, and *security properties* that agents offer and require from other agents and platforms. Therefore the proposed security model in agent-based systems should provide the security properties of

confidentiality, integrity, authentication, non-repudiations and accountability between any two interacting entities. The proposed security model is discussed in the next subsections.

3.2.1.1 Framework for an Active Interface

The interface of an agent defines the agent, and serves as the basis for the agents understanding, use and implementation. The agent's interface should be the only definitive source for understanding the agent, and therefore the description of the agent's interface should be as definitive and as comprehensive as possible including its security properties. The security characterization framework is based on the notion of an active interface proposed by Khan et al [54, 55]. Hence the security properties are incorporated as an integral part of the agent's interface. An active interface consists of component identity, a static interface signature, and a static security knowledge base. In this scheme an agent has an *identity*, *interface signature* and *security properties*. The structure of an active interface is depicted in Fig. 3.1.

It is assumed that if two agents are interacting, one makes use of some service discovery protocol such as SLP, JINI, and UDDI [56], to discover the other which publishes its services as defined by its interface. If identity and security properties cannot be verified, no interaction is allowed to occur since this implies trust cannot be established between the two. The security model consists of the following modules: Agent identity, interface signature, and the security knowledge base. The description of these modules is as follows:

(i) *Agent Identity*

The identity of the agent is a crucial element as it contains information about the origin of the agent. The identity segment consists of the agent unique ID, the home platform ID, and

a certificate that approves whether an agent is trusted or not. This certificate is issued by a published certification authority which confirms that the home platform of the agent can be trusted. In this scheme it was assumed that public key cryptography was used to identify agents and hosts, hence every agent has a private key only known by it and a public key available for any entity that wants to interact with the owner of the public key. The following is the structure of the agent identity template:

identity(aid, home_platform_id, certificate)

(ii) *Interface signature*

This segment of an interface hosts static operations and attributes. These properties cannot be altered by entities in the system. These are used to make a structural match before two agents can start to collaborate.

(iii) *Security Knowledge Base*

The security knowledge base hosts and makes available the security properties of agents. These security-related characteristics of agents and platforms are categorized as the *required* and the *ensured* security properties. A required property is a precondition that other agents or platforms must satisfy in order to get ensured security services. An ensured security property is a post-condition the agent or platform is responsible for in order to maintain the committed security assurances during the interaction.

These properties are characterized by three basic elements: *operations*, *security attributes*, and *data*. Operations are security-related operations such as: encrypting, hashing through functions such as Secure Hash Functions (SHF), and verifying. These operations are performed by agents and

hosts to enforce security properties. Security attributes are used to perform those operations, and include items such as passwords, keys, and so on. The data are used or are manipulated by the operations, for example, a file or a variable that holds a value such as an account balance.

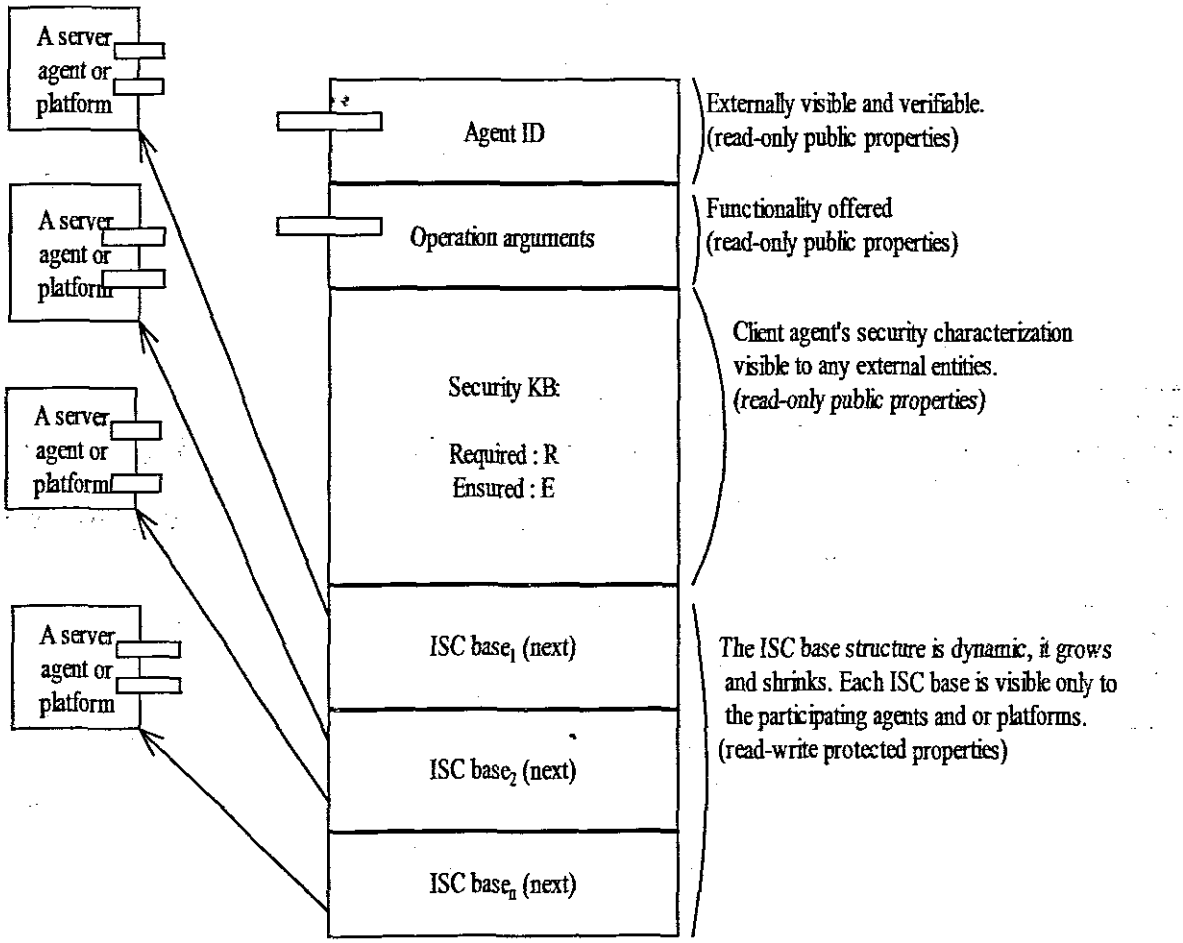


Fig. 3.1 The structure of an active interface adapted from [55]

Based on Khan et al in [15], the security properties of an agent or a platform were characterized with a predicate-like structure such as:

$f(O_i, K_j, D_k)$

Where:

f is a name of the security function formed with three associated arguments.

O is a security related operation performed by an agent or platform in the interaction contract, subscript i is the identity of the agent.

K is a set of security attributes used by the agent and the subscript j contains additional information about **K** such as key type, owner of the key and so on.

D is an arbitrary set of data that are affected by the operation **O**. The subscript k contains additional information attached with **D** such as digital signature use or not, and so on.

3.2.2 The Interaction Security Contract

The ISC is based on the degree of conformity between the required security properties of an agent, and the ensured security properties of another agent. The results obtained from the compatibility tests, is the new security property called the ISC. This contract defines rules for forming a trust relationship based on the conformance of the security properties between two agents intending to collaborate. The security contract is formed when an agent meets the security requirements of a particular agent or host. An agent and or platform will perform runtime checks to verify whether the required security properties of one agent conform to the ensured security properties of another agent or platform. An ISC mechanism should enable an agent to develop a trust relationship with another agent, and or platform based on its own trust evaluation results. Consider an ISC between two agents X & Y, denoted by $I_{X,Y}$, and this relationship can be modeled as follows:

$$I_{X,Y} = (E_Y \Rightarrow R_X) \Lambda (E_X \Rightarrow R_Y)$$

Where

- ***I*** is an interaction contract between two agents, subscripted with the identities of the agents taking part in the contract negotiation.

- ***E*** and ***R*** are the ensured and the required security properties of the participating agents respectively.

- **a⇒b** depicts the implication (a implies b). The evaluation of each pair will result in a Boolean true or false value. In an ISC, interaction only occurs if the result is true.

- The operator Λ denotes a Boolean "and".

Therefore the above expression means that if client agent ***X*** has the required security properties R_X, server agent ***Y*** will ensure the security properties E_Y after interaction, and if server agent ***Y*** has the required properties R_Y, client ***X*** will ensure the security properties E_X. The examples of ***E*** and ***R*** might be:

$R_X = f1(verify, password_Y, file10)$

$E_Y = f2(encrypt, key, file10_X.digisign)$

Client agent X must use and verify the password of server Y to access file10, and will ensure that it encrypts *file10* and digitally signs it using its key.

The algorithm for the agent interface incorporated with security properties is as follows:

<begin AGENT> {AID, Home_Platform_ID, Certificate}

<begin INTERFACE SIGNATURE>

<operation> {<argument$_1$,

, ...

argument$_n$> }

<end INTERFACE SIGNATURE>

<begin SECURITY>

<begin REQUIRED>

<security_function$_1$>{

<security_argument$_1$,

, ...

<security_argumentn>}

, ...

<security_function$_n$>{

<security_argument$_1$,

, ...

<security_argument$_n$>}

<end REQUIRED>

<*begin* ENSURED>

<security_function$_1$>{

<security_argument$_1$,

, ...

<security_argument$_n$>}

, ...

<security_function$_n$>{

<security_argument$_1$,

, ...

<security_argument$_n$>}

<end ENSURED>

<end SECURITY>

The above agent interface is extended with an executable part which is as follows:

<begin ISC>

$\mathbf{R_{AID}}$ **=get(<function>, <REQUIRED>,** $\mathbf{<A_{ID}>}$**);**

$\mathbf{E_{AID}}$ **=get(<function>, <ENSURED>,** $\mathbf{<A_{ID}>}$**);**

$\mathbf{SQ_{FROM}}$ **= conform (**$\mathbf{E_{AID}}$**, <REQUIRED>);**

$\mathbf{SQ_{TO}}$ **= conform (**$\mathbf{R_{AID}}$**, <ENSURED>);**

ISC = conform ($\mathbf{SQ_{TO}}$**,** $\mathbf{SQ_{FROM}}$**);**

Display = out (<ISC>);

<end ISC> <end AGENT>

The purpose of the binary executable part is to compute and generate the ISC. The **get** function reads the security properties from the interface of a server agent (or host platform) and stores it in R_{AID}, the *AID* subscript is the identity of the server agent. In a similar way, the ensured properties are read and stored in E_{AID} using **get**. The variable $\mathbf{SQ_{FROM}}$ stores the security conformity result between the required property of the client agent and the ensured property of the server agent (E_{AID}). $\mathbf{SQ_{TO}}$ stores the security conformity result between the required property of server agent ($\mathbf{R_{AID}}$) and the ensured property of the client agent. The **conform** operation generates the conformity results. A true value indicates a security conformance, and a false value indicates non-conformance. Interaction only occurs if a true value results.

3.2.3 Access Control Enforcement

Access to agent's resources was enforced by using the notion of roles. Roles are not defined for the human users in this context but for agents interacting in the system. Therefore, role-oriented interaction protocols were defined for a given agent. A scenario provides the contexts of use for agents. An agent may be used in different scenarios and therefore has different role partitions in those scenarios. The roles that agents assume in the system were identified, and the security requirements of each role relative to the context were specified. Hence the adoption of the Gaia methodology [36] which uses the notion of a human organization where a software system is conceived as the computational instantiation of a (possibly open) group of interacting and autonomous individuals (agents). Each agent is seen as playing one or more specific roles: it has a well-defined set of responsibilities or sub-goals in the context of the overall system and is responsible for pursuing these autonomously. Hence agents are assigned to roles, and then roles are assigned to permissions.

