MODELLING TRUST MANAGEMENT IN AGENT-TO-AGENT SECURITY SCENARIO

Mzomuhle Thuthuka Nkosi

(20011359)

A dissertation submitted in fulfilment of the requirements for the degree of

Master of Science in Computer Science

Department of Computer Science, Faculty of Science and Agriculture, University of Zululand

2006

Declaration

This dissertation represents author's own research work and has not been submitted in any form to other institution of tertiary education for another degree or diploma. All the material used as source of information has been acknowledged in the text.

Signature

Dedication

I dedicate this dissertation to my family which has always been behind me especially my mother. Her encouraging words have kept me going through difficult times. I also dedicate this work to my wife, who has been on my side continuously and has given me freak hope for success.

Acknowledgement

I would like to acknowledge my appreciation to all the staff members of the Department of Computer Science of the University of Zululand. I would also like to express my sincere gratitude to my supervisor, Prof. M.O Adigun, for his support and commitment to make this work a reality. To my fellow research students of the Department, I would like to thank them for their help. I would also like to thank all the sponsors for their financial contribution in conducting this research.

To God, the pillar of my strength, who always gives me a way through where it seems to be no way out, I am most grateful to Him.

Table of Contents

List of Figures

List of Tables

Abstract

Over the previous years, researchers have given much attention to explore the effective use of distributed systems to search, retrieve, and share information. Mobile agent technology has had a great impact in providing solutions in different dimensions of distributed systems. This research work presents the development and implementation of trust management model that ensures agents communicating in a Context-Aware Component Interfacing Pattern (CACIP) based Service Oriented Distributed Environment.

The Agent-to-Agent Trust Management model is proposed to ensure security of agents that want to access services from the CACIP. The developed model uses a reputation-based approach to evaluate trustworthiness of each agent. Each agent must be authenticated and authorized in order to get an access to services in CACIP. The trust management model ensures trustworthiness of each agent participating in a CACIP based environment. Mobile agent security requirements were considered in developing the model.

The developed model was implemented to demonstrate how requesting agents are authenticated and authorized before accessing services. Simulation of the model was also conducted to assess the performance to the model. The results show the efficiency of the model when agents simultaneously make requests over a given amount of time. The performance of the model shows scalability when number of requests increases.

The use of both foreign and local reputation to evaluate trust of agents guarantees that all agents that access services, are trustworthy.

CHAPTER ONE

1. Introduction

1.1 Overview

The advent of agent technology has provided improvements in communication of application users. Agent technology has been applied in several computer systems to enhance their performance. This technology has brought some distinctive changes in distributed and client-server computer systems. Agents are defined as computer programs that act in a computer network on behalf of human user or application. Agents are designed with specific goals to achieve. Autonomous agents have the capability of moving from one machine to the other and perform their duties without their owner's intervention, i.e. the owner of an agent sends the agent to go and perform a duty in another remote host according to the itinerary given to it. Agents communicate among themselves, to achieve their design goals, without the intervention of their owners. Agents with the capability of moving from one machine to the other are called mobile agents. Mobile agent technology has been efficiently used to improve users' performance in different contexts.

The need of agents to communicate with other agents in order to achieve their goals has resulted in the emergence of multi-agent systems. Multi-agent system is composed of several agents that are capable of mutual interaction. These agents interact to share information and services. Some agents act as service providers while others are service

consumers during their interaction.

Regarding the efficient utilization of mobile agent technology, there are some limitations, primarily in the area of security. Security in mobile agent technology has raised concerns about practical utilization of mobile agents. As the agent moves from one node to the other, it is open to many attacks which may destroy it or interrupt its execution. The issue of security in application of mobile agent is never easy to be ignored when successful operation is to be achieved. One dimension that impacts security of mobile agents is communication or interaction style when they serve their purpose. Therefore, appropriate interaction framework architecture is essential in order to establish proper security mechanism among agents. Context-Aware Component Interfacing Pattern (CACIP) architecture proposed in (Zuma and Adigun, 2006) provides the communication pattern for agents in multi-agent system to share services.

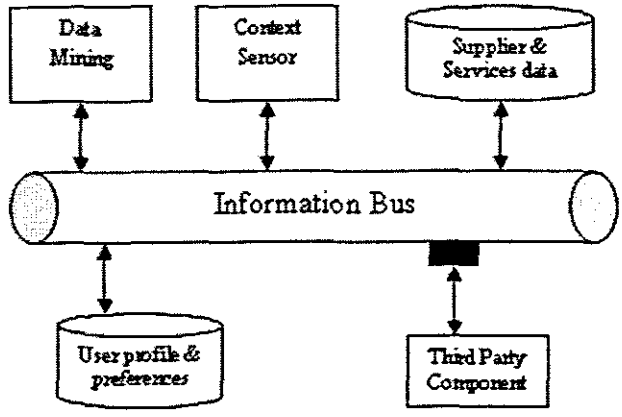

Figure 1. 1: The Context-Aware Component Interfacing Pattern (Zuma and Adigun, 2006)

The architecture is as shown in figure 1.1. The CACIP architecture manages inter-component communication through the information bus. Therefore, it prevents direct component-to-component communication; instead component interaction is managed by

the information bus. This architecture allows all participating components to register with the information bus, specifying types of messages they are interested in.

The components' registration ensures that every time a new message is written into the information bus, a component that is interested in it is notified. The way messages are accessed in CACIP architecture opens some serious security challenges or threats that have not yet been addressed.

Different approaches can be applied to provide a relevant security mechanism for mobile agents that communicate in a CACIP based mobile environment. Among several approaches that have been used by many researchers, trust has played a major role in provisioning of different security schemes. We propose a trust-based approach to provide security mechanism for mobile agents communicating in a CACIP based service oriented mobile environment. Winsborough et al. (2000) define trust as the degree of belief about the behaviour of other entities upon which we depend for reliability, timeliness, and integrity of message delivery to their intended next-hop. In this research, trust is defined as the degree of belief about the agent's behaviour upon which it is depended for reliability and trustworthiness in service or message accessibility. There is a great need to manage trust when trust-based approach is applied to security mechanism. We define *trust management* as activity of collecting, codifying, analysing and presenting security relevant evidence with the purpose of making assessments and decisions regarding service accessibility. The trust mechanism being proposed in this research would help to ensure that all agents that access services in CACIP are trust-worthy.

The concepts of trusted computing, trusted network, trusted communication, trusted agents are related to security issues, security technology and security services. All topics of security study and research are directed towards providing a secure and tamper free environment, or network or communication. In this context, trust is synonymous with secure, which is tied to security.

Open distributed systems can be modeled as open multi-agent systems (for example, the CACIP architecture) that are composed of autonomous agents interacting with one another via particular mechanisms and protocols. In this respect, interactions form the core of multi-agent systems. Thus, the agent research community has developed a number of models of interactions including coordination ((Jennings, 1993), (Durfee, 1999)), collaboration ((Pynadath and Tambe, 2002), (Cohen and Levesque, 1990)) and negotiation ((Rosenschein and Zlotkin, 1994), (Kraus, 2001), (Jennings et al, 2001)). However, their application in large-scale open distributed systems presents a number of new challenges. First, the agents are likely to represent different stakeholders that each has their own aims and objectives. This means the most reasonable design strategy for an agent is to maximize its individual utility. Second, given that the system is open, agents can join and leave at any time. This means that an agent could change its identity on re-entering and hence avoid punishment for any past wrong doing. Third, an open distributed system allows agents with different characteristics (e.g. policies, abilities, roles) to enter the system and interact with one another. Given this, agents are likely to be faced with a number of possible interaction partners with varying properties. Fourth, an open distributed system allows agents to trade products or services (e.g. various forms of auctions or market mechanism), and collaborate (e.g. forming coalitions or virtual

organisations) in very many ways. Therefore, agent designers are faced with a selecting from a number of potential interaction protocols that could help them achieve their design objectives.

Autonomous mobile agents in an open distributed system that requires a trust management that would make sure that every agent in the system is trustworthy.

1.2 Statement of the Problem

Applications running in a typical 4^{th} generation network will deploy mobile agents on an itinerary which requires them to visit fixed and mobile nodes. This poses a serious trust management challenge. In order to investigate and propose a security framework that responds to this challenge, we adopt the Context-Aware Component Interfacing Pattern (CACIP) architecture as the core of a middleware in which the security framework will be contextualised. This study constructed a trust management model that ensures security of agents communicating in a CACIP based Service Oriented Distributed Environment.

1.3 Research Goal

The goal of the study was to address the CACIP architecture's security when deployed in distributed mobile systems by providing agent based security mechanism.

1.4 Objectives

- To construct a trust management model with specialisation in authentication and authorization capabilities;
- To devise an agent-to-agent authentication and authorization mechanisms that manage trust in mobile systems and
- To show that the scheme or mechanism functions in a real environment by evaluating its performance experimentally.

1.5 Rationale of the Study

The CACIP architecture constitutes the core of a mobile application middleware on distributed systems. This raises many security concerns for agents that carry services from one node to another node. Adequate trust management is required in order to guarantee security of mobile services operating on a network. The results of this study would provide a mechanism that supports administration based on authentication and authorization scheme or framework for mobile agents in mobile application systems.

1.6 Research Methodology

A. Establishing the Background

- Literature search for existing security models designed for wireless networks and mobile ad hoc networks with the purpose of understanding how

authentication, authorization and privacy are being provided in applications running on those networks.

- Survey of relevant peer-to-peer agents' communication research in order to understand how authentication process is handled. This will be followed by subsequently by formulating an authentication scheme that draws on that understanding.

B. Formulation of a Trust Management Framework

This encompasses:

- Identifying existing reputation-based schemes that are used in different wireless applications to secure peers;
- Formulating of a model that ensures security based on trust among agents communicating in the CACIP manner on infrastructure-less networks. This entails considering appropriate trust establishing requirements and some security mechanisms that enforce those requirements and
- Constructing a corresponding framework for mobile agents' applications.

C. Proof of Concept Approach

Ordinarily this approach consists of:

- Analysing, designing and implementing the prototype to prove plausibility of the proposed model and
- Evaluation of performance of the prototype implementation.

1.7 Organisation of the Dissertation

The dissertation is organised as follows:

Chapter two presents background concepts that form the foundation of this research work. The chapter starts with brief introduction on reputation-based and policy-based trust management and security issues on mobile agents. Existing related works are discussed pointing out the challenges associated with the design of trust models. The chapter concludes with a brief overview of the proposed trust model.

In chapter three the description of the model development is presented. This chapter begins with the introduction followed by solution approach to solve the problem. The agent-based trust management model is then presented with its full details. Reputation mechanism comes as the conclusion of the chapter.

Chapter four is the description of design of the trust management system with the implementation. The experiments, analysis and results are presented in this chapter. It concludes with the performance evaluation of the model.

Chapter Five concludes the dissertation. The research conducted and described in the dissertation is presented in this chapter. Finally, the recommendations for future work are described.

CHAPTER TWO

2. Background Concepts and Literature Review

2.1 Introduction

The trust approach has been widely used by many researchers to tackle problem of security in multi-agent systems. Ramchurn et al (2004) defined trust in multi-agent systems as a belief an agent has that the other party will do what it says it will (i.e. being honest and reliable) or reciprocate (i.e. being reciprocative for the common good of both), given an opportunity to defect to get higher payoffs. Trust issues have generated active research interests for many researchers in attempting to propose security mechanisms for open distributed systems. Trust management has been an issue of much consideration for effective utilization of trust approach in the research community. An appropriate trust management to propose security scheme is essential because trust is evaluated on individual basis for a specific aspect. For example, component A, B and C are interacting. B may be trusted to purchase items but not trusted to download files from A. On the other hand, C is trusted to download files but not trusted to purchase items. Therefore, in order for B to download files from A, trust evaluation must be done. And trust evaluation must be conducted for C to purchase items from A. Thus, trust management must be appropriate and relevant for such situations.

Reputation and policy have become the fundamental concepts in building trust among interacting agents in multi-agent systems. Reputation is defined as an opinion or view of

someone about something (Sabater and Sierra, 2003). When reputation is used to propose security scheme, it must be apparent how reputation is collected to evaluate trustworthiness of agents. Policies to access resources must be well stated in order to protect multi-agent systems.

A Lot of work has been done on securing multi-agent application systems using reputation-based and policy-based approaches to manage trust among agents. This chapter reviews some existing security architectures and trust models that relate to our research. The chapter starts by discussing some background concepts of the research in Section 2.2. Section 2.3 presents the framework analysis of the literature followed by review of existing mechanism. Section 2.4 discusses trust-based security design challenges and the chapter concludes with the proposed scheme presented in section 2.5.

2.2 Trust Management in Open Multi-Agent Systems

Several researchers have broadly used trust approach to formulate security architectures for multi-agent systems. Trust is a belief or expectation someone has that the other party will behave according to its promises. Trust provides a form of social control in environments in which agents are likely to interact with others whose intentions are not known. It allows agents in that system to reason about the reliability of others. The trust approach has raised the need for trust management (Blaze, et al., 1996) when it is being used to formulate security architectures. Thus, trust management has been the challenge to many researchers in research community. Aspects of trust management problem include formulating security policies and security credentials, determining whether

particular sets of credentials satisfy the relevant policies, and deferring trust to third parties. As a result, quite a number of security schemes have been proposed around this paradigm ((Huynh, et al., 2004), (Blaze, et al., 1996), (Alfalayleh and Brankovic, 1998)). A number of definitions for trust management have been suggested by different researchers in diverse contexts ((Bonatti, et al., 2005), (Huynh, et al., 2004)). Bonatti et als (2005) defined trust management as the activity of collecting, encoding, analysing and presenting evidence relating to competence, honesty, security or dependability with the purpose of making assessments and decisions regarding trust relationships. Trust management, regardless of its security challenges, has been mostly used in open multi-agent systems to develop effective security schemes. Multi-agent systems are composed of autonomous agents that interact with another using a particular mechanisms and protocols. These autonomous agents interact to achieve their design goals in uncertain and dynamic environments. Interaction pattern in multi-agent system has an impact in formatting the security architecture. Thus, the way agents interact determines the type of security models that is required to protect them.