3.3 The Design of a Security-Aware Multi-Agent System

Agent-oriented software engineering is a promising software paradigm. New methodologies have been introduced to accommodate new abstractions and design/development issues that were not prevalent in the traditional software development approaches. The Gaia methodology [36] adopts the organizational metaphor and lays emphasis on the study and the identification of the organizational structure. In the model, the security requirements from the analysis stage have been incorporated. This ensures that all the functional and the security specifications integrate seamlessly in the system design. The approach favors the definition of security properties that are custom-made for a specific functionality of the system, and not just

the overall security properties are defined. As agent systems are immersed in an open environment, the ISC equips agents with a mechanism that detects trust levels among agents anticipating an interaction

As mentioned earlier, the methodologies for MAS's introduce some new modeling abstractions that are specifically tailored for the MAS environments. The abstractions that are exploited in the analysis and the design phase of the shopping mall system are:

i. The environment in which the MAS is immersed;

ii. The roles to be played by different agents in the organization;

iii. The interactions between these roles;

iv. The organizational rules which capture the responsibilities of the organization as a whole and

v. The organizational structure.

3.3.1 The Shopping Mall System: A Case Study

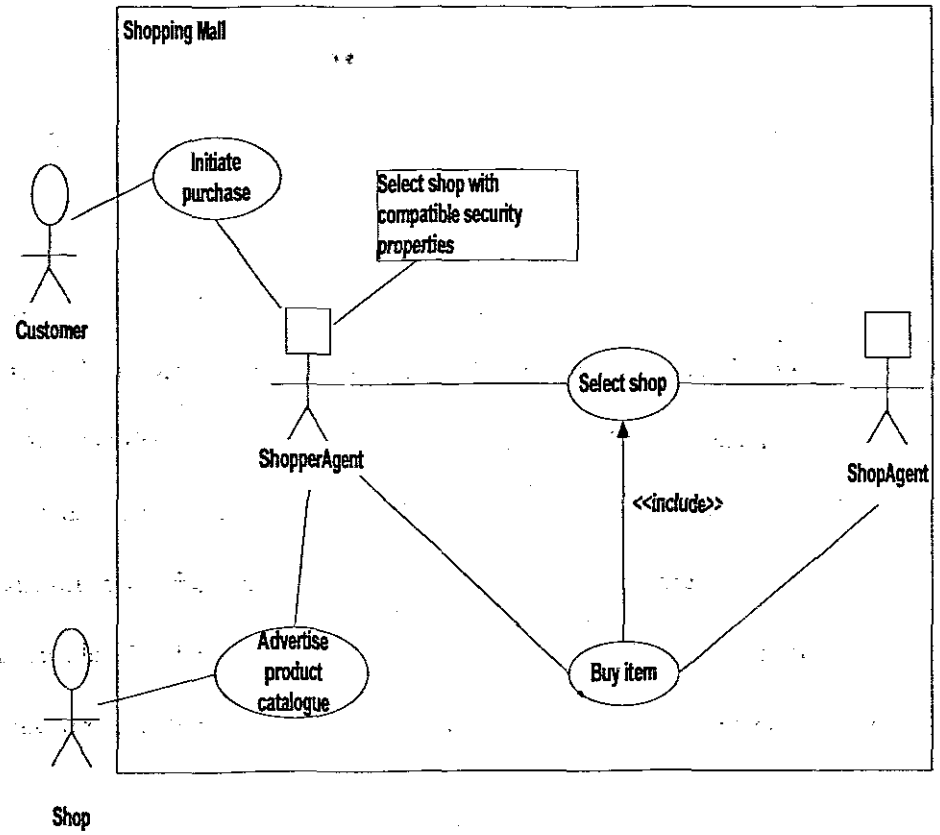
To realize the proposed framework, a security-aware agent-oriented shopping mall system was designed and implemented. Below is the analysis and design of this system, it should be noted that in this study security properties are considered at the inception of the systems design to ensure that all possible system vulnerabilities are taken into consideration. Therefore, the system design includes the systems security requirements. The first step towards the system design is the system analysis. The analysis in this work is divided into two elements i.e. the requirements model and the detailed requirements analysis. UML [41] was employed to model

the system's requirements. The requirements model consists of the use case and the sequence models. A transition is made from the requirements model to the detailed requirements analysis. For the detailed requirements analysis, the Gaia methodology was employed. Furthermore, is the system design, which is made up of the architectural design and the detailed design. From the analysis and specification of the abstractions listed in section 3.3 the system was ready to be implemented.

3.3.2 The Requirements Model

The first step towards a requirements model is the creation of a use case diagram. Use cases provide an abstract view of the system by identifying the main actors using it, and the main functions that the system provides to them. In agent based systems, use cases are extended by the special kind of actor representing the agents within the system. In this system, there is a ShopperAgent, an agent that looks for the clothing items to purchase on behalf of the customer. Then there are a number of ShopAgents that stock items that are needed by the ShopperAgent. ShopAgents query the shops databases for items and prices and present that information to the ShopperAgent. Although a ShopperAgent is looking for the best offer, however, the emphasis here is on the ShopAgent that can provide the security levels required by the ShopperAgent. Therefore even if a ShopAgent offers the lowest price, but a trust contract must first be established with the ShopperAgent otherwise the ShopperAgent cannot continue with the proposal. All ShopAgents are configured with the ISC mechanism for security evaluations, and defined different required and ensured properties for each of them. The ShopperAgent is also configured with the ISC mechanism and has its own security properties. Therefore, a ShopperAgent must be able to query other ShopAgents about their security properties and select the one whose properties match its own. Fig. 3.2 illustrates the

described interactions in the system. The use case diagram is elaborated upon using an overview of the workings of the system and an agent-specific feature set.



Legend:

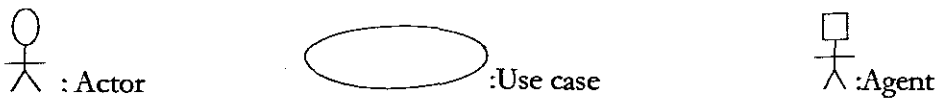


Fig. 3.2 A use case diagram of a shopping mall system

An overview of the shopping mall system

To clearly identify the requirements of agents in this system, a typical shopping situation was examined. The customer needs to purchase a certain item from a Shop. The ShopperAgent then goes to the mall where there is a number of ShopAgents that sell the kind of items that the ShopperAgent needs. The ShopperAgent however has a specific price that it is prepared to pay for that item. More importantly though is that it also has specific security requirements that have to be met before it can commit itself to an agreement. Therefore when looking for the desired items, checking the security requirements is the first thing that the ShopperAgent does. Otherwise if the ShopperAgent does not get the security properties it requires it terminates and looks for the items elsewhere. Following are requirements that the system must satisfy.

Agent-specific features

i. Different modes for request/response: the user does not need to be connected while a request completes. A user specifies a service request with all the relevant service descriptions and its security requirements within the ShopperAgent to query the service provider's agents (ShopAgents) about the items, price, their security properties etc. On the other hand Service Providers (shops) send response messages to the ShopperAgent and wait for a proposal based on the choice of the ShopperAgent.

ii. Comparison of offerings: the agents are capable of comparing the security properties of other agents as their first step, in order to determine whether they conform to the same security policy or not. Furthermore, the system evaluates and provides the user

with different service dimensions such as cost or other user's experience, to enable the ShopperAgent to make an informed choice and

iii. Learning capabilities: firstly, agents in the system are able to keep record of security tests that have been performed and contractual agreements formed. This ensures that agents develop trust relationships with agents whose security has been evaluated. *It should be noted that in this work it is assumed that agents publicize their trust* security properties truthfully and these properties do not change. Secondly, the system becomes more efficient toward the user's needs and habits with continued experience so that when new merchandise or discounts that meets the user profile are advertised on the system, the system is able to generate an event to notify the user about those items.

From the use cases identified in Fig. 3.2, the sequence of messages involved in each use case is formulated. Fig. 3.3 shows the sequence diagram for the Initiate purchase use case diagram, while Fig. 3.4, Fig. 3.5, and Fig. 3.6 shows sequence diagrams for the Select shop use case, Buy item use case, and the Advertise product catalogue use cases respectively. The Initiate purchase use case is invoked when a customer instantiates the ShopperAgent. The Select Shop use case invokes the concurrent sending of request messages to all the service provider agents. The Buy Item use case is fulfilled when a ShopperAgent receives its confirmation for a purchase after it has bought item(s) from the shop. The Advertise Product catalogue use case is invoked when a shop generates notification events to alert the ShopperAgent of special bargains or new items in its database that might be of interest to the customer.

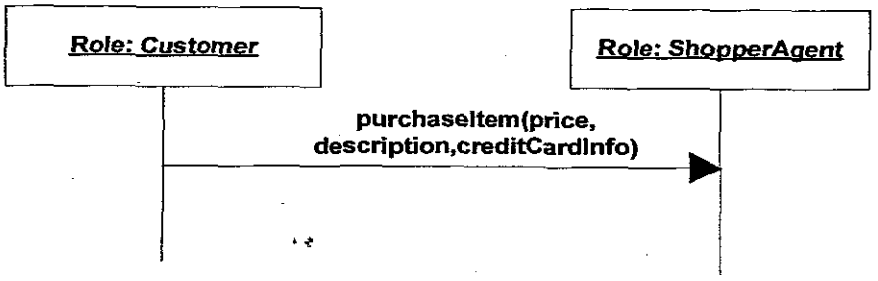


Fig. 3.3 The sequence diagram for the Initiate Purchase use case

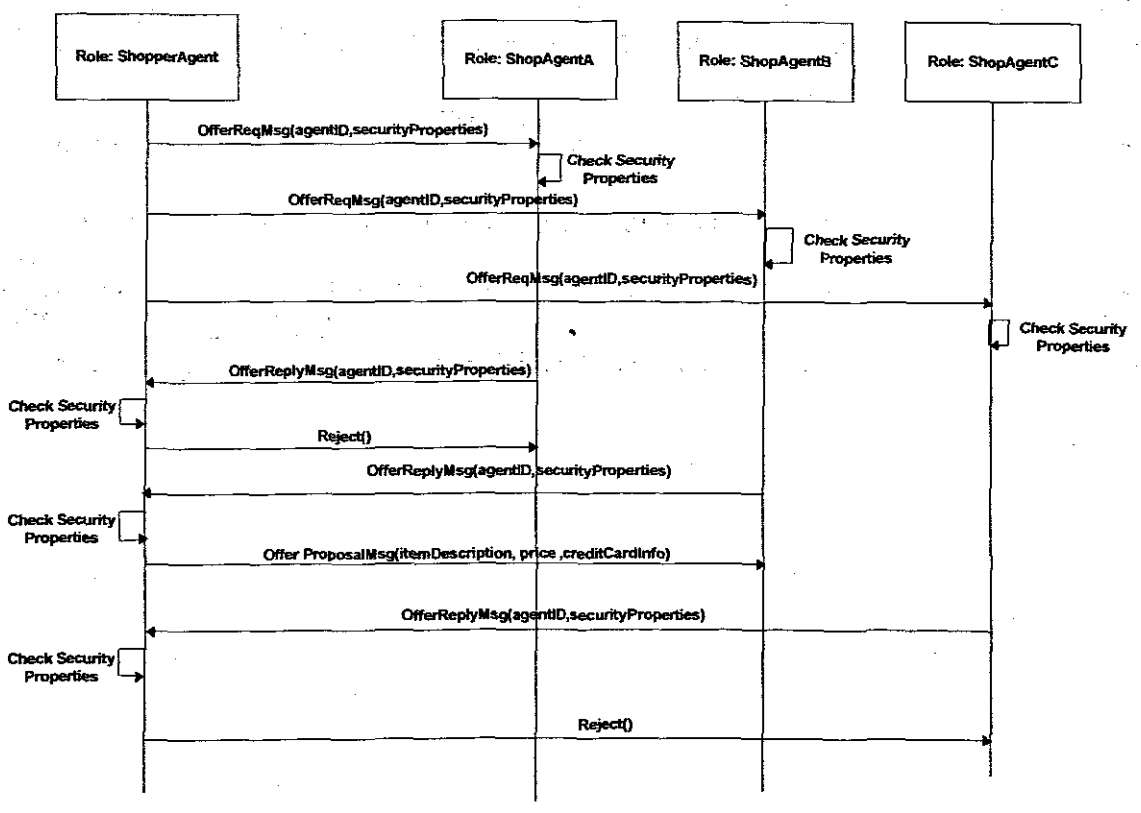


Fig. 3.4 The sequence diagram for the Select Shop use case

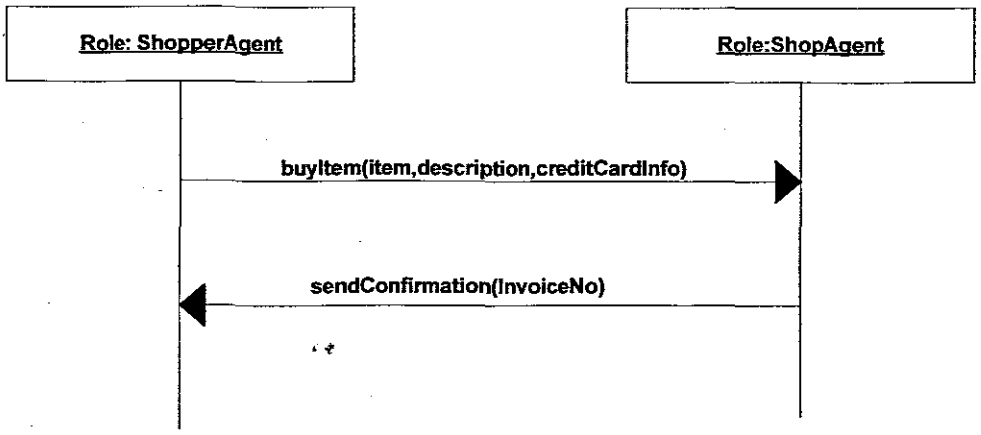


Fig. 3.5 The sequence diagram for the Buy Item use case

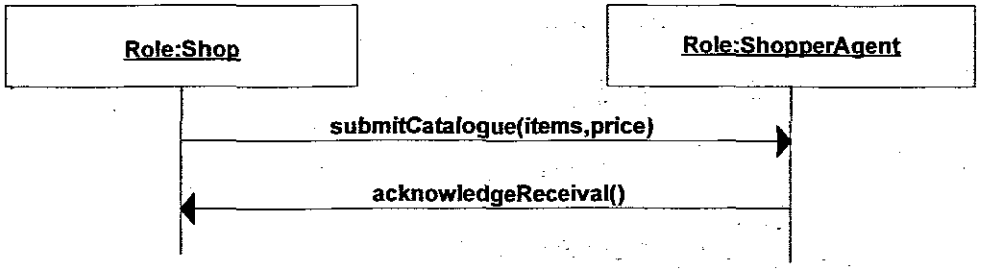


Fig. 3.6 A sequence diagram for the Advertise Product Catalogue use case

3.3.3 Detailed Requirements Analysis

This second element of the system analysis is aimed at analyzing and refining the requirements obtained in the first phase. In this phase, the Gaia modeling abstractions are adopted to organize the collected requirements for the systems into an environmental model, role and interaction models, and a set of organizational rules, for each of the sub organizations composing the system. The Gaia modeling abstractions allows the incorporation of the agent specific characteristics that cannot be modeled in UML.

3.3.3.1 The Organizations

The shopping mall system in this work is comprised of two sub-organizations that are interacting i.e. shoppers (clients) and shops (service providers). Each sub-organization has a specific goal to achieve in the system. The goals of the first sub-organization are to purchase items required by the customer. This sub-organization is responsible for capturing the description of items from the customer, look for shops who sell those items, select shop(s) that meet the client criteria, buy the selected items, and return the result to the customer. The goal of the second sub-organization is to serve the ShopperAgents in the system. This sub-organization is responsible for capturing the description of items specified by the ShopperAgent, reply to the purchase requests, and process the purchase. In all of the above cases there is a clear goal to be pursued by each sub-organization. The most important thing though is how they interact with each other to accomplish their goals in the environment. Therefore the roles that they assume in the system are identified, in doing that the agent's domain of activity is determined; furthermore the inter-domain security policies are specified, to govern the behavior of agents within the same domain. Therefore the first step is to identify the roles in the environment. The roles are elaborated on below.

(i) The ShopperAgent- acts on behalf of the user and is authorized to do so up to the level allowed by the user. The agent must be capable of remembering and adhering to the user's instructions and learning the user's preferences. It must be noted that there is a role that is involved in the system, which is the Directory Manager (DM). This role is inspired by the Directory Facilitator (DF) role in [49]. The DF is a FIPA (Foundation for Intelligent Physical Agents) defined role to provide agents with directory services. The DM role concerns the operational level of the system, and not the application itself, that is why a Gaia representation

for it is not supplied; however it is used later in the design. Furthermore, interactions with this role are presented as protocols, as they are defined in the Gaia methodology, but as activities. Therefore the activities **RegisterDM** and **QueryDM** are DM services provided directly by the executing platform, provided not as a result of interactions between agents, but as methods invocations. Therefore, our ShopperAgent is enabled to register to the DM, deregister from the DM, query the DM, send request messages on the platform, receive messages from service providers affiliated on that platform, and query security properties offered by the ShopAgents whose replies have been received.

(ii) The ShopAgent-this agent provides a service to the ShopperAgent in the sense that it sells the items that are required by the ShopperAgent. Subsequently this agent is responsible for maintaining data access, interpretation, and delivery to the ShopperAgents. The ShopAgent is enabled to register to the DM, deregister from the DM, receive request messages from various ShopperAgents, query the ShopperAgents security properties, and reply the ShopperAgents request messages.

The organizational role models describe all the roles that constitute the computational organization. This is done in terms of their functionalities, activities, responsibilities, interaction protocols and patterns. Based on these constructs, the following schema for the role models was obtained. In this schema all the specifications needed for the identified role in the system to perform its functions are defined. The schema for the role ShopperAgent is depicted in Fig. 3.7, and the schema for the role ShopAgent is depicted in Fig. 3.8.

From the role models obtained, the interaction models were defined. The organizational interactions model describes the protocols that govern the interactions between the roles. Furthermore, these models describe the characteristics and dynamics of each protocol, e.g. when, how, and by whom a protocol has to be executed. Next is the presentation of the protocol models in the system, where a protocol is viewed as an institutionalized pattern of interaction. Hence, in Fig. 3.9 and Fig. 3.10 are interaction models for the RequestShops protocol and the RespondShops protocols respectively.

3.3.4 Architectural Design

3.3.4.1 The Agent Model

In this phase the agent model which was adopted in this study is identified. The agent model creates agent types by aggregating roles. Each emerging agent type can be represented as a role that combines all the aggregated roles attributes (activities, protocols, permissions and responsibilities). The system under discussion adopts a peer-to-peer agent model, which comprises of two agents types: the ShopperAgent and the ShopAgent. However there is also a DM role included since agent's search each other over the DM as illustrated in Fig. 3.11. It is apparent from the illustration that there is only one ShopperAgent in the system, however the implementation is assumed to use a multi-user, server-based design.

Role: ShopperAgent
Description: This role acts on behalf of a profiled user. Whenever a user needs to purchase an item, it searches for the ShopAgent that best fit the needs of the user and recommends the appropriate one to the user. It also receives information on new merchandise and special discounts, and presents that information to the user. Furthermore it is able to check the security properties of the ShopAgents to determine if the proposed negotiation can continue or should be terminated, and updates its ISC base.
Protocols and Activities: CheckProperties, RegisterDM, QueryDM,InitUserProfile, UserRequest, InferUserNeeds, RequestShops, RespondShops, UpdateISC.
Permissions: create, read, update user profile data structure, read acquintance data structure
Responsibilities: **Liveness:** ShopperAgent=InitUserProfile. (ServeUser)$^{\omega}$ ServeUser=UserRequest.RequestShops.RespondShops.InferUserNeeds.PresentShops. **Safety:** The security properties are compatible

Fig. 3.7 The schema for the role ShopperAgent

Role: ShopAgent

Description: It wraps databases for different shops, and provides a shopping facility for ShopperAgents. It registers to the DM, and queries the DM for other ShopAgents that have joined the system, so that it is able to have the latest information on new merchandise, special offers etc. It also gets acquainted with specific agents.

Protocols and Activities:
CheckProperties,RegisterDM,QueryDM,RequestShops,RespondShops,UpdateISC.

Permissions: read DM

Responsibilities:

Liveness:

ShopAgent = RegisterDM. (FindShops)$^{\omega}$
FindShops = RequestShops.QueryDM.RespondShops

Safety:

A successful connection with the DM and shops databases is established.
Security properties are compatible.

Fig.3.8 The schema for the role ShopAgent

Protocol Name: RequestShops		
Initiator: ShopperAgent	Patner: ShopAgent	Input: Service description and security requirements description.
Description: The ShopperAgent requests shops that meet the user requirements from the platform. A set of shops is presented for the ShopperAgent to chose from. The choice is then made based on the security properties of the ShopAgents.		Output: Response on shops that meet all the user's requirements.

Fig 3.9 The RequestShops protocol definition

Protocol Name: RespondShops		
Initiator: ShopAgent	Patner: ShopperAgent	Input: Service descriptions and security properties
Description: The ShopAgent presents the shops that meet the descriptions provided by the ShopperAgent.		Output: -

Fig.3.10 The RespondShops protocol definition

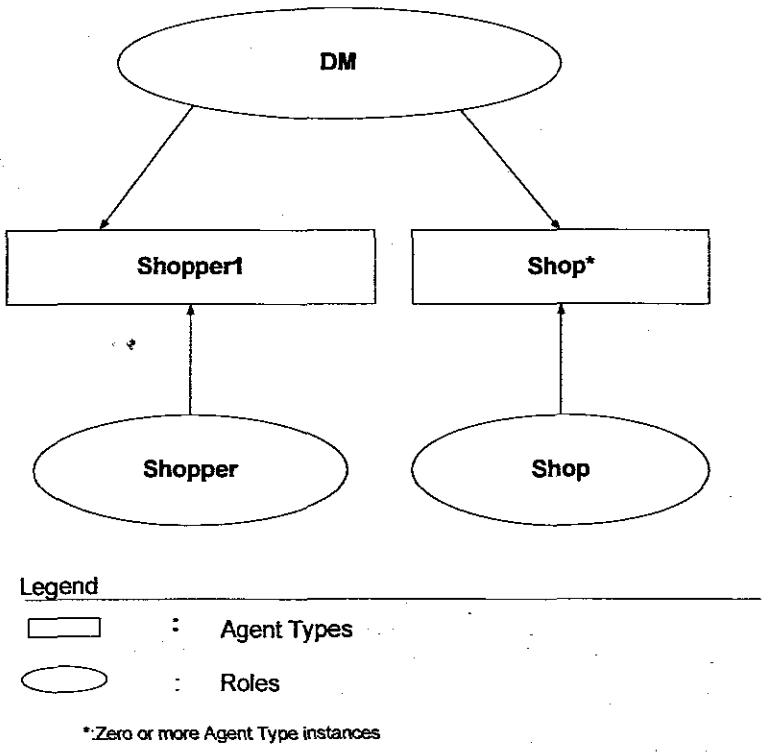


Fig.3.11 The agent model

From the foregoing analysis, a service model was formulated.

3.3.4.2 The Service Model

The aim of the services model is to identify the services associated with each agent class, or equivalently with each of the roles to be played by the agent classes. The services that compose an agent are derived from the list of protocols, activities, responsibilities, and liveness properties of the roles it implements. Table 3.1 shows the service model of the system. This model defines the input and output elements of the identified service. Furthermore, it also states the pre and post conditions for that specified service.

Table 3.1 The service model

Service	Inputs	Outputs	Pre-Conditions	Post-Conditions
Find shops	Product descriptions + security properties	Products from a service provider with compatible security properties	ShopperAgent exist and security characterization tests performed	The appropriate service provider agent is selected or the service terminates

Finally, the acquaintance model is defined. This model depicts the roles that interact when agents are fulfilling their design purposes. It also takes into account the idea that an agent can interact with another agent without having any knowledge about that agent. Therefore, this model doesn't only define interactions, but it also specifies whether or not one role is acquainted with another. Table 3.2 shows the acquaintance model of the system, where *I* depicts the agents that interact with each other and *A* depicts agents that are acquainted with each other.