In Context-Aware Component Interfacing Pattern (CACIP) architecture (Zuma and Adigun, 2006) allowed different components to indirectly interact in order to share services. Furthermore they defined the information bus that is used by participating components to communicate. Information bus carries the messages (both requests and published messages) that have been attached by components. Service provider components attach messages to the information bus in order to publish services they offer and can detach their messages at any time. On the other hand, service consumer components attach request messages to the information bus in order to request for

published services. If the requested message is not available at the time of request, a component can leave the message in the information bus so that it can be notified once that service is available. The interaction style defined in CACIP shows some characteristics of multi-agent system by allowing components to share services without intervention of a user. Thus, in this research, CACIP is modeled as the multi-agent system. Agents communicate using the CACIP style of communication.

However, CACIP architecture opens some security challenges that need to be addressed in order for agents to freely share services. Security challenges that are raised by CACIP interaction style include masquerading, unauthorised access, annoyance attack, eavesdropping, denial of services and alteration (Alfalayleh and Brankovic, 1998). As it is briefly discussed above that participating provider components in CACIP put their messages in the information bus to be accessed by other components, therefore, access control is needed that would make sure that only trusted agents are permitted to access the information bus. The access control would prevent malicious agents from interrupting other agents' messages and make them useless. This is because it is possible that a harmful agent accesses the information bus and starts to alter other agents' messages. This suggests that an appropriate trust management is required to secure agents communicating in CACIP based service oriented environment. Therefore, in this research trust management model is formulated to secure agents communicating in CACIP based systems.

2.3 Trust Evaluation Mechanisms

This section analyses existing research works in the literature based on the following approaches to develop trust-based security schemes:

i) Reputation-based – utilization of previous behaviour information of an agent to evaluate its trustworthiness and

ii) Policy-based – declared policies are used to specify access control conditions that yield the requested resource either granted or denied.

2.3.1 Reputation-based Security Schemes

A reputation-based approach has been used in the context of online electronic commerce systems, e.g. eBay. Reputation mechanisms are built using two different approaches: centralized and decentralised approaches. Therefore, researchers formulate reputation mechanisms using either of the two approaches.

2.3.1.1 Centralised Reputation Mechanisms

E-Bay Reputation Trust Model

Since traditional security mechanisms cannot protect an agent from unreliable service providers, novel models have been developed to model service provision trust, i.e. trust that a service provider is competent and will provide a service in a reliable manner (Grandison, and Sloman, 2000). The main building block of these models is information

about an agent's past behaviours. This information is used to reduce the trustworthiness of that agent in terms of its competency and reliability. Online reputation mechanisms (e.g. eBay (eBay Site), (Resnick and Zeckhauser, 2002) and Amazon Auctions (Amazon Site) are probably the most widely used of such models. They are implemented as centralised rating system so that their users can report about the behaviour of one another in past transactions via rating and leaving textual comments. In so doing, users in their communities can learn about the past behaviour of a given user to decide whether it is trustworthy to do business with. For example, an eBay user, after an interaction, can rate its partner on the scale of −1, 0, or +1, which means positive, neutral and negative rating respectively. The ratings are stored centrally and the reputation value is computed as the sum of those ratings over six months. Thus, reputation in these models is a global single value representing a user's overall trustworthiness.

However, this is too simple for applications in multi-agent systems since they only consider the trustworthiness of an agent as one dimension. Since the ratings are aggregated equally, the mechanism cannot adapt well to changes in a user's performance (e.g. a user may cheat in a few interactions after obtaining a high reputation value, but still retains a positive reputation).

In summary, the reputation values in these systems contain very little information, and users of these systems need to look for textual comments providing more information. Therefore, such mechanisms are not well suited to computational agents, which must usually make decisions autonomously. In addition, since there is no central authority that can control all the agents in open multi-agent system, an agent may well question the credibility of those centralised reputation models and decide not to use them.

SPORAS Reputation Model

Zacharia and Maes (2000) have proposed SPORAS which extend the online reputation models mentioned above by introducing a new method for rating aggregation. Specifically, instead of storing all the ratings, each time a rating is received it updates the reputation of the involved party using an algorithm that satisfies the following principles:

1. New users start with a minimum reputation value and they build up reputation during their activity on the system;
2. The reputation value of a user never falls below the reputation of a new user;
3. After each transaction, the reputation values of the involved users are updated according to the feedback provided by other parties, which reflect their trustworthiness in the latest transaction;
4. Users with very high reputation values experience much smaller rating changes after each update and
5. Ratings must be discounted over time so that the most recent ratings have more weight in the evaluation of a user's reputation.

In general, SPORAS is a centralized reputation model with more sophisticated characteristics to model the trust dynamics than other simple models. For example, Principles 1 and 2 above are to prevent a user with a bad reputation leaving the community and entering with a fresh reputation (since the reputation of a new user is the lowest reputation possible). However, at the same time, this penalises newcomers and may discourage them from participating in the community. In addition, SPORAS also introduces a reliability measure based on the deviation of rating values. This is an

indication of the predictive power of SPORAS for that user's reputation. For instance, a high deviation value can mean either that the user has not been active enough to be able to generate a more accurate prediction for his/her reputation, or that the user's behaviour has a high degree of variation. Hence, each user has a reputation value and a reliability value globally available to other users. In SPORAS, the reputation value of a user and its reliability are discounted over time as a new rating is received. Therefore, SPORAS can adapt to changes in a user's behaviour according to the latest rating.

In summary, SPORAS provides a trust measure that has more desirable features than that of similar online models such as eBay's, or Amazon's. However, its centralized design is not suitable for applications in open multi-agent system, e.g. CACIP. Moreover, SPORAS is very susceptible to rating noise resulted from agents' subjective views that are commonplace in open multi-agent systems.

2.3.1.2 Decentralised Reputation Mechanisms

Incentives-oriented Reputation Mechanism

Jurca and Faltings (2003a) introduce a reputation mechanism where agents are given incentives to report truthfully about their interactions' results. They define a set of broker agents (called R-agents) whose tasks are buying and aggregating reports from other agents and selling back reputation information to them when they need it. All reports about an agent are simply aggregated using the averaging method to produce the reputation value for that agent. Although the R-agents are distributed in the system, each of them collects and aggregates reputation reports centrally. Hence this approach still

possesses the inherent shortcoming of centralised models above (i.e. the questionable objectiveness of R-agents in open multi-agent systems). In order to incentivise agents to share their reports truthfully, Jurca and Faltings propose a payment scheme for reputation reports. This scheme guarantees that agents who report incorrectly will gradually lose money (during the process of selling reports and buying reputation information), while honest agents will not. Therefore, this mechanism makes it rational for an agent to report its observations honestly and this is the main contribution of their work. However, reputation reports are limited to the values 0 and 1 (0 for cheating agents and 1 for cooperating agents in an iterated Prisoner's Dilemma environment (Conte and Paolucci, 2002), and the rational property may not hold if an application requires reports represented by more than these particular values (e.g. 0.1, 0.75).

Regret Reputation Model

Another trust model that was reviewed is Regret. Regret (Sabater and Sierra, 2001), (Sabater, 2003)) is a reputation model in which the trust evaluation process is completely decentralized. Employing Regret, each agent is able to evaluate the reputation of others by itself. In order to do so, each agent rates its partner's performance after every interaction and records its ratings in a local database. The relevant ratings will be queried from this database when trust evaluation is needed. The trust value derived from those ratings is termed *direct trust* and is calculated as the weighed means of all ratings. Each rating is weighed according to its recency. Intuitively, a more recent rating is deemed to be more current and is weighted more than those that are less recent. However, the method Regret uses to calculate the weights for each rating has a shortcoming regarding

time granularity control and does not actually reflect a rating's recency. Like SPORAS, Regret also provides a reliability value for each trust value to represent its predictive power. The reliability value is calculated from two reliability measures: the number of ratings taken into account in producing the trust values and the deviation of these ratings. In addition, agents are assumed to be willing to share their opinions about one another. Based on this, Regret develops a witness reputation component along with a sophisticated method for aggregating witness reports taking into account the possibility of dishonest reports. The operation of this component depends on the social network built up by each agent. In particular, Regret uses the social network to find witnesses, to decide which witnesses will be consulted, and how to weight those witnesses' opinions. However, Regret does not specify how such social networks are to be built, and this means that component is of limited use.

Besides direct trust and witness reputation, Regret also introduces the concepts of neighbourhood reputation and system reputation. The former is calculated from the reputation of the target's neighbour agents based on fuzzy rules. However, this again requires a social network to work. The system reputation is a mechanism to assign default trust values to the target agent based on its social role in an interaction (e.g. buyer, seller). However, this is only useful if additional domain-specific information is available.

In summary, the decentralising approach of Regret allows agents to evaluate trust by themselves without relying on a centralised mechanism. It also takes various sources of trust information into account and considers the possibility of disinformation. Therefore, the approach Regret adopts is compatible with the requirements for a trust model in open multi-agent system. However, apart from the direct trust component, the rest of the model

is not readily applicable. The main reason is that Regret does not show how each agent can build the social network on which Regret heavily depends.

Referral-oriented System

In building a reputation system based on witness information, Yu and Singh (2002), (Yu and Singh (2003) develop a mechanism to locate information sources (i.e. witnesses) based on individual agents' knowledge and help (through each agent's contacts) without relying on a centralised service. Hence, this approach is well suited for applications in open multi-agent system which is distributed by nature. In particular, in this system, agents cooperate by giving, pursuing, and evaluating referrals (a recommendation to contact another agent). Each agent in the system maintains a list of acquaintances (other agents that it knows) and their expertise. Thus, when looking for some information, an agent can send the query to a number of its acquaintances who will try to answer the query if possible or, if they cannot, they will send back referrals pointing to other agents that they believe are likely to have the desired information (based on those agents' expertise). Yu and Singh (2002) referral system uses a vector space model (VSM) (Salton and McGill, 1983) to model agents' expertise. An agent's expertise is then used to determine how likely it is to have interaction with or to know witnesses of the target agent. This mechanism does not define how to make sure that referenced agents are trusted to provide true behaviour information about the requester agent. And the mechanism is not efficient because it may take much time to get responses from the agents that were referenced to provide history record, consequently a system take too long to execute evaluate one agent.

TRAVOS Reputation Model

Teacy, et al (2005) proposed Trust and Reputation model for Agent-based Virtual OrganisationS (TRAVOS), a trust model that is built upon probability theory and based on observations of past interaction between agents. In this model, the outcome of an interaction is simplified into a binary rating (i.e. 1 for a successful interaction, 0 for an unsuccessful one). Using binary ratings allows TRAVOS to make use of the beta family of *probability density functions* (PDF) (DeGroot and Schervish, 2002) to model the probability of having a successful inter action with a particular given agent. This probability is then used as that agent's trust value. In addition, using PDFs, TRAVOS also calculates the confidence of its trust values given an acceptable level of error. If the confidence level of a trust value is below a predetermined minimum level, TRAVOS will seek witness information about the target agent's past performance. Witness information is shared in the form of frequencies of successful and unsuccessful interactions that the witness has had with the target agent. After interacting with the target agent itself, the evaluator compares the received witness report with its own observations. By this means, the evaluator calculates the probability that the witness's information supports the true behaviour of the target agent within a reasonable margin of error, and uses this probability to weight the impact of the witness' opinions on future decisions made be the evaluator. However, TRAVOS's simplified representation of interaction ratings is rather limited and not suitable for a wide range of applications in open multi-agent system.

2.3.2 Policy-based Security Schemes

The policy-based approach has been proposed in the context of open and distributed services architectures as a solution to the problem of authorization and access control in open systems. In this approach, trust management mechanisms utilize different languages and engines for specifying and reasoning on rules for trust establishment. The goal is to determine whether or not an unknown user can be trusted, based on set of credentials and a set of policies. Policy-based trust relies on objective "strong security" mechanisms such as signed certificates and trusted certification authorities (CA) in order to regulate the access of users to services. Moreover, the access decision is usually based on mechanisms with well defined semantics (e.g. logic programming) providing strong verification and analysis support. The result of such a policy-based trust management approach usually consists of a binary decision according to which the requester is trusted or not, and thus the service (or resource) is allowed or denied.

2.3.2.1 REWERSE

Employing policies to protect services is a crucial aspect in security systems. Policies benefit users to get full access to available services and also give necessary access control to services so that only those that meet minimum access requirements can access resources. The important thing about policies is that users must understand them in order to enjoy benefits offered by a particular system. Policies play crucial roles in enhancing security, privacy, and service usability. Policies are typically divided into different kinds

among which include: access control policies, privacy policies, and business rules. The work in (Bonatti, et al., 2005) proposed the REasoning on the WEB with Rules and SEmantics (REWERSE) which integrates these kinds of policies to a single framework. Bonatti and others argue that integrating these kinds of policies to a coherent framework can be effective in a way that (i) a common infrastructure can be used to support interoperability and decision making, and (ii) the policies themselves can be harmonized and synchronized. However, REWERSE the use of all these kinds of policies simultaneously may be time consuming when decision is taken. This approach influences our research with its provisional policies that it introduces. Provisional policies are used in REWERSE if the system has to specify certain credentials that it needs in order to authenticate the component.

2.3.3 Agent Security Systems

The review of the two architectures that impact our model is presented in the following subsections. The strengths of these two architectures are combines to form our trust model.