Table 3.2 The acquaintance model

	Shopper	Shop	DF
Shopper		I,A	I,A
Shop	I,A		I,A
DF	I,A	I,A	

3.3.5 Detailed Design

When moving from the Gaia model towards the system implementation, messages that are communicated in this system are defined. To achieve this, the FIPA ACL Message Structure Specifications in [40] was employed. The ACL Messages **RequestShops** and **RespondShops** are presented in Table 3.3.

Table 3.3 ACL Message Definition

ACL Message: RequestShops **Sender:** ShopperAgent **Receiver:** ShopAgent **FIPA performative:** REQUEST **Protocol:** RequestShops **Language:** SL **Ontology:** sell-product **Content:** Ontology action: RequestShops	**ACL Message:** RespondShops **Sender:** ShopAgent **Receiver:** ShopperAgent **FIPA performative:** INFORM **Protocol:** RespondShops **Language:** SL **Ontology:** sell-product **Content:** Ontology concept: Shops

At this stage, the internal structures and methods of the system are defined. For this system, the following structures and methods are obtained:

i. The user profile structure contains all the information there is to know about the user, and how it is organized. The Shopper role maintains this structure (see Fig. 3.7), and

ii. The shop structure contains all the information about the shops. It defines the shop and the attributes associated with it. This structure is needed by the Shop role and the Shopper role, the former instantiates such objects by the information that it gets from

the DM (**QueryDM** activity), while the latter filters the shop structure objects according to the user profile (**InferUserNeeds** activity).

From the above analysis, the agent class diagram is created (see Fig. 3.12); this diagram depicts the classes and their attributes and operations. To model the class diagram for classes in this system, the AUML is employed. This allows for the specification of the agent specific attributes.

The class diagram in Fig. 3.12 shows that there is a capability "Buy" in the system. That capability is owned by the ShopperAgent. There is also a capability "Sell" which is owned by the ShopAgent. Moreover, there is another capability "CheckSecurityProperties" which is owned by both the ShopperAgent and the ShopAgent. The ShopAgent implements a "Shop" interface. The ShopperAgent requires a "FindShops" service that is provided by the ShopAgent.

Finally, a flowchart diagram is used to depict the RequestShops behavior of the system. The flowchart diagram enables the observation of the information exchanged between entities in the system. This also enables the system to view what behavior is next to be added in the agent's scheduler. Fig. 3.13 is the flowchart diagram of the RequestShops behavior.

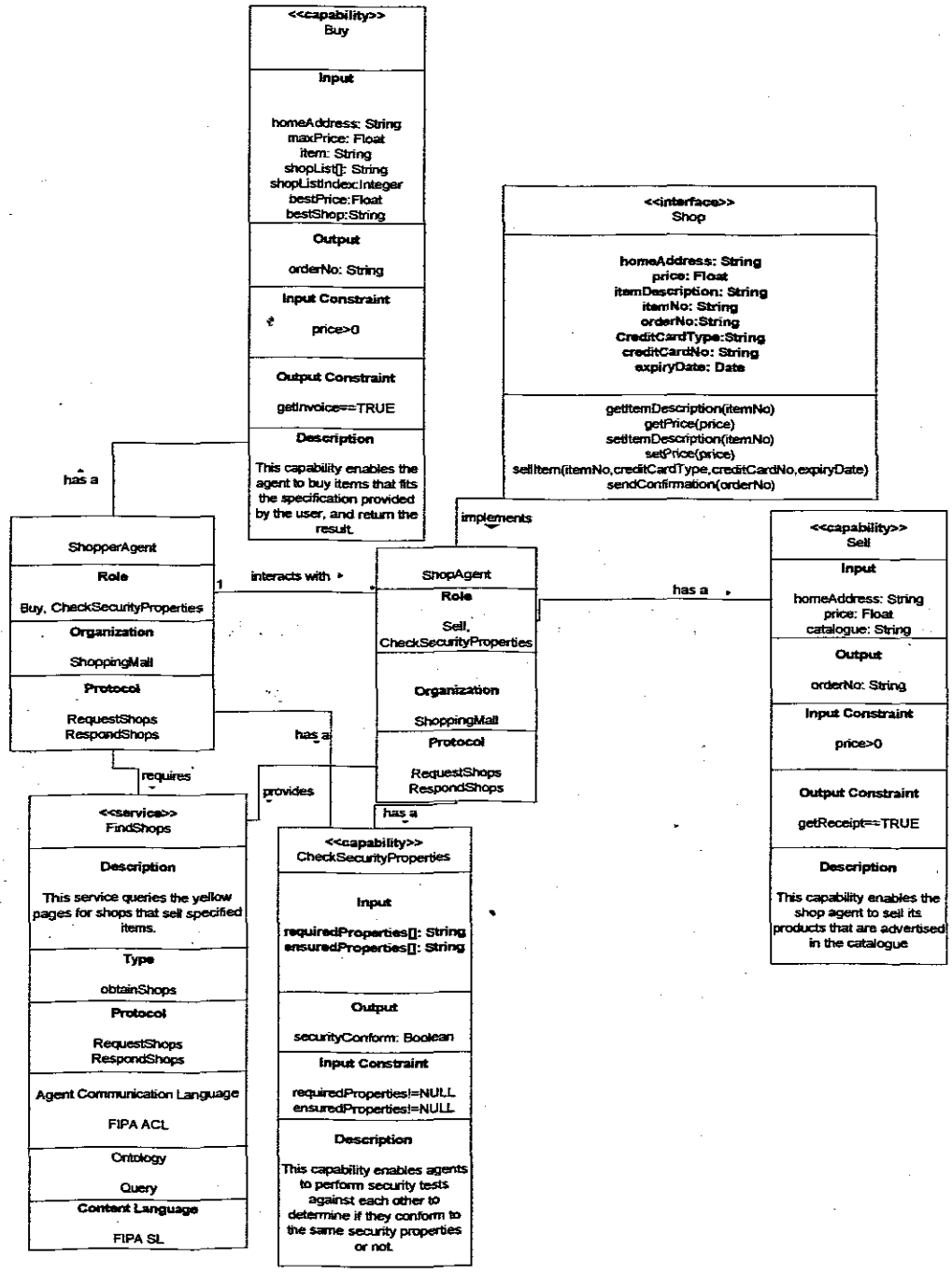


Fig.3.12 A class diagram for the shopping mall system

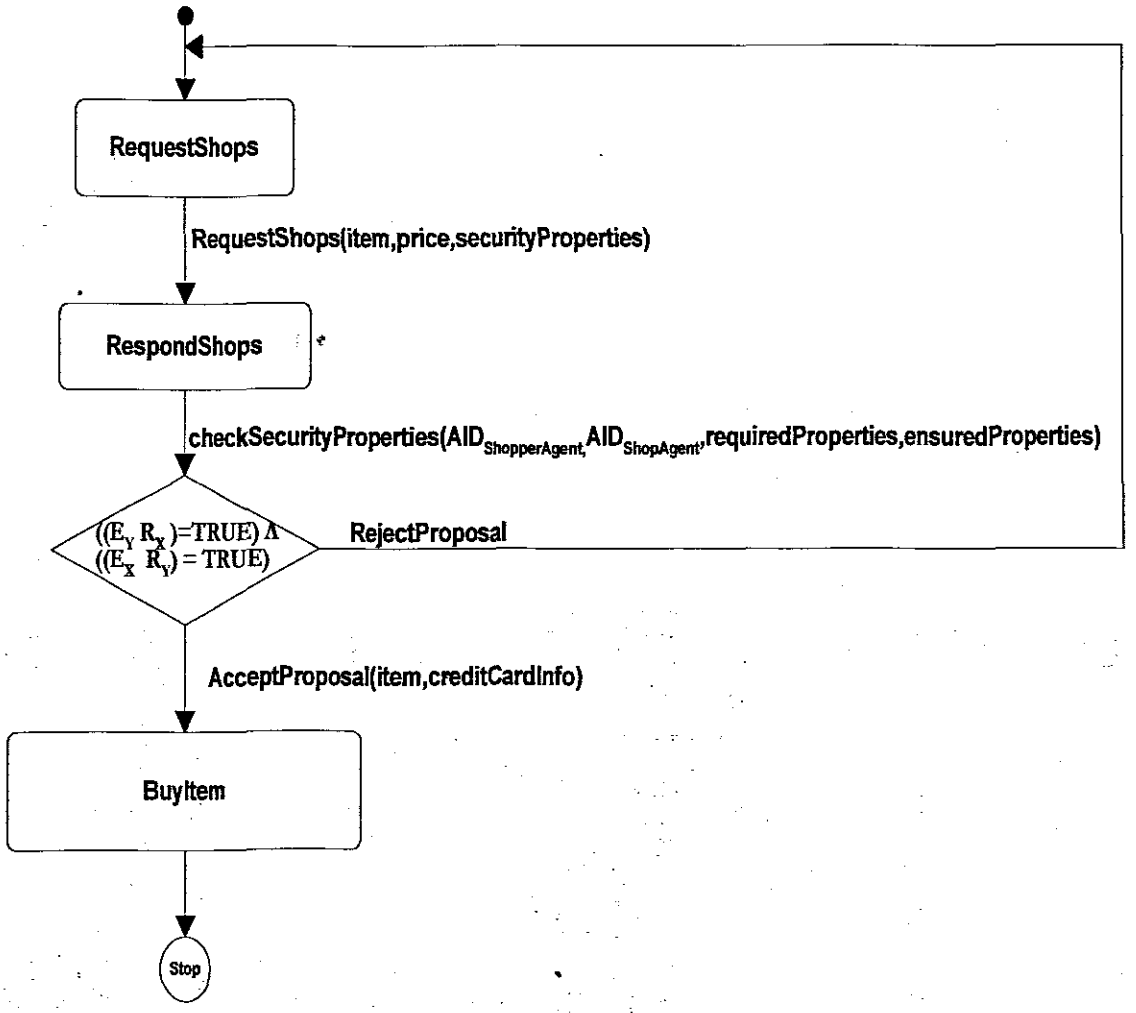


Fig. 3.13 A flowchart diagram for the RequestShops behavior

3.4 The Simulation Model and Design

At this stage the design of the program that simulates the proposed ISC mechanism is presented. In this work it is assumed that the agents in the system are in a controlled environment, i.e. access control, authentication and encryption mechanisms are in place. It is also assumed that all the agents in this system report their trust information truthfully. The first

step towards the simulation of the system is the identification of the objects required to build the model.

3.4.1 The Simulation Model

The simulated system consists of a number of agents interacting to achieve their design goals. This number is varied to monitor the behavior of agents in the system. All the agents in the system are incorporated with a number of security properties, and they have no knowledge of each others past behaviors. It is assumed that there is only one service in this testbed so all agents in the system are communicating for the same service. Validating the compatibility of an agent and other agent's properties is the key factor in determining whether an agent can trust another agent to provide a service without compromising the service requestor agent.

Agents in this system can either be in one of four states, i.e. they can be in a sending state, waiting for communication state, validating properties then respond according to the acquired result state, or exiting the platform state. Each agent has a radius of conversation, which represents the permitted vicinity of the agent's interactions with its neighbors. Simulations are run in a testbed in rounds of agent interactions. A client agent broadcast a service request message that is received by all service provider agents in its radius of operation. The service providers respond to the broadcasted message. The selection of the service provider depends on the outcomes obtained from the security evaluations performed by the client agent. This prevents agents from randomly selecting the service providers without knowing their security status.

3.4.2 The Simulation Design

To evaluate the scheme, a simulation model consisting of an agent platform class, agent class, message class, knowledge base, and the security property class was created. All the classes that were used in the simulation and implementation were written in the JAVA programming language, in the JBuilder 5 environment. Java was chosen as a programming language in this study because of its object-orientedness, portability, multi-threadedness, and its security services. Below are classes that are prevalent in the system.