Maarof and Krishna

Maarof and Krishna (2003) proposed the trust management model for multi-agent system trading society. It is argued in this model that, there is a need of a mechanism that gives option to have different combinations of trust for different situation in multi-agent trust management. The model has three components: objective trust-based agents, reputation

and the trust mechanism. These components are combined together to form the trust opinion needed in decision making. Each of these components has the role to play at different times in the overall life of a trading partnership. The reputation component is used because of the fact that it can be beyond each individual's resources to evaluate all aspects of a given situation when making a trust decision. However, the model does not clearly define how recommenders are selected to provide required reputation.

Ping and Others

The system agent architecture proposed in (Ping, et al., 2004) typically designed to run in ad hoc networks. The architecture is based on the framework of the immune system that is capable of detecting and identifying an attack, elaborating a specialised response measure to isolate the invader, and recover from the attack. Three important agents are defined in this architecture: monitor agent, decision agent and killer agent. The interesting part of this work is the delegation of these agents to perform different tasks. Monitor agent monitors incoming agents that wants to participate in the network and decision agent makes decision based on the information provided by monitor agent about agents coming to join the network. Killer isolates invaders from the network. Although this architecture is designed for ad hoc network but its approach of detecting agents and decision making can be adopted to design security scheme for open multi-agent system running in wireless networks. However, neither reputation-based approach nor policy-based approach is used in this scheme to make decision.

2.4 Trust Management Design Challenges

Managing trust in multi-agent system is modeled according to specifications and security design principles. Trust-based security schemes analysed in section in 2.4 and 2.5 experience the following design challenges: i) Fairness, ii) End-to-end security, iii) Robustness, iv) Scalability, and v) Visibility. Next section discusses each of these design challenges.

2.4.1 Fairness

Decision making based on trust needs to be fair for every participants in a multi-agent system. In general, there is no entity in a multi-agent system that can have a global view of all agents that communicate in it. This suggests that, at times unfairness may occur when trust is used to make decision about who is suppose to access resources. However, a mechanism must be defined to make sure that all participants are treated fairly.

2.4.2 End-to-end Security

An agent that makes request must be checked for security from the point of origin to the destination before given an access to the system. The fact that the component has been authenticated in a particular point does not necessary mean it has access to all parts of the system, thus in each point the component must meet the requirements declared. If this is not taken care of, cheating of component may increase.

2.4.3 Robustness

Some components may present invalid information in trying to falsely get access to the system. The trust model should be able to resist such situation and be able to correctly operate under such situation. If the system is not robust, it is subject to failure when quite a number of malicious components attempt to get access into the system

2.4.4 Scalability

The increase in number of components that make request can be a challenge to a security scheme if scalability problem was not considered during designation. The trust model has to evaluate a number of components requesting for services. The security mechanism that is not scalable may cause the system to be inefficient in processing service requests

2.4.5 Visibility

Participating components should be able to know and understand security implication of their actions. Each component must know its state so that it would decide how it should behave in future actions in the system.

2.5 The Proposed Trust Model

The previous sections reviewed the trust models that have been proposed to manage trust in diverse context of security requirements. However, these security schemes do not provide a global solution for trust management in multi-agent system. The reviewed schemes experience some challenges stated in section 2.6.

In this research, we propose an agent-to-agent trust management model. The proposed trust model is constructed to provide security for agents communicating in CACIP based system. Our model combines reputation-based with policy-based approaches to manage trust among agents that share services in CACIP based system. Agents that make requests are authenticated based on their reputation and authorized based on declared policies to access services in CACIP system. Behaviour information for each agent that has accessed the system is stored in the local repository and is used to evaluate that agent when it makes request nest time. It is possible for an agent to not have any historical record in the local database. In that case, only foreign reputation is used to evaluate trustworthiness of the agent. Foreign reputation is queried when the agent make request for the first time or if the local reputation is not sufficient for trust evaluation.

The proposed model was constructed by integrating two security models defined in (Maarof and Krishna, 2003) and (Ping, et al., 2004). The strengths of these two schemes are used to construct our trust model. Table 2.1 shows the brief descriptions of the two schemes which were used to construct the trust model that is also describe in the same table.

Table 2. 1: Description of Existing Schemes and the Proposed Trust Model

Model/ Architecture	Context	Problem to be solved	Solution Mechanism
Hybrid Trust Management Model	The previous interaction experience with a requesting agent is sometimes not sufficient to make a fair decision about that agent when it wants to join the network. Therefore, reputation from 3rd party recommenders can be used to get the previous behaviour information about an agent that request a service	"Mobile agents can be unfairly isolated from a network because a system does not have trust mechanism that gives an option to have different combination of trust from different situations and sources."	"Combining components of a trust based trading relationship, reputation, subjective trust and objective trust to ensure that the information to make a decision about trustworthiness of an agent is sufficient. Defining reputation component that is only enabled when trust ratings based from experience is not sufficient, and bring the flexibility to permit the reputation component to be enabled or disabled."
System Agent architecture	Security architecture that is based on the framework of the immune system, that is capable of detecting and identifying an attack, elaborating a specialized response measure to isolate the invader, and recover from attack. It has the same learning and adaptive capability of the human immune system, and so it is able to react to unknown attacks and to improve to response under subsequent exposures to the same attack	"An agent that wants to participate in ad hoc network may be an invader, i.e. it would harm other agents in the network. Therefore, identifying such agent in open distributed systems can be a challenge."	"Develop an immune-based security architecture for mobile ad hoc networks that has the mechanism to detect and isolate the invader agent. Behaviour of an agent participating on the network is monitored by the monitor agent which resides on each node"
(Proposed) Agent-based Trust Management Model	Mobile nodes are interacting in a mobile network; trustworthiness of each node is evaluated based on trust. Mobile agent technology is adopted to establish trust management among agents that are communicating in a CACIP based service oriented mobile environment. Trust is built based on reputation of each agent and access to services is given to agents that meet access requirements as stated policy in the information bus	Mobile agents communicating in a CACIP based service environment are open to some security challenges. Some of security challenges are unauthorized access, eavesdropping, annoyance attack, masquerading, alteration and denial of service. A security mechanism is required to ensure protection of agents when they share services.	Evaluating trust based on reputation (local or foreign) and policies, i.e. each agent is authenticated according to its previous behaviour information and authorized based on access policies that are set in the information bus.

CHAPTER THREE

3. Methodology and Model Development

3.1 Introduction

The previous chapter presents the analysis of existing trust-based security schemes that have been proposed for multi-agent systems. However, these proposed security schemes have their shortcomings without support for complete fair mechanisms for agents' trust evaluation.

We propose agent-to-agent trust management model to address some of these limitations: fairness, end-to-end security, and scalability. Our formulated trust model intends to offer end-to-end security examination of agents making requests. It is crucial in trust models that all agents are fairly treated in open multi-agent system. Therefore, the formulated trust model would ensure fairness in trustworthiness evaluation of agents.

This chapter presents development of agent-to-agent trust management model and the detailed components that constitute this model. Section 3.2 discusses the design principles; agent-based trust management model is described in section 3.3. Section 3.4 discusses the communication framework for our formulated model followed by authentication and authorization frameworks discussed in section 3.5. Section 3.6 describes the reputation mechanism used to evaluate trustworthiness of agents.

3.2 Trust Management Design Principles

The existing trust models have attempted to solve the problem of trust management in open multi-agent systems. However, there has been no global solution to the problem; instead some of the proposed schemes, discussed in section 2.4 and 2.5 of chapter two, provide solutions with some limitations. The limitations recognized from the reviewed schemes make them unfit to secure the CACIP. We, therefore, propose own trust-based security scheme to secure CACIP architecture. The design principles for our scheme are categorized as follows:

i) End-to-end security;

ii) Fair trustworthiness evaluation;

iii) Robustness and scalability support and

iv) Provision of system visibility.

Each of these design principles is briefly described below.

3.2.1 End-to-end Security

A security scheme should handle end-to-end security checking of components that wants to access resources. In each point leading to a particular resource, an agent must be checked for security in order to make sure that only risk-free agents get an access. In a multi-agent system, end-to-end security examination can be achieved by assigning some agents or components to perform security checking at each point on the itinerary of the agent until it reaches the resource. This suggests that different mechanisms may be used at each point and different access requirements set for each requester agent has to meet in

order to be granted the access to resources. The requester agent should present sufficient requested requirements in each point to pass to another point.

3.2.2 Fair Trustworthiness Evaluation

Trust decision sometimes can be dependent on different entities of a security system. This poses a challenge of making sure that agents are given same opportunities during trust decision. We define fairness as the ability to make judgment free from discrimination or dishonesty. Decision making in multi-agent system must be fair to all agents that make request. To achieve this, various sources of behavior information must be used to make decision in the case whereby behaviour information is not found from the local repository.

3.2.3 Robustness and Scalability

An increase in number of agents making requests may cause the system to malfunction if it is not robust and scalable. In trust model, neighbouring agents may provide invalid information about the agent being evaluated for trustworthiness. Thus, the system must be able to accurately and constantly deal with erroneous data sent to it. A multi-agent system should be able to correctly and continuously evaluate all agents making requests in spite of how huge the number of requests is. Thus, the system is scalable if it does not get confused with executing the huge number of requests.

3.2.4 System Visibility

In a multi-agent agent system that uses trust-based security mechanism, every action of the agent would impact its future trust evaluation. Therefore, a security mechanism should enable participating agents to know and understand security implications of their actions.

3.3 Agent-based Trust Management Model

The proposed trust model aims at providing a security mechanism for multi-agent system that ensures fairness in evaluating agents making request for resources. The model was constructed by integrating the strengths of two schemes: *An hybrid trust management* and *system agent architecture*, both discussed in section 2.6 of the previous chapter.

The *hybrid trust management model* (Maarof and Krishna, 2003) defines trust mechanism that is used to make decision about an agent that requests for a service. This model reveals that, it can be beyond individual's resources to evaluate all aspects of a given situation when making trust decision; hence, agents must rely on other sources of information to get the reputation of an entity. Reputation is provided by 3rd party recommenders in the network, and recommenders' component of the model is enabled if the direct previous interaction behaviour information with the agent is not sufficient to make a decision. However, this model does not define a mechanism to deal with malicious agents once they have been identified based on their reputation. This limitation is addressed in (Ping, et al., 2004). Ping et al (2003) proposed system agent architecture that defines how malicious agents are detected and identified. The Monitor agent resides in each node and monitors behaviour of the neighbouring node.

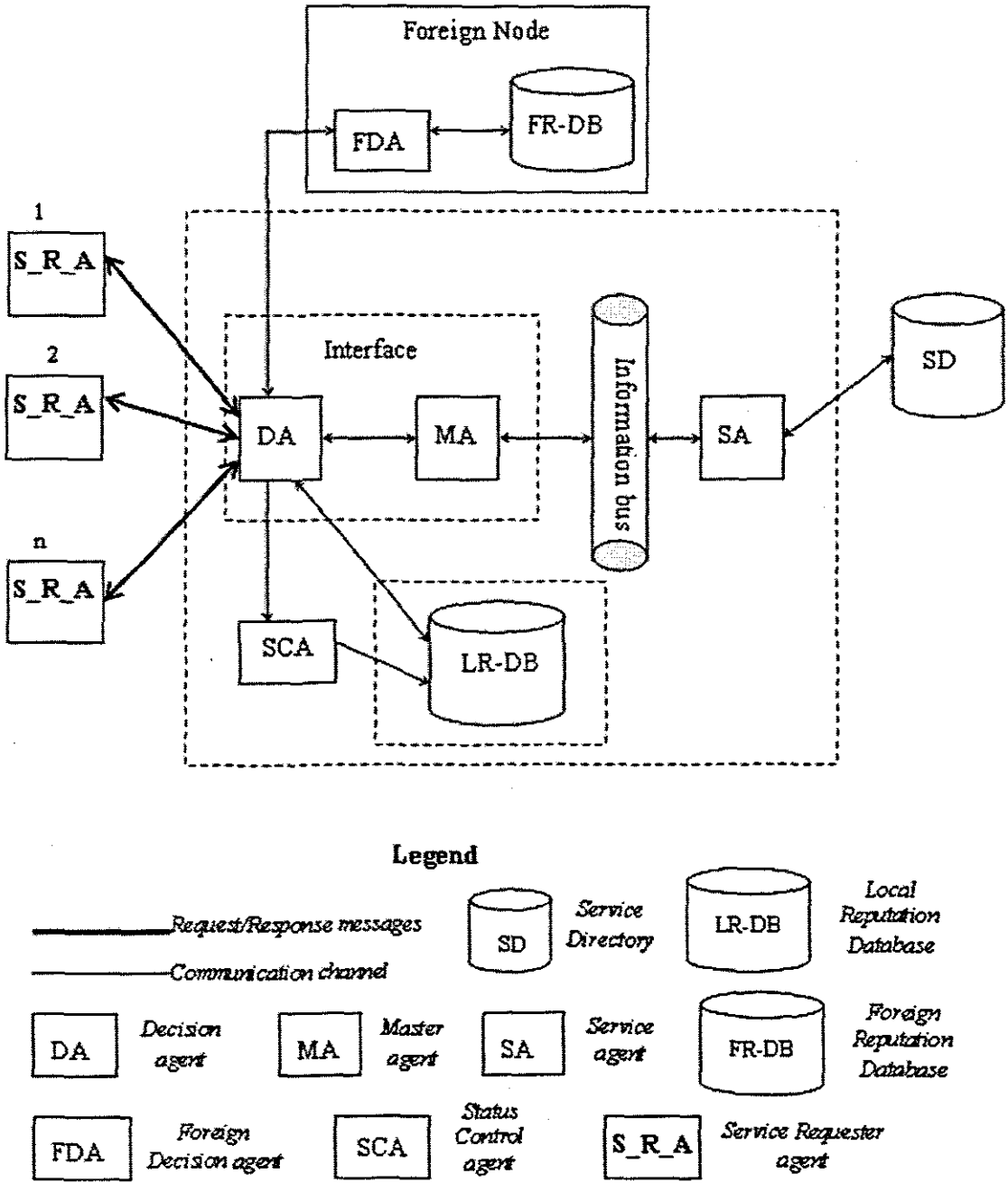

Figure 3. 1: Agent-based Trust Management Model

It then sends the behaviour information to the decision agent that makes decision whether to give access or isolate the requester agent from the network. However, this architecture does not ensure the end-to-end security as behaviour information is transferred from

monitor agent to decision agent.

The core of the developed agent-based trust management model, shown in figure 3.1, is in the delegation of two agents: *Decision Agent* (DA) and *Master Agent* (MA), which are at the interface of the information bus to authenticate and authorize every agent that makes a request. This means every agent that wants to access services in the information bus has to first interact with DA for evaluation of trustworthiness. S_R_A sends a request message to a CACIP based system and DA receives the request. In order for DA to make a decision about S_R_A, it checks from local reputation if there is a previous interaction record (behaviour information) of the S_R_A. If previous interaction record of the agent is not found from local reputation or is insufficient to make decision, DA broadcasts a message, requesting for behaviour information, to foreign reputation (third party recommenders). Neighbouring agents that has received the reputation request message reply to DA with a reputation value of the S_R_A.

Local-reputation is formed from previous direct interaction experiences of the Decision Agent with agents that has requested for services. On the other hand, *foreign-reputation* refers to reputation obtained from other neighbouring nodes (recommenders), in the network, which has the previous interaction records of the agent that request for service.

When the S_R_A has been authenticated by DA, it then interacts with the Master Agent which holds access requirements (policies) that have to be met by the requesting agent in order to get access to resources. Next, the description of the interaction of agents communicating in CACIP based manner is presented in the following section.

3.4 Communication Framework for the Agent-based Trust Management Model

Interaction style plays an important role in choosing an appropriate security mechanism to secure mobile agents. This is because in an open distributed system, agents are able to interact with one another to share services without intervention of their senders. CACIP architecture proposed by (Zuma and Adigun, 2006) provides the indirect interaction of components (refer to figure 1.1). This form of interaction style brings some security weaknesses that open room for malicious agents to access resources in multi-agent systems. Therefore, an access control is required to cover this gap. The agents at the interface of information bus in our developed trust management model control access to services and by so doing they facilitate and monitor communication of participating agents. Hence, Decision Agent performs authentication and Master Agent authorizes the authenticated agents to access services. The Master Agent can perform its duty after decision agent has authenticated the agent, i.e. Master Agent is dependent on Decision Agent results to execute its task. Trustworthiness of agents is evaluated based on local or foreign reputations when authentication process is performed. And some policies (access requirements) are set in the Master Agent for authorization process. In case a requester agent is found to be an invader, Decision Agent instructs the Status Control agent (SCA) to set status (either suspended or permanently removed) in the database (LR-DB). To clearly describe communication of agents, the components participating in the process are individually defined in details in the next sections.

3.4.1 Decision Agent (DA)

DA performs authentication of agents that make requests. It uses the communication component to interact with other entities in the develop trust model.

The communication process entails the following sub-tasks:

- Receive requests from service requester agents (S_R_A);
- Send responses to service requester agents;
- Broadcast reputation queries to 3rd party recommenders for behaviour information (*foreign reputation*) of the requester agent and receive the computed trust value from FDA when it responds and
- Establish connection between DA and MA to transfer security control.

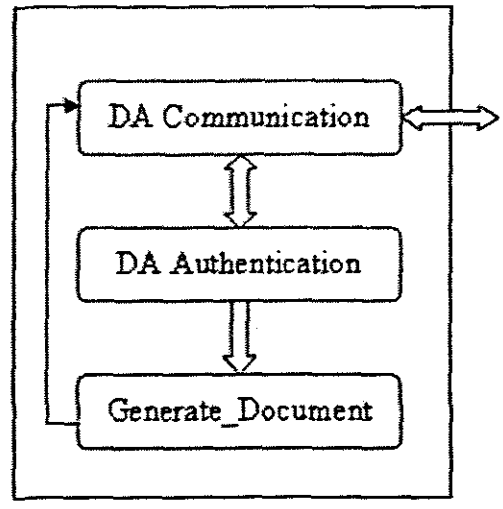

Figure 3. 2: Components of Decision Agent

In order for the DA, shown in figure 3.2, to analyse and assess each service request it receives, *authentication* component plays the role of security coordinator by first watching out for requests from *communication* component. Then it instructs the

communication component to broadcast reputation queries if it is necessary. The same *authentication* component is charged with deciding what to do with a request. Only when a requester agent is trustworthy that *authentication* component instructs *Generate_Document* component to produce the service document (service_doc) to be presented to the Master agent.

The *Generate_Document* component constructs a service_doc and sends it to the *Communication* component. The generated service_doc is then supplied by the DA before it is sent to the Master agent.

3.4.2 Status Control Agent

Status Control Agent (SCA) administers the records in the LR-DB. The status of a requester agent can either be *active, suspended,* or *permanently removed* from the database.

Active state – means the agent can be possibly trusted based on its records.

Suspended status – means the agent is in the state whereby it would be considered later because of wrong actions it had performed.

Permanently removed status – means the agent will never be given access because it has history of bad behaviour.

3.4.3 Master Agent (MA)

The communication component in Master agent, shown in figure 3.3, receives the service_doc that was generated by Decision agent. It then sends it to Authorization

component for authorization process.

The Authorization process entails the following sub-tasks:

- Verify the digital signature on the service_doc;
- Reads the service description on the service_doc and
- Check the access requirements from the Access Policy component when each service is requested.

The Access Policy component carries access requirements that each requester agent has to meet in order to get access to the information bus. To give access to services that are in the information bus, Master Agent authorizes agents after they have been authenticated by the Decision Agent.

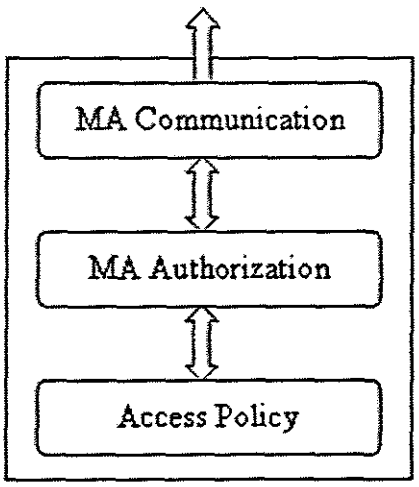

Figure 3. 3: Components of Master Agent

Master Agent holds set of rules (access requirements), for each service in the information bus, that are being used in authorizing agents. Master Agent knows the encrypted digital signature of the Decision agent on the service_doc it is receiving. It then verifies that signature to determine that a requester agent has been authenticated by the assigned Decision Agent. Master agent uses the shared secret key (Kritzinger, et al, 2003) to

decrypt the signature to confirm that the requester agent has been authenticated.

3.4.4 Foreign Decision Agent (FDA)

When DA broadcast reputation query, it communicate with foreign decision (FDA) in foreign CACIP oriented nodes. The FDA performs the same tasks as the DA. It replies to the reputation query from DA by sending the computed trust value of the requester agent if it is available from its LR-DB.

3.4.5 Local Reputation Database (LR-DB)

During direct interaction of Decision Agent with requester agents in the network, behaviour information of each agent is stored to build history record (reputation). This reputation is called local reputation and is used to make decision during trust evaluation.

3.4.6 Foreign Node

Decision agent cannot depend only on local reputation to make fair decisions, but it needs other external but reliable sources (which in the context of this model are neighbouring or foreign nodes) to obtain previous behaviour information of agents that request for services. Reputation from those external sources is called foreign reputation.

3.5 Authentication and Authorization Frameworks for the agent-based Trust Management Model

When an agent requests for service from information bus, its motive and intention are not known until an evaluation is performed. In order to perform complete trust verification, each agent sends its request message coupled with credentials (Winsborough, et al, 2002) to decision agent. Decision Agent either verifies the message and decides whether to certify the agent and give the service_doc to be checked for authorization or denies the request based on its reputation. Authentication is done by decision agent and authorization is the task of Master agent. In the following sections, authentication and authorization frameworks are discussed.

3.5.1 Authentication Framework

In order to ensure safe communication among agents, all agents that want to participate in the network must be authenticated. Reputation from different sources is the major entity that is used during authentication if the local reputation is not sufficient or there is no record in the requester agent. The decision agent authenticates requester agents based on their behaviour information as discussed in section 3.4.1.

Service requester agent sends request message to *decision agent* to verify its identity and do some assessments. Decision agent first checks if previous behaviour information of that agent is available from its local reputation database. If information in local reputation is sufficient to make the decision, the decision would be made based on that information.

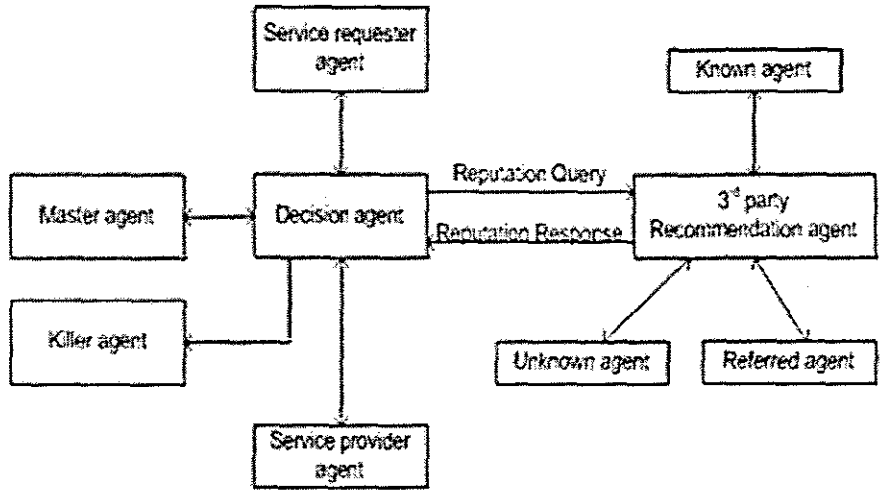

Figure 3. 4: Component Diagram of the Authentication Framework

But in a case where behaviour information in local reputation is insufficient to make decision, *decision agent* request for reputation from 3rd party recommenders. It does this by broadcasting the request to the neighbouring nodes. Those neighbours that have the requested behaviour information reply to the request. And when Decision agent has received the responses from recommenders, it then takes decision based on that recommendation combined with local reputation if available. Third party recommendation agents that provide behaviour information are not all known by the Decision agent; hence they are divided into three types. Figure 3.4 shows the framework to authenticate agents and its three main components, namely *Known Agents*, *Referred Agents* and *Unknown Agents*.

- The *Known Agents* are known by the service requester agent (including trusted and un-trusted agents). They are non-anonymous or pseudo-anonymous agents;
- The *Referred Agents* are not known to the Service Requester Agent itself but through references provided by known agents. They are non-anonymous agents and

- The *Unknown Agents* – agents with no referral or direct interaction with the Service Requester Agent. They are anonymous agents.

When it is found that the Service Requester Agent is an invader, the decision agent triggers the SCA to perform its task, as discussed in section 3.4.1. But when the Service Requester Agent is trusted, decision agent generates a service document (service_doc) and gives it to the Service Requester Agent. The service_doc contains the following information:

- The *Agent ID* is the unique identity number of the Service Requester agent;
- *Service Description* refers to the list of items that the Service Requester agent is asking the information bus to provide and
- The *Agent's signature* is the token appended to the service_doc for verification by Master Agent.

3.5.2 Authorization Framework

An agent that request for a service can be granted the access to available services based on some policies that are set in the Master agent. When the agent requesting a service has been authenticated by the Decision Agent, it obtains the service_doc which would be presented to Master Agent for authorization, as shown in figure 3.5. The Master Agent receives the service_doc from Service Requester Agent and verifies three things. Firstly, it decrypts the signature on the service_doc to confirm that it was really signed by the Decision agent.

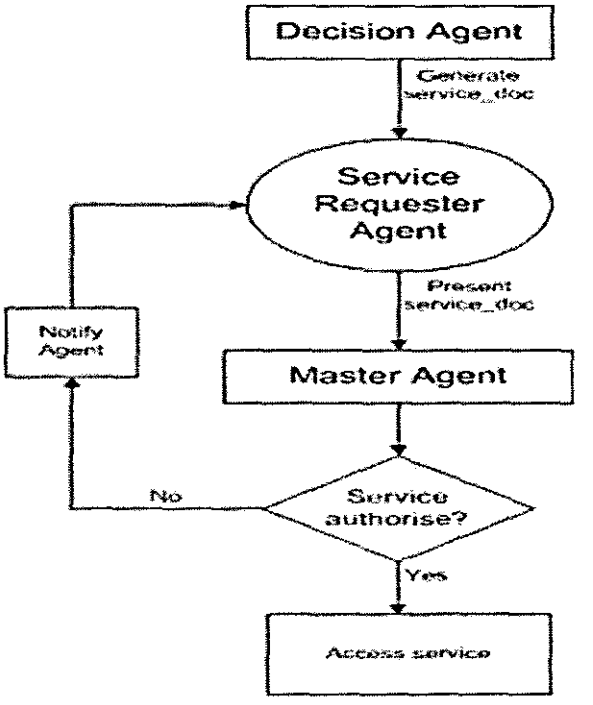

Figure 3. 5: Authorization Framework

The signature on the service_doc indicates that the agent has been authenticated. Secondly, it checks if the required service, described in service_doc, is available in the information bus.

Thirdly, it is possible that the required service is available in the information bus but the requester is not allowed to access it, possibly because it does not meet the access requirement stated in the stipulated policy. Therefore, when the service is available and the requester is authorized to get it, it is then given the access to that service.