The Agent Class

The agent class is an abstract class that defines the minimum requirements of agents in the system as shown in Fig. 3.14. The agent class provides basic operations performed by agents, such as registering on a platform after arrival, deregistering from a platform executing. Agents in the system can either be client agents or service provider agents; this is depicted in the specialization relationship of the agent class.

The Message Class

The message class defines the information used in the message passed back and forth between agents. It must be noted that the content of the first request message is combined with the security specification of the agent anticipating collaboration.

The Platform class

The platform class provides an execution environment for agents.

The security property class

This class hosts the functionality specific security properties of an agent. There are two security properties that must prevail for each agent, i.e. the required and the ensured security properties that are communicated to check security conformity. After the security compatibility of agents has been validated, the results obtained are stored in the security knowledge base.

The knowledge base class

This class serves as a repository for security contracts formed by any two or more agents in the system when interacting. This class stores these agreements for future references, so that agents don't perform security tests on each other every time they anticipate to collaborate. If an agreement has been formed between any two or more agents, then the involved agents can simply communicate the nest time they meet. It should be noted that in this study it is assumed that the security properties do not change.

A sequence of messages that are communicated in the system can be viewed in Fig. 3.4, from section 3.2 above, where agents broadcast service request messages and select a service provider agent that meets a similar set of required and ensured security properties.

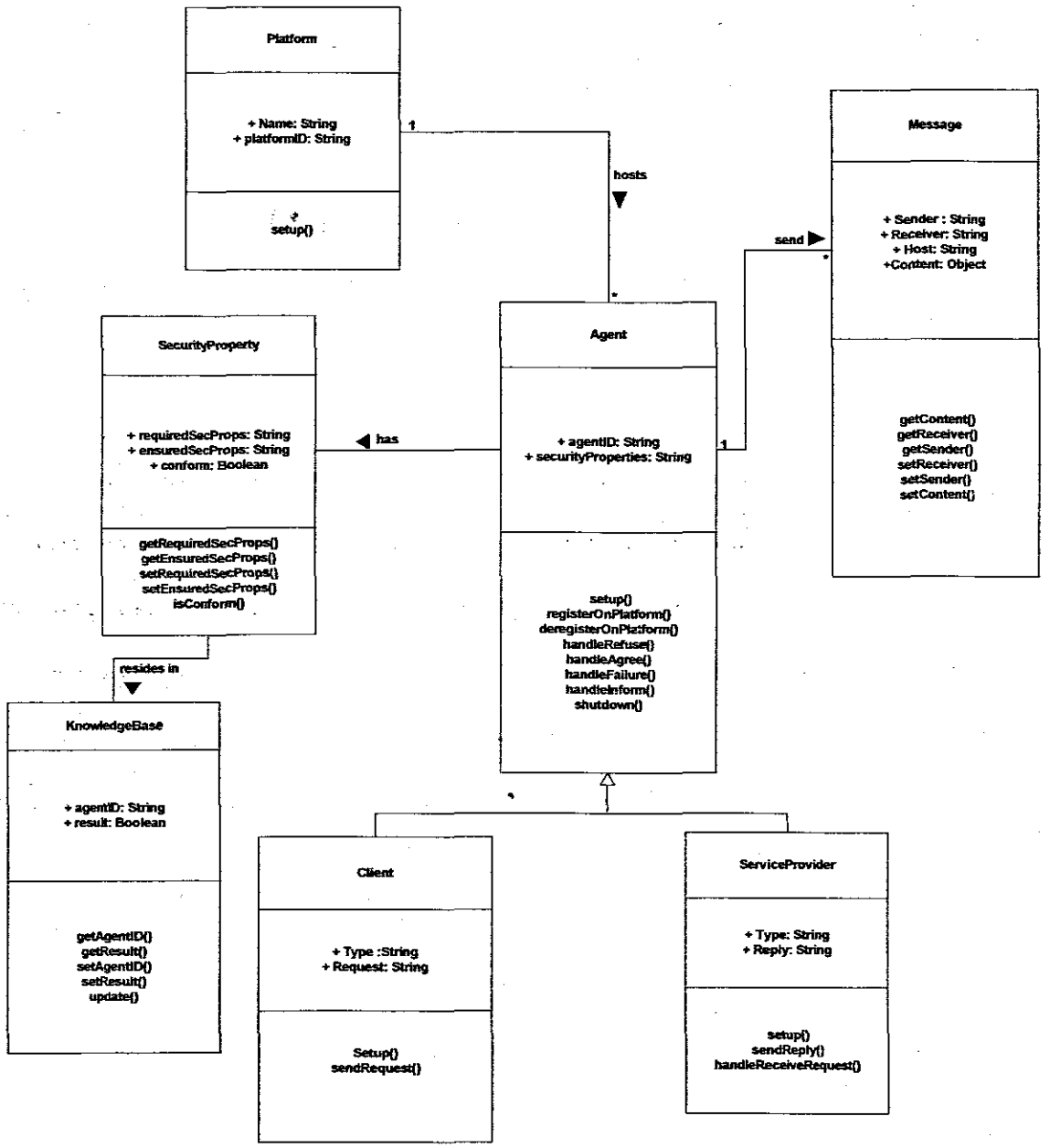


Fig 3.14 The class diagram for the simulator

3.4.3 The Simulation Environment

The simulator depicted in Fig. 3.15 was designed to show the agents in the system, and the messages that are communicated in the system. The agent states are color-coded such that one can tell the state an agent is in at a particular time. In order to expose the workings of the system, a status window was used to display messages as they are exchanged by agents in the system. The color changes are a reflection of the state changes. In the simulator the start button enables the simulation to commence. When the simulation commences, agents arrive in the platform with their security properties. After arrival, service request messages are broadcasted. Service provider agents validate the security properties of the client agent, and send replies if the client agent meets their security requirements. Fig 3.15 shows agents sending messages to other agents within their radius of communication. The status window is filled with messages that allow viewing the actual working mechanism of the scheme.

3.5 Summary

In this chapter trust issues in multi-agent systems have been presented, and the importance of agents to have a capability to assess trustworthiness of other agents in the system based on their own findings have been emphasized. The proposed security characterization framework was expatiated upon. This framework enables agents to incorporate security properties and publicize them as part of their interfaces. Therefore, an interaction contract between two agents will only be formed if the publicized security properties of one agent conform to the publicized security properties of another agent. The security issues in this research framework have been investigated with a view to yielding a trust support system in agent environments.

The system design for the agent oriented shopping mall system was demonstrated. The modeling abstractions from Gaia methodology, UML, AUML, and FIPA specifications were combined to yield a comprehensive model of the system. This enabled seamless transitions from analysis up to the implementation models of the system. Furthermore a simulation model and its design were presented.

Next, is the presentation of the simulation and implementation of the system, and present the findings.

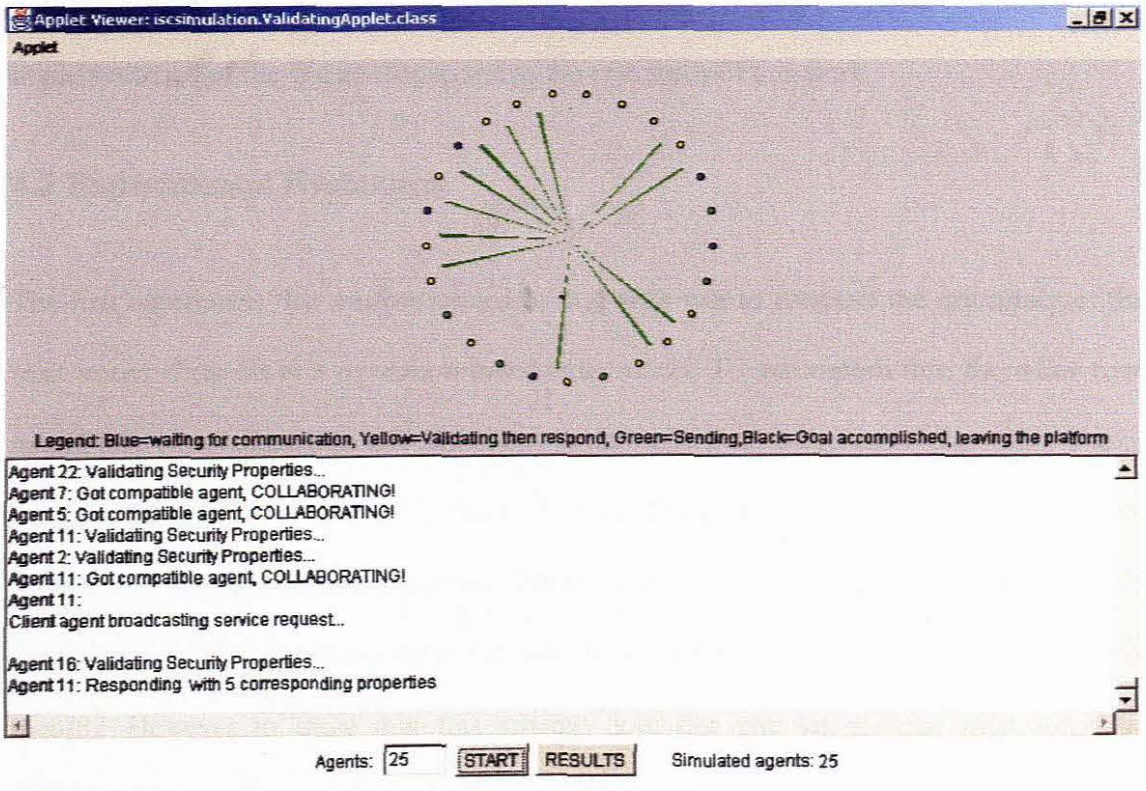


Fig. 3.15 The simulator for the ISC mechanism.

CHAPTER FOUR

4.0 MODEL SIMULATION AND IMPLEMENTATION

4.1 Introduction

This chapter is divided into two sections, section 4.2 and section 4.3 respectively. In section 4.2, the presentation and the evaluation of the results is given. Section 4.3 presents the implementation of the system that served as the case study of this work.

4.2 Experimental Evaluation

The first experiment that was performed in this work was to establish the inaccuracy of the trust values obtained in a reputation based trust model. To accomplish this, the e-Bay trust model [57] was evaluated. In this model buyers and sellers can assign each other either one of the three trust values i.e. [-1, 0, 1], where -1 means absolutely negative, 0 means uncertain or neutral, and 1 means absolutely positive. These values are assigned based on the outcome of the transaction. The reputation value is computed as the sum of those ratings over the past six months. However to show that this scheme does not give an accurate trust value, an experiment was run where an agent which occasionally performs maliciously was involved in twenty five transactions. The behavior of the agent during a particular transaction was monitored and depicted as shown in Fig. 4.1. Furthermore a good agent was allowed to participate in ten transactions, and it was also rated after each transaction. However, the

malicious agent got a higher trust value compared to its good counterpart since it was involved in more transactions than the good agent. This was because the total number of transactions performed truthfully masked the instances where the malicious agent performed dismally. The trust values obtained are depicted in table 4.1. These values portray an image that the malicious agent is good, and the good agent is malicious. Hence in a reputation based scheme, malicious agents can simply disguise their misconduct by being involved in a lot of transaction and therefore score high trust values as illustrated in Fig. 4.1. This then led to the proposal of the contract based trust mechanism.

Table 4.1 The e-Bay trust Model

Transactions	1	2	3	04	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Malicious	1	1	1	-1	1	1	0	1	1	1	1	1	-1	0	1	1	0	1	1	1	-1	1	1	1	1
Honest	1	1	1	1	1	1	1	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

In the proposed scheme, when each agent arrives at the system to execute, it has a specific value of security properties that it requires from and ensures to other agents in the system. The client agent has a set of functionality specific required and ensured security properties that it requires from and ensures to service providers respectively. Therefore, the client agent checks the compatibility of properties from service providers. If it finds a service provider with compatible properties, the collaboration can commence.