On the other hand, if the required service is not available or the requester is not authorized to access any service in the bus, the Master Agent sends appropriate notification message to the Service Requester Agent.

Authentication Algorithm	Authorization Algorithm

if (local_reputation is sufficient)
 	
Use reputation
 	
Else
if (foreign_reputation is sufficient)
 	
Use reputation
 	
Else
Detach from network
 	
End if
End if
	if (service available)
	if (signature is correct)
	
	Permit entry
	
	Else
	Isolate on network
	
	End if
	
	Else
	Detach from network
	
	End if

Figure 3. 6: Control Structure for Trust Evaluation of Agents Requesting for a Service

Table 3.1 gives an overview of the control structure for identifying agents that wants to participate in a network, and the algorithm to allow or deny an agent from accessing services in the information bus. The identities for the Service Requester Agent are checked by Decision agent during authentication. During the authentication, the availability of the requested service is not confirmed until the requester presents the service_doc to Master Agent for authorization process.

Model entities and their relationships can be illustrated by using a class diagram. Figure 3.6 shows the class diagram of agent-based trust management model. The following is the description of each class:

a. **Service Requester Agent**. Any mobile agents that request for a service in a network are represented by this class;

b. **Agent**: This is the abstract class that defines shared attributes and methods;

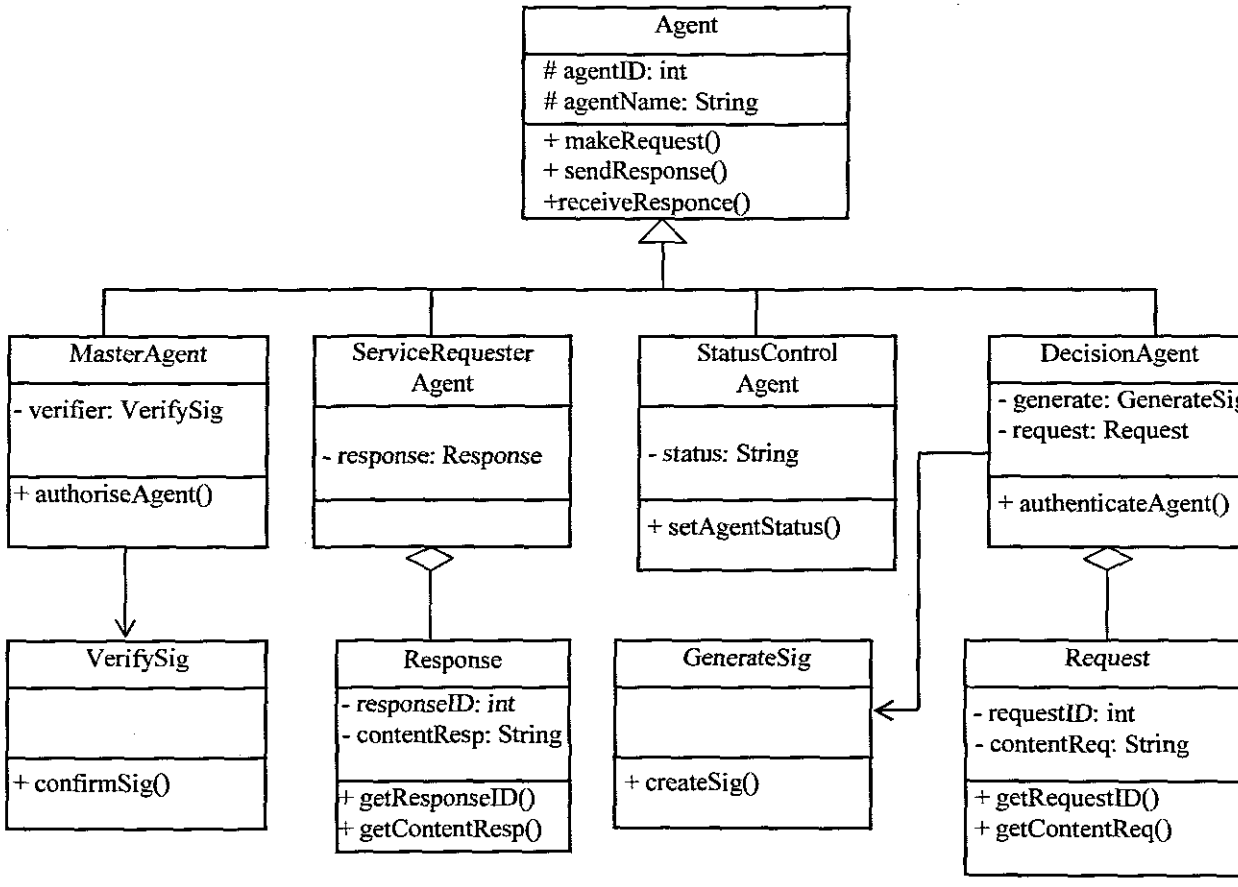

Figure 3. 7: Class Diagram for Agent-based Trust Management Model

c. **Decision Agent.** This is the class that represents an agent which determines whether a requester agent can participate or not in the network or it must be denied it request;

d. **Master Agent.** This class represents the agent that authorizes other agents to access services in the information bus;

e. **Status Control Agent**. This class represents the agent that sets status, based on computed reputation, of each agent that has interacted with DA.

f. **VerifySig**. This class verifies the information in the service-doc and

g. **GenerateSig**. This class is used to generate and sign service_doc

3.6 Reputation Computation

The reputation of an agent is computed based on the behaviour information from either local reputation (direct previous interaction) or foreign reputation (from 3rd party recommenders). Decision agent determines the trustworthiness of each requester agent based on its reputation. The equation below computes the reputation (which is the trust value) for an agent that request for a service.

$R_V = T_V + (G_A - B_A) / T_A * 100$ ..... (1)

The reputation value is calculated in percentage.

- R_V – a reputation value for each agent that determines its trustworthiness;
- G_A – number of good behaviour instances of an agent when it was accessing resources;
- B_A – number of instances of bad behaviour of an agent when it was accessing resources;
- T_A – total number of actions an agent has done and
- T_V – a reputation value from 3rd party recommenders.

The reputation policy adopted states that an agent that has the trust value of at least 50% is trusted. Therefore, an agent is deemed malicious if its trust score is below 50% and it is then denied an access. An agent can improve on its trust value by simply increasing the number of instances good actions. The computation is performed by the DA every time agent requests for a service. Initially, T_V is set to 0 for all agents requesting for services. It is dependent on using T_V parameter to make decision in our model, since it is possible for

the DA to have sufficient reputation from its local reputation database. For instance, let the requester agent A with G_A=70, B_A=15 then the total number of actions (T_A=85). In this case, the DA does not need to request for reputation from neighbouring nodes to authenticate agent A. Therefore, T_V remains zero for agent A when its R_V is computed. The R_V would be $(G_A - B_A)/T_A * 100$, (70 - 15)/85 * 100, so agent A scores 65% which means it is trusted.

Each DA maintains a local reputation repository in a tuple of the form:

[*Agent_id, total number of actions, number of bad actions, number of good actions*]

The reputation repository stores the updated behaviour information of all agents that has accessed the information bus. The information in the in repository is used by DA to take decision when agents make requests for services. Each history record in the repository has the indication of its status. The SCA controls the status of all agents that has interacted with the DA; therefore it determines whether the status of an agent should be *activated*, *suspended* or *permanently removed* from the repository based on the behaviour information.

CHAPTER FOUR

4. Design and Implementation

4.1 Introduction

The previous chapter presented the proposed trust management model with full details of how it intends to achieve the goal of this research. The model has shown that trust-based approach can be used to establish security in multi-agent application system when agents communicate in CACIP based service oriented environment. In the developed model, multi-agent system paradigm is applied as the foundation for securing services or messages that are shared by mobile nodes on mobile wireless network. Requester agents are judged by decision agent (DA) based on their trust scores after computation. Trust level of each requester agent is measured based on its reputation. An agent's trust score must reach a minimum trust level in order to be granted a full access to available services. Hence, this chapter presents the implementation of the proposed model, performance analysis and results of the research.

4.2 Development of Agent-to Agent Trust Management System

The implementation of the agent-to-agent trust management system intends to show how agents are used to secure information bus (from CACIP architecture). Agents in this multi-agent system play different roles.

4.2.1 Overview of the System

A multi-agent system (MAS) is a system composed of several agents, capable of mutual interaction. The interaction can be in the form of message passing or producing changes in their common environment. The agents can be autonomous entities, such as software agents or robots. Our developed multi-agent system is comprised by software mobile and static agents. Requester agents are mobile agents (they move from one node to the other to get the required service). Before getting an access to resource, they have to meet the stipulated access requirement. Therefore, authentication and authorization process must be performed in order to ensure that only trusted agents can access resources.

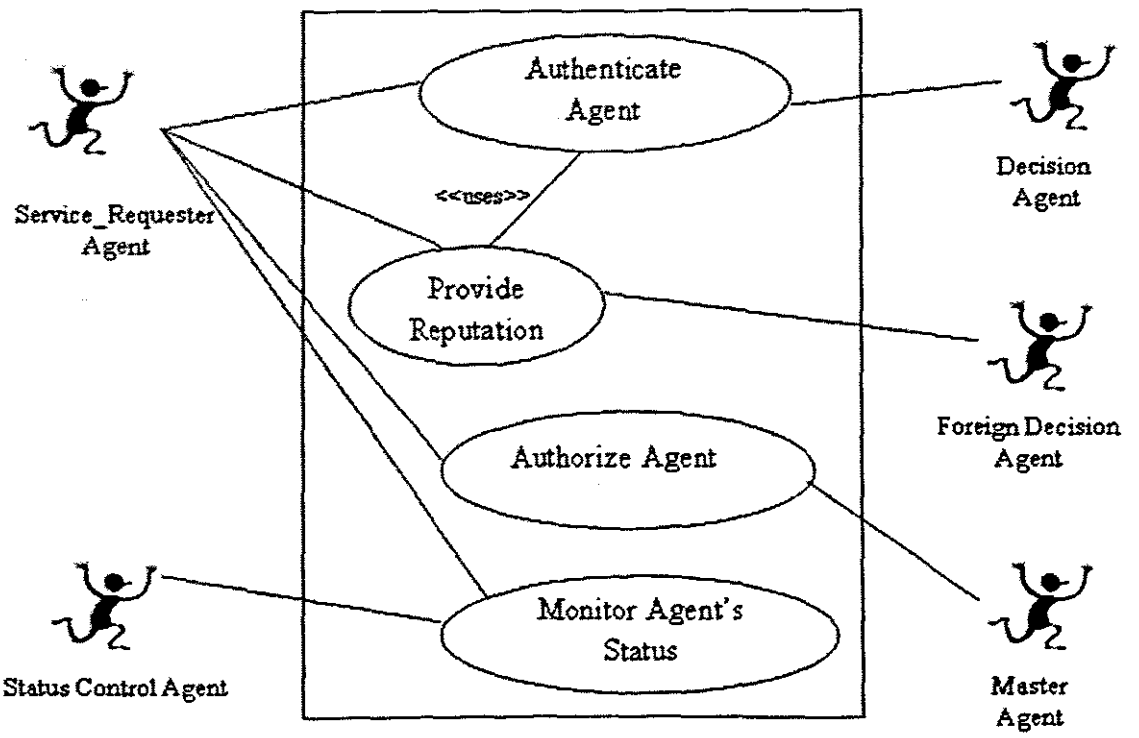

Figure 4. 1: A Use Case diagram for agent-to-agent trust management

Authentication and authorization processes are separate tasks performed in different

points of security. They are carried out by static agents (Decision agent (DA), Master agent (MA), and Status Control agent (SCA)) residing in each node that has CACIP. Figure 4.1 presents the use case diagram that depicts functionalities and actors of our system.

4.2.2 Processes for each request

Requester agents (S_R_A) send request messages to nodes that have CACIP; each message sent by requester agent has to be evaluated in order to ensure trustworthiness of that agent. Authentication occurs between two interacting agents: S_R_A and MA.

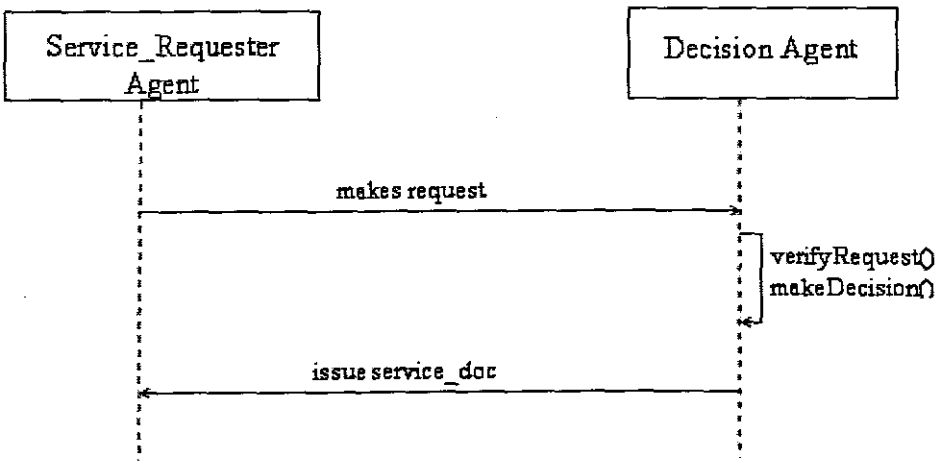

Figure 4. 2: A sequence diagram for authentication

The requester agent interacts with the Decision agent that deals with authentication of all agents that make request. Upon receipt of request, Decision agent performs trust evaluation based on previous behaviour information in the LR-DB. The reputation value is computed for each agent that is making a request. The computed reputation value

determines the trust level of the requester agent. In figure 4.2, it is illustrated the authentication process for each agent that requests for service from the system.