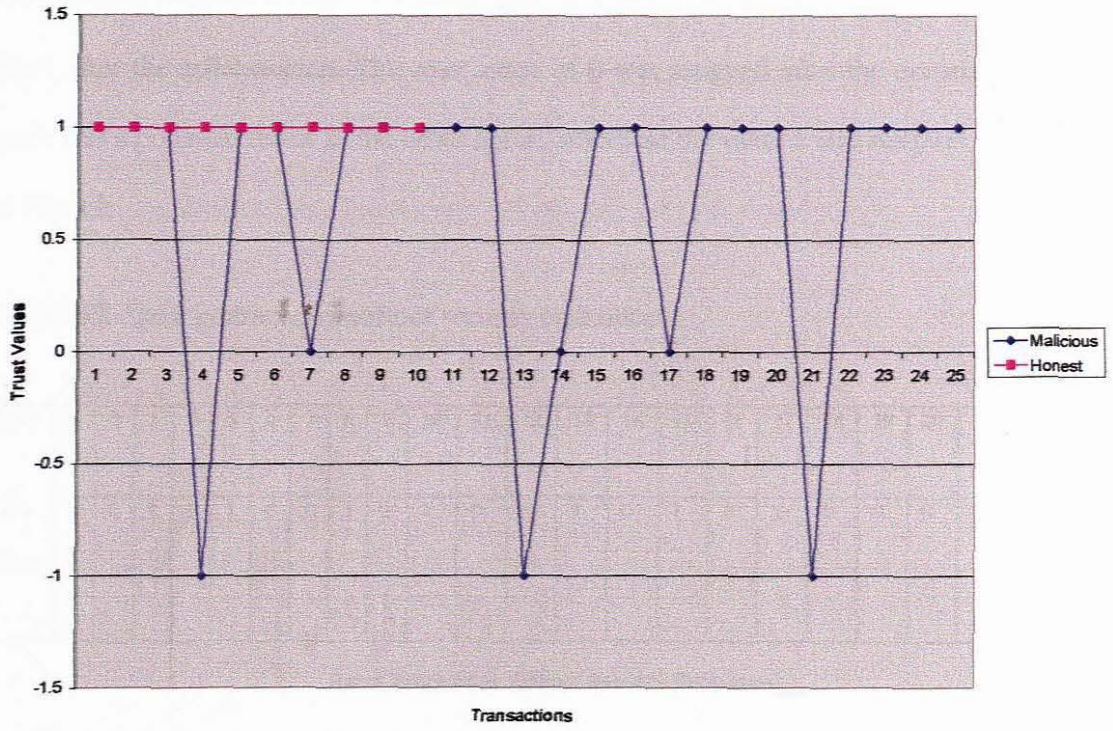


Fig. 4.1 The e-Bay trust model

To test the proposed scheme, two sets of experiments were performed to establish the ISC feasibility and accuracy. The first experiment evaluated the effect of agents interacting randomly without any security validation mechanism. To accomplish this, a set of twenty five agents that do not compare their security properties was set. Agents did not perform any validations on each other's security related characteristics, therefore, they collaborated freely. This then led to agents collaborating with agents without property compatibility. This therefore, implies that agents collaborated with agents who may be malicious, since their security status was not verified. Therefore, this experiment yielded low instances of trustworthy interactions as illustrated in table 4.2. This was derived from the results obtained from agents since they were monitored after an interaction, and it was found that agents had collaborated with agents who would have otherwise not collaborated with if they validated

their security properties status. The value pair [0, 1] was used to indicate the trust value of an agent after the collaboration. The trust value of 0 was assigned after the occurrence that an agent was a potential threat to the other agent; otherwise the value 1 was assigned as illustrated in Fig. 4.2.

Table 4.2 Agent interactions without security evaluations

Agent	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Trust Value	0	0	1	0	1	0	0	1	1	1	0	0	1	0	1	1	1	0	0	0	1	0	1	1	0

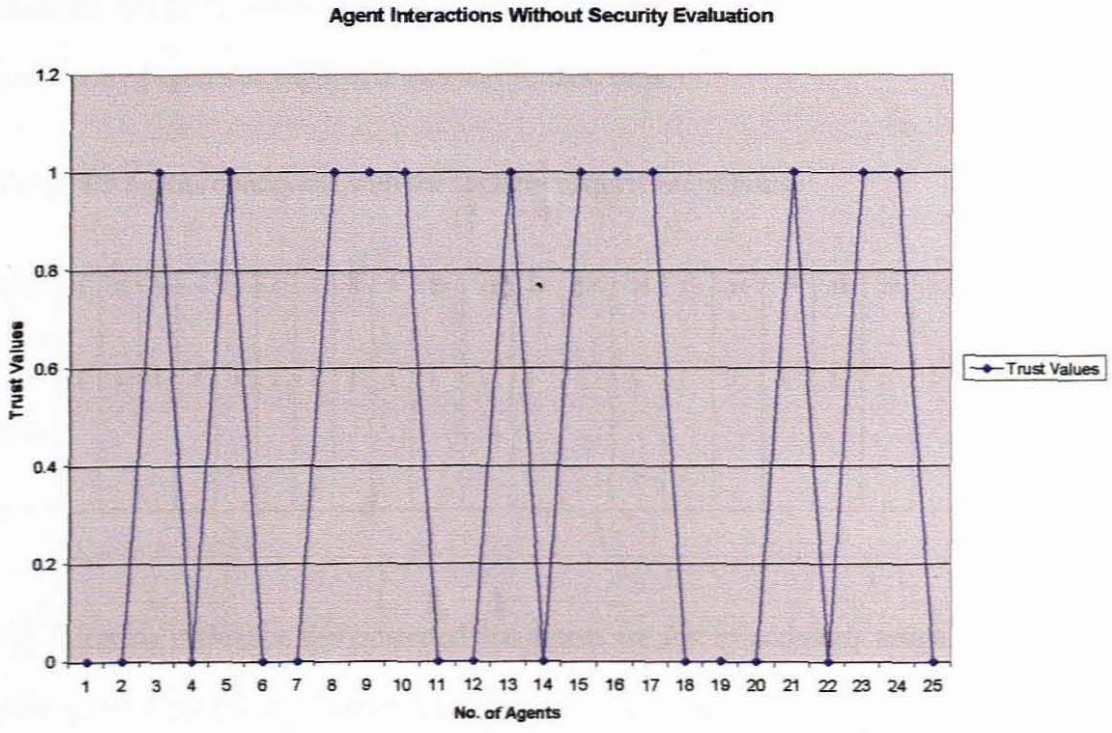


Fig. 4.2 Agent interactions without security evaluations

The number of 0 values obtained in Fig. 4.2 shows that an agent was not safe and could have been compromised in the real world. Therefore, it is more appropriate for agents to only collaborate with agents whose security properties are public; hence they are able to protect themselves against malicious agents through the security assessments.

The second experiment evaluated the accuracy achieved in a system whose agents are instrumented with an ISC mechanism. The agents in this system are able to evaluate each other's trustworthiness based on the publicized properties completely. So agents in this system could evaluate each other's trustworthiness at a 100% level of accuracy. It should be noted that in this system it is assumed that agents publicize their security properties truthfully. Table 4.3 illustrates the trust values obtained in an ISC-enabled interaction and Fig. 4.3 shows trust evaluation of agents incorporated with an ISC mechanism.

Table 4.3 Agents configured with the ISC mechanism interactions

Agent	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Trust Value	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

The foregoing evaluation demonstrated that agents are able to accurately assign trust values to other agents if the ISC mechanism is employed. Therefore we advocate the employment of the ISC mechanism as a trust guaranteeing mechanism in agent based systems.

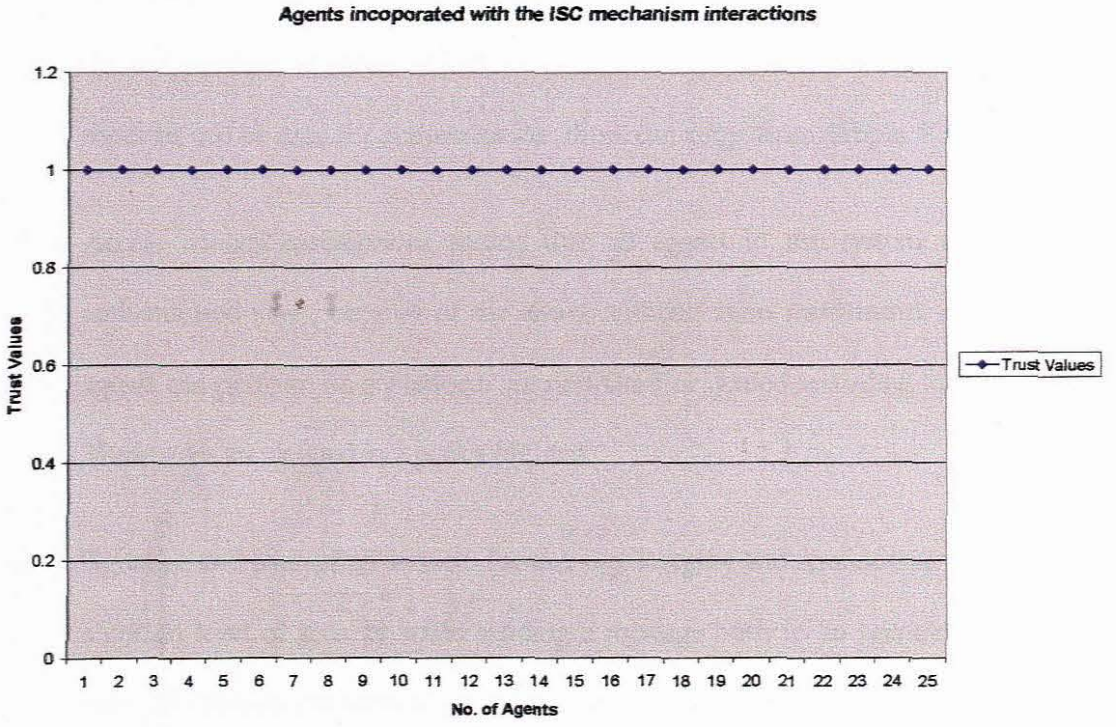


Fig. 4.3 Trust accuracy levels using the ISC approach

4.3 Implementation

The implementation of the system comprised of setting up an agent environment, where the shop and shopper agents were hosted. We employed JBuilder 5, to create agents and test the mechanism on a Pentium4 CPU, and Windows XP as our operating system. To realize the proposed concept a few assumptions were made for the agent platform. The platform is assumed to provide the following basic security services:

i. Authentication service to provide a guarantee that a user starting a platform is authenticated, and therefore agents within that platform are considered legitimate within the secured scope of the computational system hosting the main container of

the platform. Hence, it is assumed that the agent platform has a capability that enables the enforcement of differentiated access control on system users. A selection of login modules can be used for authentication, these can range from Simple, Kerberos etc.;

ii. Access control assumptions means that all agents in the system are owned by authenticated users, because of the above authentication mechanism, all actions that agents can perform on a platform are permitted or denied according to a set of rules. These rules are defined in a policy file, and

iii. Signature and encryption services for message integrity and confidentiality to guarantee a certain level of security when sending a message both to an agent running on the same or a foreign platform. Digital signatures are a well-known safeguard to ensure the integrity of a message and the identity of the message originator. Encryption on the other hand ensures the confidentiality of the message by protecting message data from eavesdropping.

All the agents in this system are configured with different security properties which are hosted as databases of required and ensured security properties. These properties are invoked when an agent is initialized, the actual execution is delayed until the proper security tests have been done and results obtained. The internal working of the simulator is illustrated next.

A shopper agent is dispatched to go shopping in the shops each wrapped by its ShopAgent. Fig.4.4 and Fig.4.5 shows a ShopperAgent being instantiated and dispatched by the customer.

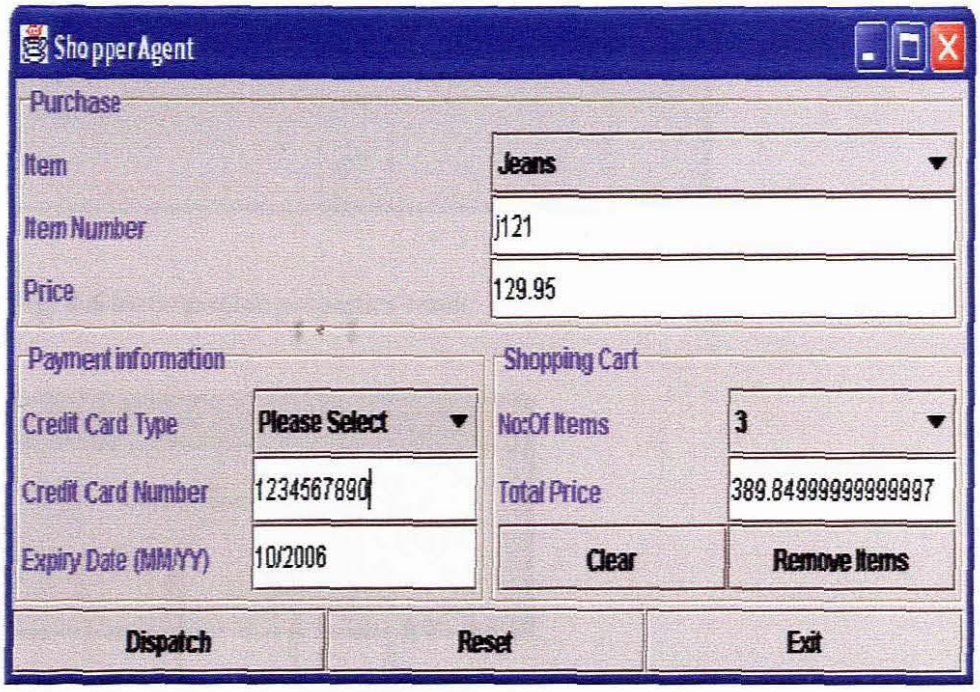


Fig. 4.4 The ShopperAgent interface

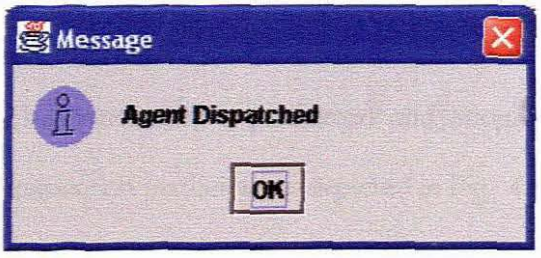


Fig. 4.5 The dispatching of an agent

In this system all agents are specified in such a way that they halt all their functionality related goals until the security analysis completes. However, to show the results of the test, a text message was enabled to depict the result of obtained in each test. These are depicted in the Fig. 4.6 and Fig. 4.7.

Fig. 4.6 Incompatible properties result

Fig. 4.7 Compatible properties result

The above results show that ShopAgent has compatible security properties. Therefore after the security analysis completes, the ShopperAgent then purchases an item through ShopAgentB. The transaction is performed and ShopAgentB, returns with the invoice to confirm the transaction performed as depicted in Fig. 4.8. below.

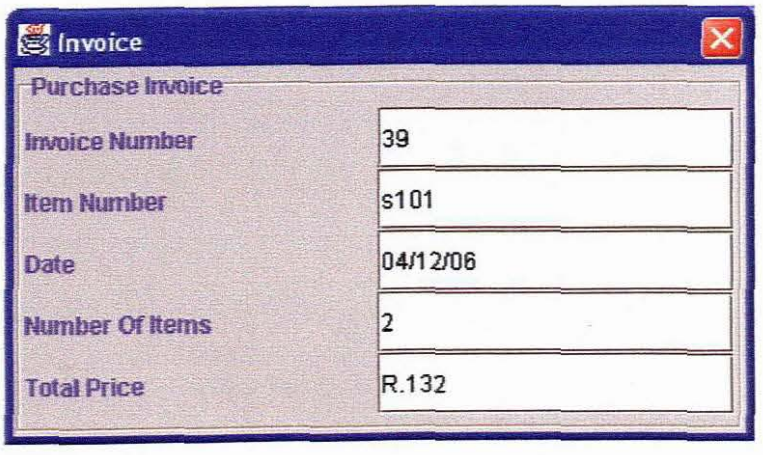


Fig. 4.8 The returned invoice

4.4 Summary

The results obtained from the evaluation of the trust mechanism have been demonstrated. The implementation of the proposed security scheme has been presented using the Shopping mall case study. The proposed mechanism clearly depicts that an agent is empowered to make trust decisions based on its own discretion. Therefore, trust relationships formed as a result of the proposed mechanism are based on accurate trust values compared to trust relationships formed and based on trust values obtained from third party entities.

Chapter 5 presents conclusions of this work and some possible direction for future work.

CHAPTER FIVE

5.0 CONCLUSIONS AND FURTHER WORK

5.1 Conclusions

The openness of multi-agent systems makes them susceptible to attacks. As long as this is the case, multi-agent systems may not be able to deliver on current promises. Therefore, research has recognized the need for MAS to implement security properties such as integrity, confidentiality, accountability, anonymity, and availability of agents and platforms. However the question of trustworthiness of agents by making them security-aware at runtime is yet to be addressed. In this regard there is a similarity between the security need of components in a component based system and security of entities in a multi-agent system. The former was addressed by Khan et al [15], while the latter has been addressed in this dissertation. Just as a component's integrity needs to be declared upfront before reuse, so an agent needs to declare its security properties and also have access to other agent's security properties before collaboration.

Therefore, the first objective of this research was to formulate a model to enforce security contract negotiation between two interacting agent entities. Following the security-awareness mechanism for components by Khan and Han [15], an Interaction Security Contract (ISC) was formulated for entities in an agent based system.

Secondly the work sets out to simulate and implement the model developed. This has been achieved by implementing a demonstration system known as a shopping mall system. Moreover, the ISC mechanism has been experimentally compared with the reputation based scheme in the same category.

The limitations of the simulated and implemented systems are as follows:

i. The systems only allow complete trust or no trust at all. This does not give an agent the liberty to continue with the collaboration even if the level of trust is not complete but acceptable;

ii. Agents in the systems can validate each other but they cannot validate platforms, this leaves room for platforms to compromise agents. Therefore there is a need to extend this scheme to also include security validations for agents against platforms, and

iii. There is no mechanism to validate whether the security properties publicized by agents are trustworthy, since this work assumed that agents publicize their security properties truthfully.

The foregoing limitations provide some work designated for the future.

5.2 Further Work

The current binary status of the ISC mechanism can be improved upon by upgrading the mechanism to a non-discrete system such that ISC is computed as a percentage. Should this be the case, an entity will be at liberty to define a safe ISC percentage range under which it can afford to collaborate.

The present implementation is limited to agents. It is envisaged that a future extension will allow platforms to be instrumented with the ISC mechanism.

REFERENCES

1. Minar N, "Designing ecology of distributed agents. http://www.media.mit.edu/nelson/.

2. Smith D, Cypher A, Spohrer J, "Programming Agents Without a Programming Language". The Communications of the ACM, 1994, 37(7), pages 55-67.

3. Selker T, "A Teaching Agent that Learns". Communications of the ACM, 1994, 37(7), pages 92-99.

4. Riecken D, Architecture of Integrated Agents". Communications of the ACM, 1994, 37(7), pages 107-116.

5. Coen M.H, "SodaBot: A Software Agent Environment and Construction System, MIT AI Lab Technical Report 1493, June 1994.

6. Maes P, "On Software Agents: Humanizing the Global Computer", IEEE Internet Computing, Vol. 1, July/August 1997, pages 10-19.

7. Franklin S, Graesser A, "Is it an Agent or Just a Program? A Taxonomy for Autonomous Agents". In J.P.Müller, M.J. Wooldridge, N.R Jennings, editors, Intelligent Agents III, In Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Lecture Notes in Artificial Intelligence,1193, Springer-Verlag, pages 21-35.

8. Weiss G, "MultiAgent Systems a modern Approach to Distributed Artificial Intelligence". MIT Press, March 1999.

9. Ng S, " Protecting Mobile Agents Against Malicious Hosts", M.S.c. thesis, Division of Information Engineering, The Chinese University of Hong Kong, June 2000.

10. Jennings N.R and M. Wooldridge, "Intelligent agents: Theory and practice," The Knowledge Engineering Review, vol. 10, no. 2, pp. 115–152, 1995.

11. Chess D, Harrison C, Kershenbaum A, "Mobile agents: Are they a good idea?" In Jan Vitek, Christian Tschudin (eds.), Mobile Object Systems: Towards the Programmable Internet, pages 25-45. Springer-Verlag, April 1997. Lecture Notes in Computer Science No.1222. http://www.research.ibm.com/massive/mobag.ps (1994 version).

12. Chess D, Harrison C, Kershenbaum A, "Mobile agents: Are they a good idea?" IBM Research Report 1995.

13. Farmer W.M, Guttman J.D, Swarup V, "Security for Mobile Agents: Issues and Requirements".
http://csrc.nist.gov/nissc/1996/papers/NISSC96/paper033/SWARUP96.PDF.

14. Gray, R, Kotz D, Cybenko G, Rus D, " D'Agents: Security in a Multiple-Language, Mobile-Agent System", in Giovanni Vigna (Ed.), Mobile Agents and Security. pages 154-187. Springer-Verlag, 1998.

15. Khan K, Han J, "A Security Characterization Framework for Trustworthy Component Based Software Systems". 27th Annual International Computer Software and Applications Conference (COMPSAC), Dallas, 2003.

16. Object Management Group (OMG). The Common Request Broker: Architecture and Specification (CORBA), revision 2.2. http://www.omg.org/corba/corbaiiop.htm

17. Roger Sessions, "COM and DCOM: Microsoft's Vision for Distributed Objects", John Wiley & Sons, 1997. ISBN: 0-417-19381-X.

18. RMI: Remote Method Invocation. http://java.sun.com:80/products/jdk/rmi/index.html.

19. Tanenbaum A.S, Van Steen M, "Distributed Systems Principles and Paradigms, Prentice-Hall Inc, 2002, New Jersey.

20. Mc Mahon P.V, "SESAME V2 Public Key and Authorization Extensions to Kerberos", ISOC Symposium on Network and Distributed Systems Security, IEEE Computer Society Press, February, 1995.

21. Kohl J, Neuman B, "The Kerberos Network Authentication Service (V5)", Internet RFC-1510, September, 1993.

22. Vandenwauver M, Govaerts R, Vandewalle J, "Public Key Extensions used in SESAMEV4", Public Key Solutions '97, Toronto, April, 1997.

23. Farmer W, Guttmann J, Swarup V, "Security for Mobile Agents: Authentication and State Appraisal. In Proceedings of the 4th European Symposium on Research in Computer Security (ESORICS), Springer-Verlag, pp118-130, 1996.

24. Giansirascusa M, "Mobile Agent Protection Mechanisms, and the Trusted Agent Proxy Server (TAPS) Architecture". http://www.isrc.qut.edu.au/resource/techreport/qut-isrc-tr-2003-010.pdf.

25. Hohl F, "Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts". In G. Vigna, editor, Mobile Agents and Security. Springer-Verlag Berlin Heidelberg, 1998.

26. Karjoth G, Asokan N, Gulcu C, "Protecting the Computation Results of Free-Roaming Agents. Second International Workshop on Mobile Agents, Stuttgart, Germany, September 1998.

27. Lal M, Pandey R, "CPU Resource Control for Mobile Programs". Agent Systems and Applications , 1999, Pages 74-88.

28. Riordan J, Scheiner B, "Environmental Key Generation Towards Clueless Agents". http://www.schneier.com/paper-clueless-agents.pdf.

29. Roth V, "Secure Recording of Itineraries Through Cooperating Agents". In Proceedings of the ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations, pages 147-154, INRIA, France, 1998.

30. Sander T, Tschudin C.F, "Protecting Mobile Agents Against Malicious Hosts". In G. Vigna, editor, Mobile Agents and Security. Springer-Verlag Berlin Heidelberg, 1998.