Authorization of agents is dependent on the results of authentication process. When an agent has been authenticated, it is being transferred to MA for authorization. Figure 4.3 depicts authorization process that takes place after authentication has been completed for each requesting mobile agent (refers to section 3.4.2).

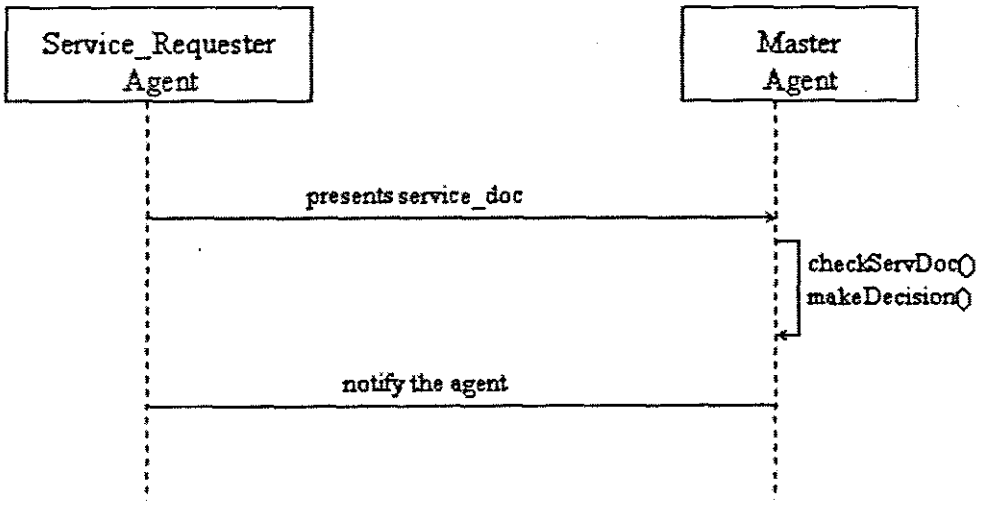

Figure 4. 3: A Sequence Diagram for Authorization process

A fair reputation-based trust decision about an agent sometimes demands that a decider must ask for help form other entities that might know better about the requester; otherwise negative decisions would be made. Thus, sometimes there is the need of collecting reputation from other agents to make decision if the decider (Decision agent) does not have sufficient behaviour information. Figure 4.4 shows the collection of reputation sequence diagram.

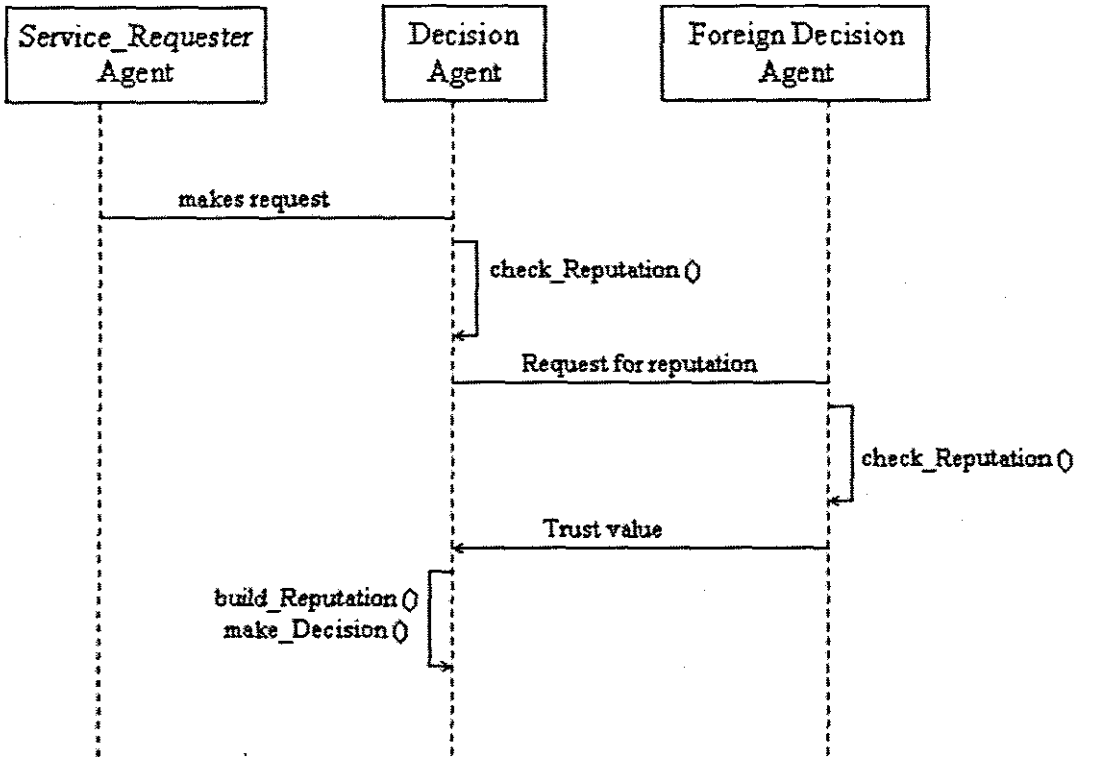

Figure 4. 4: A sequence diagram for Collection of reputation

The DA broadcast the request for reputation if it is necessary to do so. The trusted nodes that have the behaviour information of the requester agent respond with trust values. Figure 4.5 is the activity diagram showing actions taken during trust evaluation and assessment for each mobile agent.

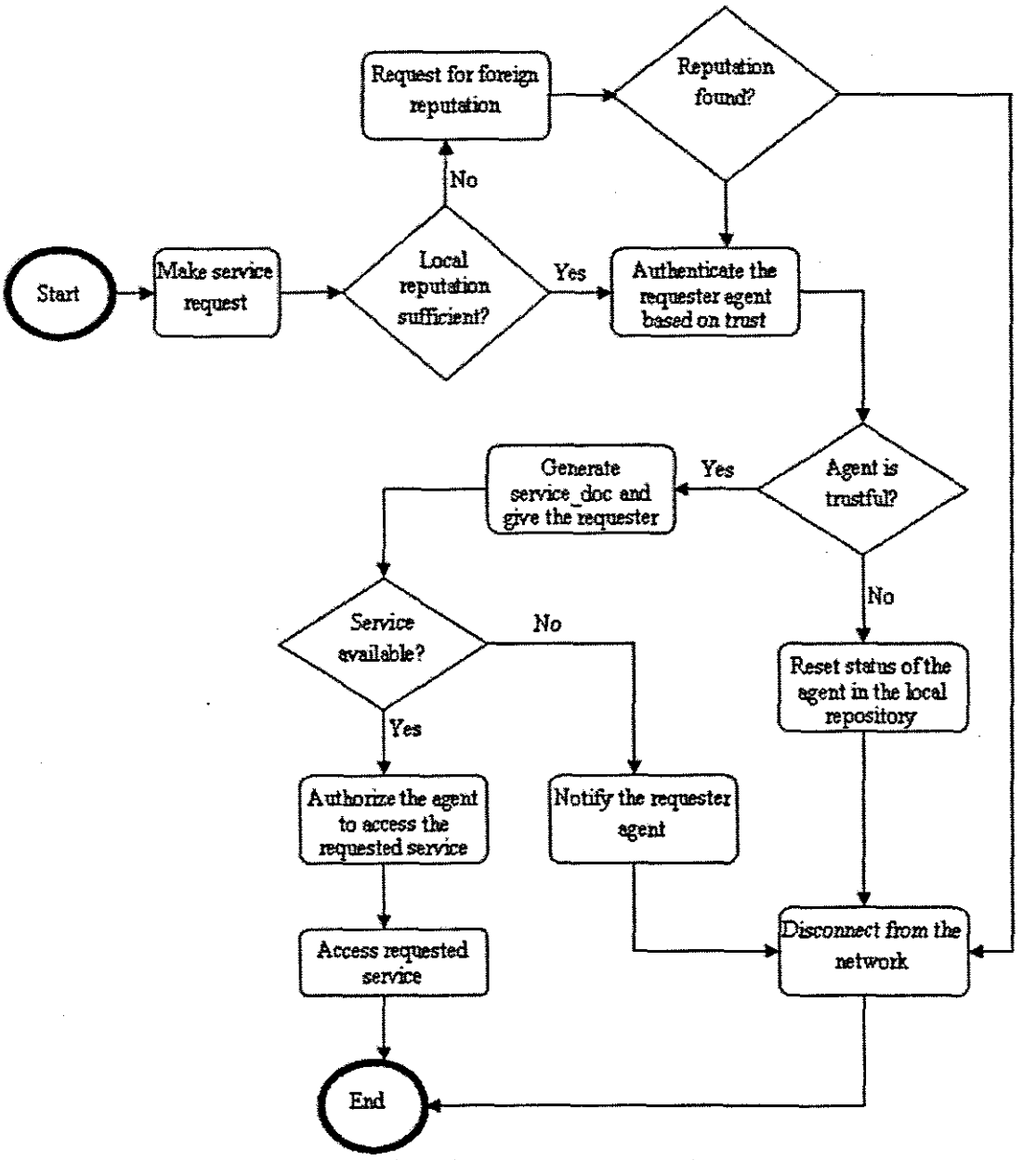

Figure 4. 5: Activity diagram which shows how agents are authenticated and authorized

4.2.3 Digital Signature Algorithms

There is the need of ensuring trust between DA and MA when they are exchanging control during trust evaluation. DA signs the service document with its digital signature and Master agent verifies the digital signature on the service_doc. Figure 4.7 depicts the pseudo code of the algorithm that was used to digitally sign the service_doc. The algorithm generates the private and public keys by using the key-pair generator. The private key is needed in order to create a digital signature, and its corresponding public key will be needed in order to verify the authenticity of the signature. The algorithm uses the SHA1PRNG pseudo-random-number generation algorithm, as provided by the built-in SUN provider. .

```
DA receives request
IF agent is trusted THEN
        generate keys to sign the service_doc
ELSE
        deny the request
ENDIF
Requester agent is notified
```

Figure 4. 6: Pseudo code for signing the Service_doc

When the Master agent (MA) receives the service_doc generated by the DA, it must verity the signature on it before authorizing the requester agent to access services. This is

to ensure that all requester agents have passed through DA for authentication. Therefore, MA uses the signature to ensure trust between itself and the DA. Figure 4.8 shows the digital signature algorithm to verify the signature on the service_doc.

```
MA receives service_doc
IF service_doc is signed THEN
        verify signature
        IF signature is valid and service available THEN
                grant access to information bus
        ELSE
                deny access
        ENDIF
ELSE
        deny access
ENDIF
Requester agent is notified
```

Figure 4. 7: Pseudo code for digital signature verification

The requester agent presents the service_doc to Master Agent and the signature on the document is verified using the algorithm above.

4.2.4 Trustworthiness Evaluation

The trust evaluation of each requester agent is performed based on reputation as it is discussed in section 3.3. Trust evaluation is the core of our model, the basic algorithm to

evaluate trust is used in our simulation in order to authenticate requester agents. The trust evaluation algorithm is illustrated in figure 4.9. In this algorithm, three condition are checked for each agent that request for a service. Firstly, it is assumed that agents are not allowed to perform (make transactions) actions that are more than a specified number. An agent that exceeds the maximum number of actions is regarded as the selfish agent, and it is then denied the access to services when it is making a request. Secondly, a maximum number of bad actions is set in order to evaluate trust. If number of bad actions reaches the maximum, the agent is considered to be untrusted and malicious; hence it is denied the access. Thirdly, the total number of actions of the requester agent must reach the minimum number in order for the reputation value to be computed.

```
Pre-condition: Trust Evaluation, DecisonAgent class
                TNOA = NOGA + NOBA

If (TNOA <= 20)
{
  If (TNOA >=5)
  {
    If (NOBA < 3)
    {
      RepV = NOGA -NOBA/TNOA * 100;
    }
      Else
        The agent is untrusted and malicious
  }
    Else
      Insufficient Reputation, ask for foreign reputation
}
 Else
   The Total number of actions is more than maximum

Post-condition: accept/deny request
```

Figure 4. 8: Trust evaluation algorithm. TNOA – total number of actions, NOBA – number of bad actions, NOGA – number of good actions, RepV – reputation value

This is based on the assumption that, an agent cannot be fairly determined to be either trusted or untrusted from small number of actions it has previously performed. Therefore, the reputation value (trust value) can be calculated when these conditions are met by the requester agent.

4.3 Implementation Environment

The model was implemented using JBuilder5 IDE (Integrated Development Environment) for Java programs development. Java client/server socket programming was used to implement agents. Microsoft Access was used as a repository, for both local and foreign reputations, to store behaviour information (history records) of requester agents. The simulation results statistics was stored in a text file. Two desktop machines were used to run the system. One machine was running client agents and server (with DA), and the other machine was acting as the neighbour node that provides foreign reputation (trust) values when requested by the DA. The ServiceRequesterAgent class sends requests to DecisionAgent class which evaluates trust based on reputation. The DecisionAgent class can also make request to the other machine that runs the duplicate of Decision agent class which we regard as foreign decision agent (FDA).

4.3.1 System User Interface

To show the execution results of the algorithms, we designed interfaces that allow users to run the system. Next, user interfaces are shown with their descriptions.

Figure 4. 9: Typical User Interface showing Nodes to be secured

Figure 10, depicts the client agents that make requests to Decision agent. To run the system, the user must select the name of the machine from the list that is displayed on the interface and click the button send to enable the agent to move to the server where it would execute its task.

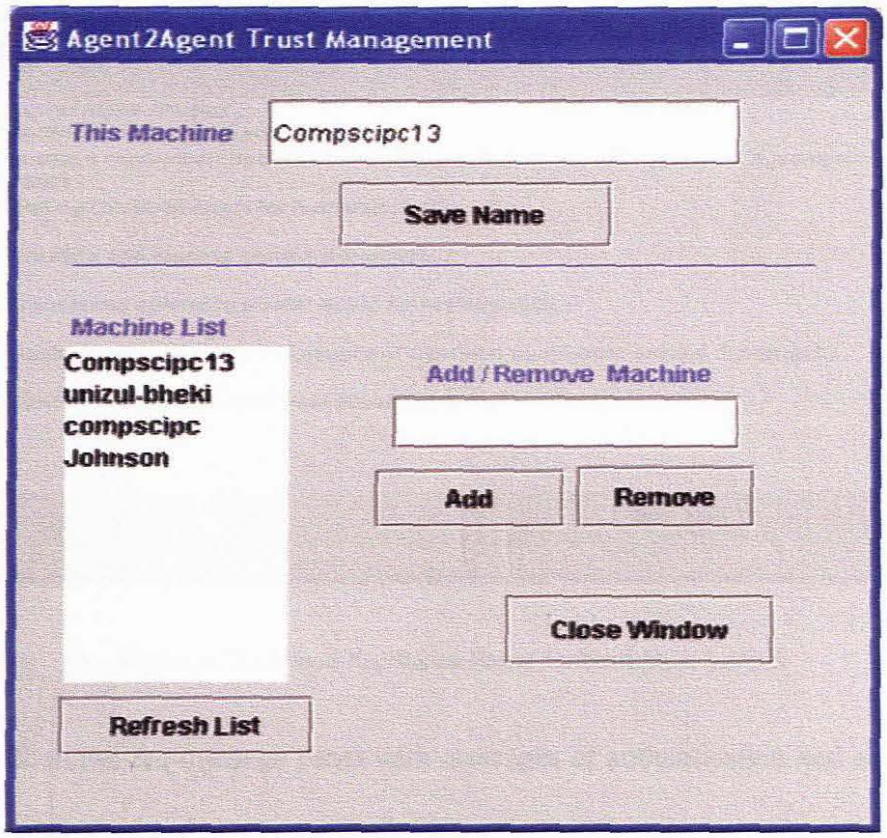

Figure 4. 10: Machine Configuration Interface

Figure 4.11 shows the configuration interface that is used to set the machine that sends the service requester agent. It is also used to add or remove machines to the database. When the agent has moved to the server to make a request, the messages are displayed in a message panel window.

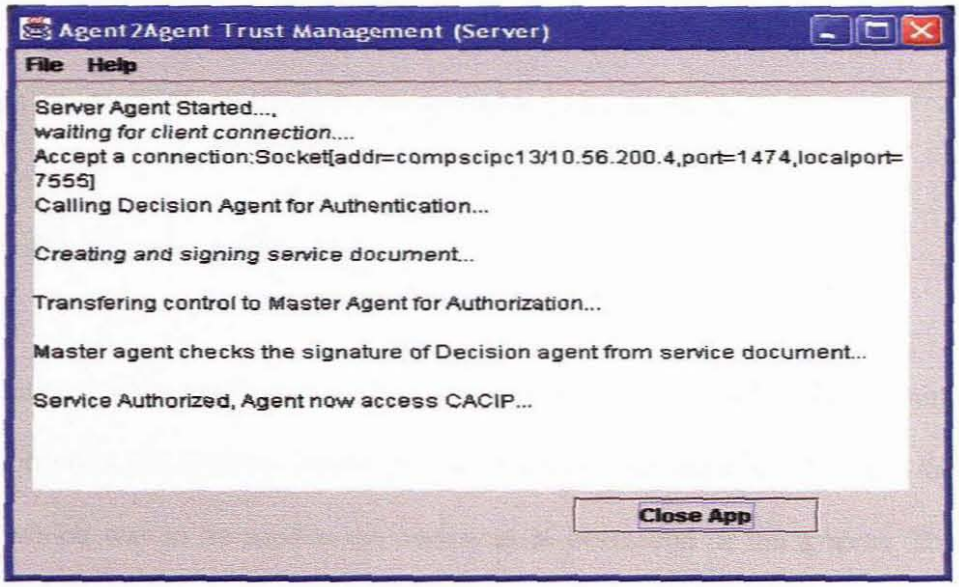

Figure 4. 11: A local Reputation Based Authentication Session

Figure 4.12 shows the message panel with messages of authentication and authorization processes when the local reputation has been used to evaluate trustworthiness of an agent.

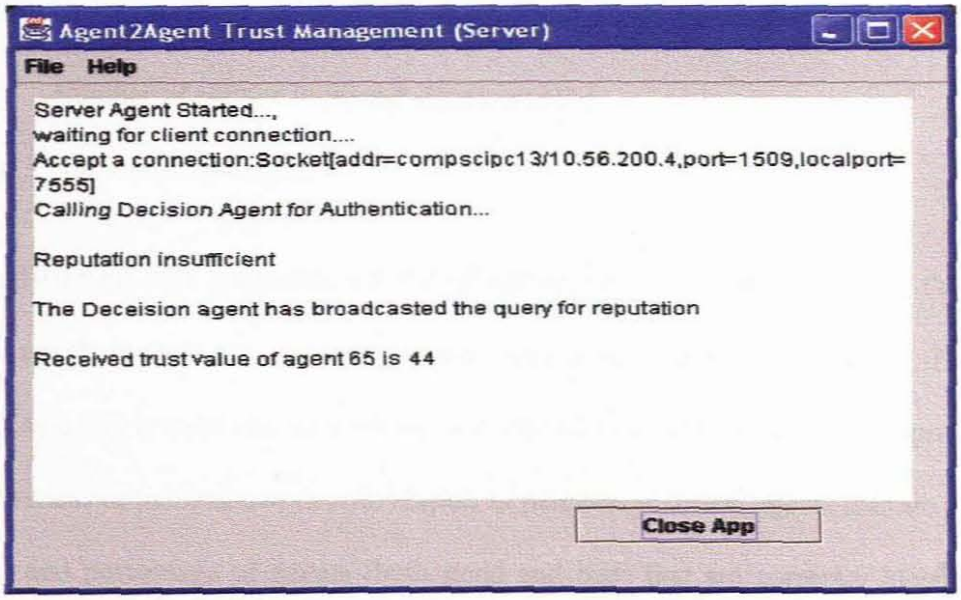

Figure 4. 12: A Foreign Reputation Based Decision Session

Figure 4.13 shows the messages of authentication and authorization processes when foreign reputation (from FDA) has been used to judge an agent.

4.4 Performance Experiment

In order to test the performance of our system, there was a need of determining a value that optimizes the system. Therefore, we conducted an experiment to get the optimal value which was to be used to do performance evaluation of the system. The optimal value would be used to determine the trust level of agents when the developed model is applied. Initially, each agent can be either good or bad agent based on the number of good action and bad actions.

The experiment was conducted as follows:

a) Parameters to get the optimal value from a single test

i) Number of service requester agents and the

ii) Number of bad agents.

We had 10 runs with a constant number of agents, i.e. 200 agents; we varied the number of bad agents in each run. We used constant estimated values (65, 68, 70, 72, 74, 75, 76, 78, 80, and 82) in each run. In each run we wanted to observe a value that optimizes the effectiveness of the framework with respect to percentage of bad agents that are identified as bad and percentage of agents (both good and bad) that are correctly identified. We observed an optimal value from the estimated values. Optimal value is the value that increases the percentage of number of bad agents that were identified as bad and the

percentage of agents that were correctly identified. We then calculated the average of the optimal values of each run. The average optimal value would be used to determine the trust value of each requester agent when performance analysis is done.

4.4.1 Experimental Results

Table 4. 1: The experimental results to find the optimal value

Runs	# of agents with originally bad actions	# of agents found to be bad by using our model	# of agents correctly identified	% of bad agents correctly identified	% of agents correctly identified	Optimal value
1	172	172	188	100%	96%	74.5
2	159	156	191	100%	95%	74.5
3	165	161	190	100%	95%	74.5
4	170	170	190	100%	96%	73
5	168	168	184	100%	92%	75
6	173	173	187	100%	96%	75
7	166	166	182	100%	96%	74.5
8	160	160	173	100%	92%	74.5
9	174	174	183	100%	97%	73
10	162	162	171	100%	97%	74.5
Average						**74.3**

Table 4.1 illustrates the results of the experiment we conducted to get the optimal value to be used to judge service requester agents. It was observed that 74.3 is the optimal value that would be used to compute reputation of agents.

4.4.2 Performance Analysis

In our experiments, we examined the effectiveness of the system in terms of identifying malicious agents as they make requests. We also examined the scalability of the system

as the number of requests increases.

In order to evaluate the performance of our system, for 5 times we kept constant the number of agents that make requests and vary the number of bad agents. We took the average percentage of agents that were sent with bad actions and were found untrusted and malicious by using the model. We also took the average percentage of agents (both bad and good) that were correctly identified with their status. Table 4.2 illustrates the results of the experiment we conducted.

Table 4. 2: The results of the performance analysis

Number of agents sent requests	% of bad agents identified as bad	% of agents correctly identified
100	100.00	94.34
200	99.17	93.12
300	99.50	93.61
400	99.67	94.25
500	99.50	94.10
600	99.53	93.83
700	99.36	93.50
800	99.67	93.42
900	99.54	93.79
1000	99.62	94.07

i). Identification of Bad Agents

Agents that make requests can either be trusted or untrusted agents. Trusted agents are those agents with good reputation, and untrusted agents are those ones with bad reputation. Agents make requests and they must be evaluated based on their status. Therefore, the aim of this experiment was to verify that how many agents were identified

as bad agents out of a number of bad agents that were sent to make requests. This would indicate the effectiveness in authenticating agents that make request. Figure 4.14 plot the identification of malicious agents that the system had evaluated in different runs.

To conduct this experiment, two parameters were varied:

- Total number of agents and the
- Number of bad agents make request.

From this experiment, we observed that the increase of number agents that make request does not affect the performance or effectiveness of the system.

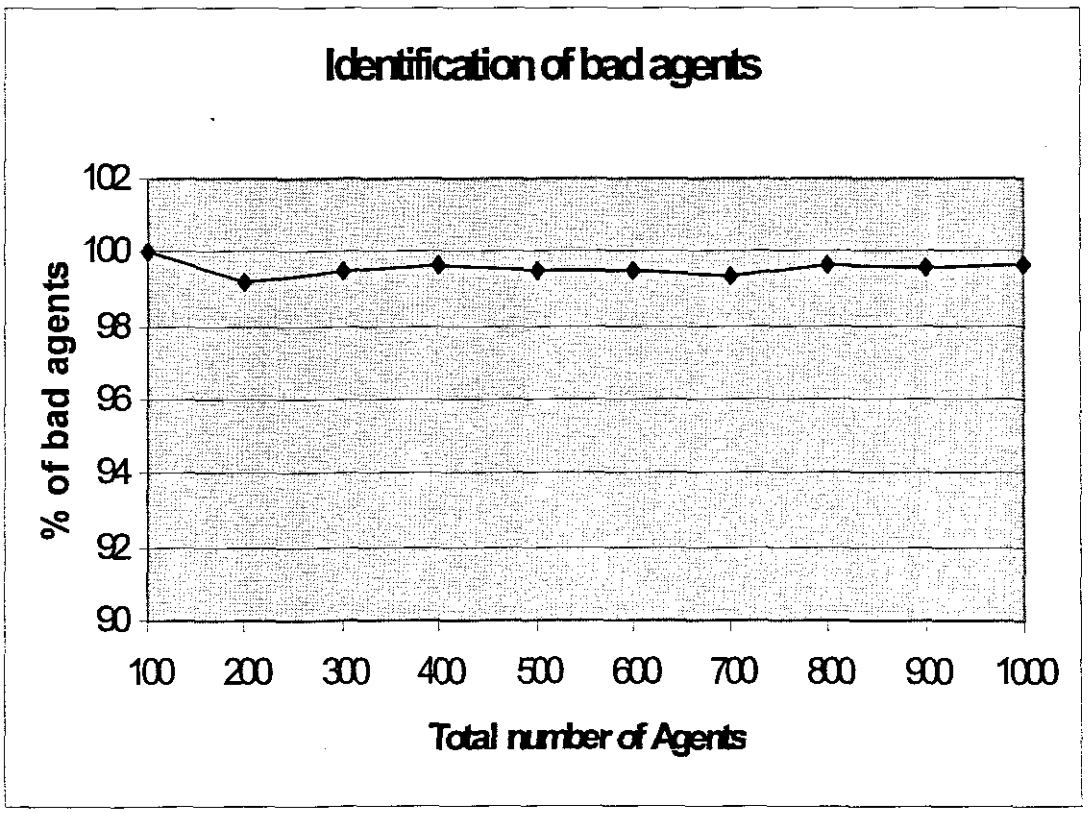

Figure 4. 13: Bad agents Identification based on their reputation

ii). Correctness of Reputation Evaluation

The aim of this experiment was to measure the correctness of the system in evaluating agents based on their reputation. In this experiment we looked at how many percent of agents were correctly identified out of the number of agents that has made requests. Figure 4.15 illustrate a plot of system correctness.