31. Schneider, F, " Towards Fault-tolerant and Secure Agentry". In Proceedings of the 11th International Workshop on Distributed Algorithms, Saarbrücken, Germany, Sept. 1997. Also available as TR94-1568, Computer Science Department, Cornell University, Ithaca, New York. http://cs-tr.cs.cornell.edu:80/Dienst/Repository/2.0/Body/ncstrl.cornell%2fTR97-1636/postscript.

32. Vigna G, "Cryptographic Traces for Mobile Agents in G. Vigna (Ed): Mobile Agents and Security, pp 137-153, Springer-Verlag, 1998.

33. Westhoff D, Schneider M, Unger C, Kaderali F, "Protecting Mobile Agent's Route Against Collusions. In Proceedings of the SAC'99, Springer LNCS 1758, 1999.

34. Wilhelm U.G, Staamann S, "Protecting the Itinerary of Mobile Agents". In Proceedings of the ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations, INRIA, France 1998, Pages 135-145.

35. Yee B.S, "A Sanctuary for Mobile Agents". In Proceedings of the DARPA Workshop on Foundations for Secure Mobile Code Workshop, 26 - 28 March 1997. http://www.cs.nps.navy.mil/research/languages/statements/bsy.ps.

36. Zambonelli F, Jennings N.R., Wooldridge M, "Developing Multiagent Systems: The Gaia Methodology", ACM Transactions on Software Engineering and Methodology, Vol. 12, No. 3, July 2003, Pages 317-370.

37. FIPA, "FIPA Modeling: Agent Class Diagrams". http://www.auml.org.

38. Giunchiglia F, Mylopoulos J, Perini A, "The Tropos Software Development Methodology: Processes, Models, and Diagrams, in AAMAS02.

39. Wood M.F, Deloach S.A, "An Overview of the Multiagent Systems Engineering Methodology". AOSE-2000, The First International Workshop on Agent-Oriented Software Engineering. Limerick, Ireland, 2000.

40. FIPA: "FIPA ACL Message Structure Specification", 2002. http://www.fipa.org/specs/fipa00061/SC00061G.pdf.

41. Eriksson H, Penker M, "Business Modeling with UML", OMG Press, John Wiley &Sons, Inc.2000.

42. FIPA: "FIPA Personal Travel Assistance Specification", 2001. http://www.fipa.org/specs/fipa00080/XC00080B.pdf.

43. FIPA: "FIPA Quality of Service Ontology Specification" , 2002. http://www.fipa.org/specs/fipa00094/SC00094A.pdf.

44. FIPA, "FIPA Agent Management Specification, 2000. http://www.fipa.org/specs/fipa00023.

45. Resnick P, Kuwabara K, Zeckhauser R, Friedman E, "Reputation Systems: Facilitating Trust in Internet Interactions", Communications of the ACM, 43(12), PAGES 45-48.

46. Yu B, Singh M.P, "A Social Mechanism of Reputation Management in Electronic Communities". In Co-operative Information Agents, 7th International Conference, CoopIS 2000, 2000.

47. Zacharia G, Maes P, "Trust Management Through Reputation Mechanisms". Applied Artificial Intelligence, 14(8), 2000.

48. Abdul-Rahman A, Hailes S, "Supporting Trust in Virtual Communities. In Proceedings of the Hawaii's International Conference on Systems Sciences, Maui, Hawaii, 2000.

49. Zacharia G, "Collaborative Reputation Mechanisms for Online Communities". M.Sc. Thesis, Massachusetts Institute of Technology, 1999.

50. www.ebay.com.

51. www.amazon.com.

52. www.bizrate.com.

53. Gambetta D, "Can We Trust Trust?, in Making and Breaking Cooperative Relations, electronic edition, Department of Sociology, University of Oxford, chapter 13, pages 213-237, http://www.sociology.ox..ac.uk/papers/gambetta213-237.pdf.

54. Khan K.M, Han J, "Composing Security-Aware Software", IEEE Software, January/February 2002.

55. Khan K, Han J, Zheng Y, "A Framework for an Active Interface to Characterize Compositional Security Contracts of Software Components", IEEE Proceedings of the 13th Australian Software Engineering Conference (ASWEC'01), 2001.

56. Langley B.K, Paolucci M, Sycara K, "Discovery of Infrastructure in Multi-Agent Systems". http://www-2.cs.cmu.edu/~softagents/papers/infrastructureDiscovery.pdf.

57. Kollock P, "The Production of Trust in Online Markets". In Lawler E.J, Macy M, Thyne S, and Walker H.A. editors. Advances in Group Processes, volume 16, pages 99-123, JAI Press 1999.

58. Basis D, Doser J, Lodderstedt T, "Model Driven Security for Process-Oriented Systems", SACMAT'2003, June 1-4, 2003, Como, Italy.

APPENDIX

Below are code snippets taken from the simulation program written in Java. Listing 1 presents the code fragment to set the messages broadcasted incorporated with the required and the ensured security properties. Listing 2 presents the code fragment that begins the simulator, and listing 3 presents the code fragment for actually checking the security properties and taking a decision

```
public Agent(String n, Message required, Message ensured, Object ref) {
    name = n;
    interactions = new Message[2];
    interactions[0] = ensured;
    interactions[1] = required;
    validated = 0;
    properties = rand(Validating.BASIC_PROPERTIES - 2, Validating.BASIC_PROPERTIES + 2);
    observerRef = ref;
    myThread = new Thread(this, n);
    status = STATUS_WAITINGFORCOMMUNICATION;
}

int rand(int min, int max) {
    return (min + (int)(Math.random() * (max - min)));
}
void sleepFor(int ms) {
    try { Thread.sleep(ms * (validate?1:10)); }
    catch (InterruptedException e) { abort = true; }
}
public void setValidate(boolean v) { validate = v; }

public synchronized void start() {
    myThread.start();
}
public synchronized void stop() {
    abort = true;
    myThread.interrupt();
}
```

Listing 1. Code fragment to set the messages broadcasted incorporated with the required and the ensured security properties

```
public void beginSimulation() {
    int i;

    if (runningthreads > 0) {
        logarea.append("Aborting simulation before completion.\n");

        for(i = 0; i < broadcasters.length; i++) {
            broadcasters[i].stop();
        }
        runningthreads = 0;
    }
    host.clear();
    String cntString = counterField.getText();
    try {
        count = Integer.parseInt(cntString);
    }
    catch (NumberFormatException nfe) {
        count = DEFAULT_AGENT;
        counterField.setText("" + count);
    }
    logarea.append("\t\nBeginning simulation with " + count + " agents\n");
  logarea.append("\t\nClient agent broadcasting service request...\n");

    synchronized (host) {
        broadcasters = new Agent[count];
        conversations = new Message[count];
    }

    Dimension psize = host.getSize();
    Point center = new Point(psize.width / 2, psize.height / 2);
    rad = ((psize.width < psize.height)?
        ((psize.width * 4)/5):((psize.height * 4) / 5)) / 2.0;
    mesrad = (rad/count) * 2.1;
    convrad = rad - mesrad;
    convSent = new Point[count];
    int csx[] = new int[4];
    int csy[] = new int[4];
    conversationsSent = new Polygon[count];
    double phi;
    double div = (2 * Math.PI)/(count * 2);
    double swdiv = Math.PI / 150;
    int c;
    for(c = 0, phi = 0.0; c < count; c++, phi += div) {
        convSent[c] = new Point((int)(rad * Math.sin(phi)) + center.x,
                                (int)(rad * Math.cos(phi)) + center.y);
        phi += div;

    csx[0] = (int)( Math.sin(phi)) + center.x;
        csy[0] = (int)(Math.cos(phi)) + center.y;
        csx[1] = (int)(convrad * Math.sin(phi)) + center.x;
        csy[1] = (int)(convrad * Math.cos(phi)) + center.y;
        csx[2] = (int)(convrad * Math.sin(phi + swdiv)) + center.x;
        csy[2] = (int)(convrad * Math.cos(phi + swdiv)) + center.y;
        csx[3] = (int)(Math.sin(phi + swdiv)) + center.x;
        csy[3] = (int)(Math.cos(phi + swdiv)) + center.y;
        conversationsSent[(c + 1) % count] = new Polygon(csx,csy,4);
    }

    for(i = 0; i < count; i++) {
        conversations[i] = new Message("Message " + i, conversationsSent[i]);
        conversations[i].addObserver(host);
    }

    for(i = 0; i < count; i++) {
        broadcasters[i] = new Agent("Agent " + i,
                                    conversations[i],
                                    conversations[(i + 1) % count],
                                    convSent[i]);
        broadcasters[i].setValidate(validate);
        broadcasters[i].addObserver(this);
        broadcasters[i].addObserver(host);
    }
    for(runningthreads = 0; runningthreads < count; runningthreads++) {
        broadcasters[runningthreads].start();
    }
```

Listing 2. The code fragment that begins the simulator

```
public Agent(String n, Message required, Message ensured, Object ref) {
        name = n;
        interactions = new Message[2];
        interactions[0] = ensured;
        interactions[1] = required;
        validated = 0;
        properties = rand(Validating.BASIC_PROPERTIES - 2,    Validating.BASIC_PROPERTIES + 2);
        observerRef = ref;
        myThread = new Thread(this, n);
        status = STATUS_WAITINGFORCOMMUNICATION;
    }

    int rand(int min, int max) {
        return (min + (int)(Math.random() * (max - min)));
    }
    void sleepFor(int ms) {
        try { Thread.sleep(ms * (validate?1:10)); }
        catch (InterruptedException e) { abort = true; }
    }
    public void setValidate(boolean v) { validate = v; }

    public synchronized void start() {
        myThread.start();
    }
    public synchronized void stop() {
        abort = true;
        myThread.interrupt();
    }

    public void run() {
        message("Responding  with " + properties + " corresponding properties");
        int m1, m2;

        while(validated < properties && !abort) {

            status = STATUS_WAITINGFORCOMMUNICATION;
            setChanged(); notifyObservers(observerRef);
            //message("validating.....");
            sleepFor(rand(0,4) * Validating.BASIC_DELAY);
            message("Validating Security Properties...");

        // a service provider, has checked the security properties configured in the service
        //request message, and is responding with its own security properties

        status = STATUS_RESPONDING;
            boolean conform = false;
            setChanged(); notifyObservers(observerRef);
            while(!conform && !abort) {
                // send another message, randomly
                m1 = rand(0,3);
                m2 = ((m1 == 0)?(1):(0));

                // send message
                synchronized (interactions[m1]) {
                    while(!(interactions[m1].isAvailable())) {
                        try {
                            interactions[m1].wait();
                        } catch (Exception e) {
                            if (abort) return;
                        }
                        // required properties conform, now check for the ensured ones
                    }

                    interactions[m1].receive();
                }

                // Check the ensured security properties, but give up
                // immediately if they do not conform.
                synchronized (interactions[m2]) {
                    if (interactions[m2].isAvailable()) {
                        interactions[m2].receive();
                        conform = true;
                    }
                }
                // If we didn't manage to get a matching first(required) set of properties,
                // then don't go any further with checking the second(ensured) set
                if (!conform) {
                    interactions[m1].broadcast();
                }
            }
            if (abort) return;

            // If our agent has got this far, it means both security properties have been validated and
            // it is established that they conform, therefore the interaction can take place.
            status = STATUS_RESPONDING;
            message("Got compatible agent, COLLABORATING!");

            message("Got compatible agent, COLLABORATING!");

            setChanged(); notifyObservers(observerRef);
            sleepFor(rand(2,5) * Validating.BASIC_DELAY);
            validated += 1;
            if (abort) return;

            //  We have achived our design goal here,
            // we can send a new set of messages if there is more to do
            message("Objective achieved for the moment, send another message if there is more to do...");
            interactions[0].broadcast();
            interactions[1].broadcast();
            if (abort) return;
        }
        status = STATUS_DONE;
        setChanged(); notifyObservers(observerRef);
        message("Stop execution, and leave the platform.");
    }
```