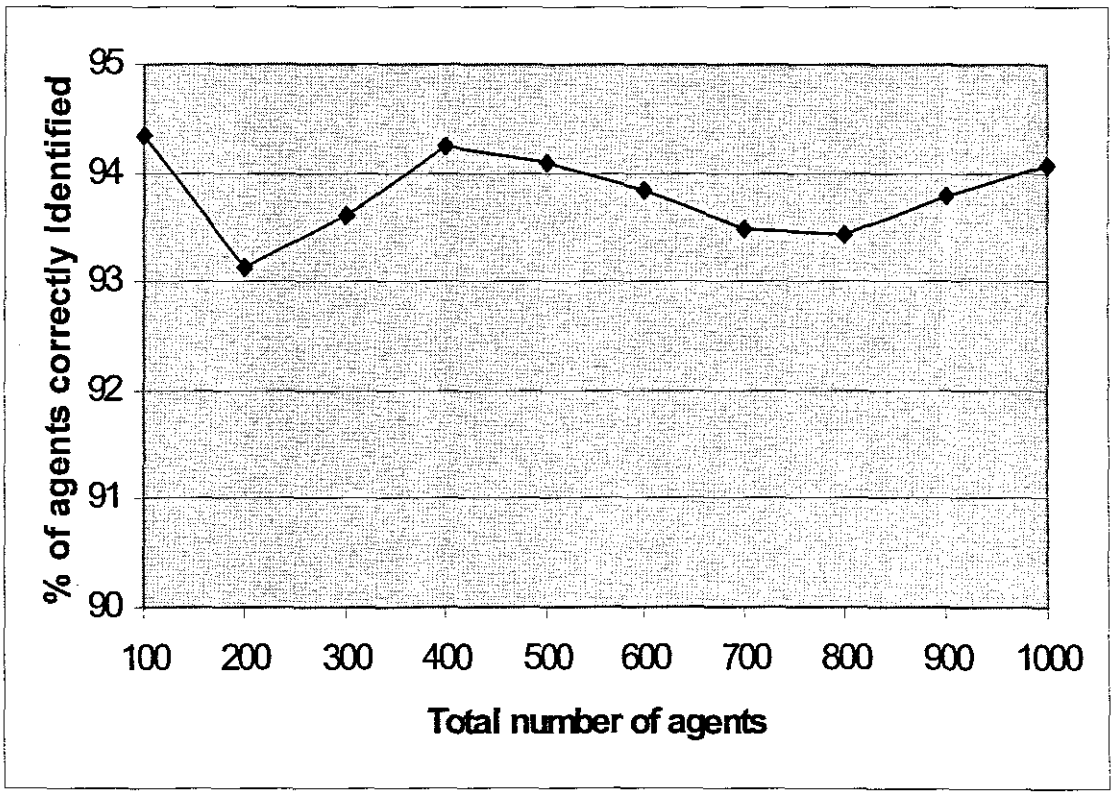

Figure 4. 14: Reputation Evaluation

From the plot, we observed that very few agents were incorrectly identified when they make requests. We also observed that the system is consistent in its performance.

CHAPTER FIVE

5. Conclusion

5.1 Introduction

Security in distributed system is the major challenge that needs to be considered in order to claim success on a system. This research revealed the need of security among agents as they interact to share services. The security challenges that were highlighted in this dissertation brought up the needs for trust management among interacting agents. Reputation-based trust approach in establishing security has been used by many researchers to achieve several goals. We have used reputation of agents to judge them to access available services. Agent technology has been a successful technology to implement systems to search for information and share services over the Internet. The CACIP architecture is the core middleware for agents communicating to share services. Due to the openness of CACIP architecture when agents are interacting, there has been a need of a security scheme that would secure services that are shared in a CACIP based system.

5.2 Conclusion

The openness of CACIP architecture has raised the need of a serious trust management scheme that would ensure security among agents communicating. The agent based trust

management model has been proposed to provide necessary trustworthiness evaluation.

The first objective was to construct a trust management model with specialisation in authentication and authorization capabilities. All agents that want to access services in the information bus must undergo a security check before getting the access to services. Therefore, a proper authentication mechanism is essential to ensure that only trusted agents are permitted to access services. In order to ensure an efficient trust management, authentication and authorization processes are conducted separately. Maintaining reputation and accurately use it has been a challenge. Thus, in our trust model we assign SCA to administer reputation repository. To achieve this objective Agent-to-agent Trust Management Model was established based on various sources of security design principles from existing security schemes. Existing design principles were adopted and reused to solve problem of security in the context of CACIP. Our trust model has some limitations. There is no mechanism that provides privacy on messages that are being transferred from one point to the other as trustworthiness evaluation is in progress.

The second objective was to devise an agent-to-agent authentication and authorization mechanisms that manage trust in mobile systems. Agents that make requests are authenticated based on their reputation. Reputation is obtained either from local reputation repository or from foreign node on the network. This objective was achieved by proposing a reputation-based authentication of agents that make requests. In this mechanism, all agents are authenticated before accessing available services. The authentication process was assigned to one agent that checks all agents that make requests. Both authentication and authorization processes use digital signatures to ensure

trustworthiness between each other. Thus, one agent that does authentication signs the message and sends it to the other agent for authorization. The signature must be verified before the message is perceive to be a valid request message.

The third objective was to show that the scheme or mechanism functions in a real environment by evaluating its performance experimentally. The simulation of the proposed mechanism was conducted to realise this objective. The aim of this experiment was to test if the model would be able to correctly identify agents based on their reputation. The results of the experiment showed that agent based trust management model can correctly and accurately identify agents according to their status. The agents that had bad reputation were correctly identified as bad agents after sending their requests. The model also showed to be consistent in evaluating trustworthiness because the increase of requests was not affecting the accuracy of the system.

5.3 Future Work

Experiments that were conducted showed that the proposed trust model could make a difference in Internet based application systems, for instance e-Bay systems, to authenticate consumers and auctioneers. The issue of privacy is essential in agent-based systems; requests/response messages that are passed from one point to the other during trustworthiness evaluation needs to be secured. Therefore, an issue for investigation in future is how the issue of privacy can be brought into proposed model. The security principles, i.e. robustness, end-to-end security, fairness and scalability are still need to be considered for experiment in future to enhance the proposed model.

The proposed security model could be tested for future generation (ad hoc) based application systems. The identification of an incoming node can be a challenging task as an appropriate mechanism is not in place. Therefore, as the future work of this study, the agents that do authentication and authorization can be suitable in controlling participation of nodes in an ad hoc network.

APPENDIX A
USER MANUAL

This section of the thesis presents the steps that a user needs to follow in order to use the trust management system.

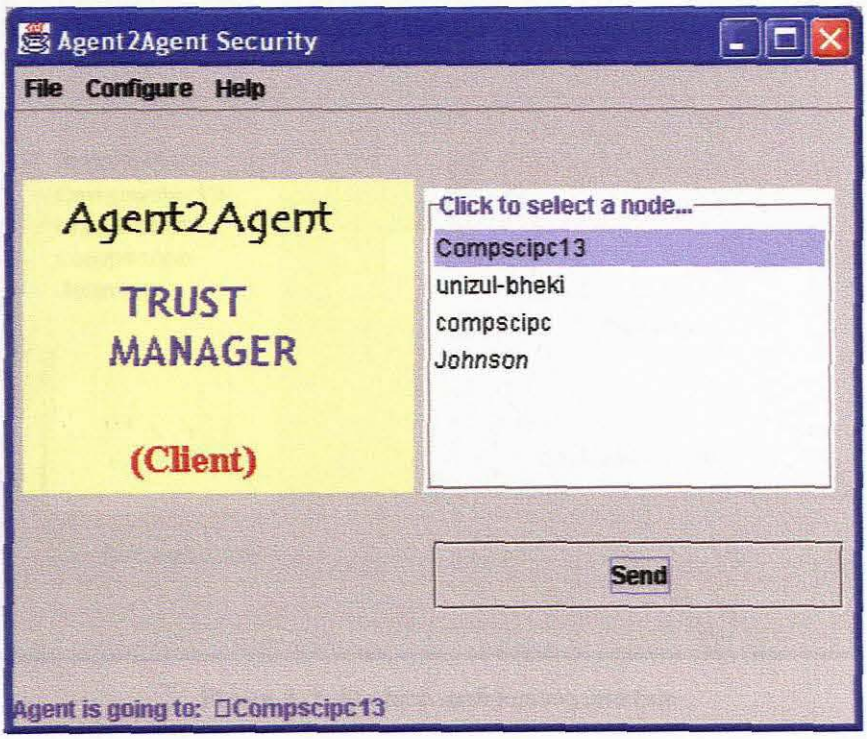

Figure A. 1: User interface showing nodes participating in the network

Our trust management system is a client/server application, therefore it runs using at least two machines. Figure C.1 is the user interface on the client side of the application. The list (computer names) of nodes or machines participating in the running of the system is shown on the interface. A user must select, from the list, a machine to which a request is to be sent. Once the machine has been selected, click **Send** button to send the agent's request to the selected machine. The Configure item on the main menu is used to set machines that are going to participate in the running of the system. When configure is

clicked, figure A.2 is shown allowing the user to add or remove machines on the network and to set the machine from which the requester agent would be sent.

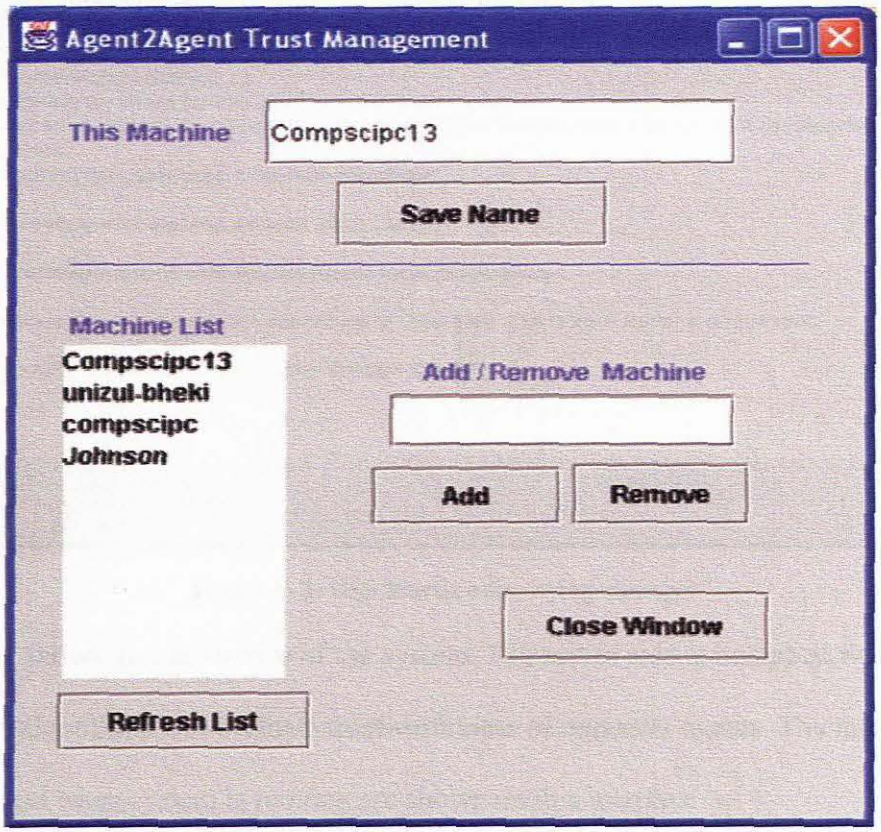

Figure A. 2: Machine configuration interface

Figure A.2 is used to configure machines that are participating to run the system. The This Machine text field is used to type the name of the client machine that is going to sent requests to the machine that runs trust evaluation algorithms. When the machine name has been correctly typed on the text field, **Save Name** button is clicked to save the name of the machine. The list box, Machine List, displays names of machines that can participate to run the system. In order to add or remove the machine, a user type the name of the machine in the Add/Remove Machine text field and click either **Add** or **Remove** buttons. The **Refresh** button is used to refresh the machine list after adding or removing a

machine. The Close Window button is used to close the window once configuration has been completed.

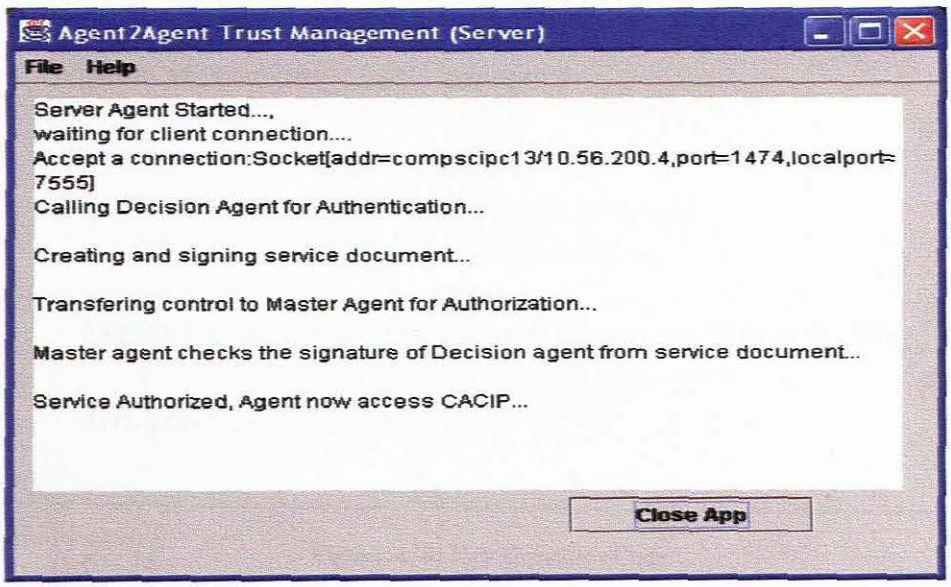

Figure A. 3: User Interface displaying messages

Figure A.3 shows the server side of the system. The server side is equipped with our trust evaluation algorithms that evaluate trustworthiness of requester agents. The messages that are generated when system is running are shown on this interface.

Figure A.4 shows the Java algorithm that generates keys to sign the service document in DA. The code generates the private and public keys by using the key-pair generator. The private key is needed in order to create a digital signature, and its corresponding public key will be needed in order to verify the authenticity of the signature. The algorithm uses the SHA1PRNG pseudo-random-number generation algorithm, as provided by the built-in SUN provider.

```
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA", "SUN");
SecureRandom random = SecureRandom.getInstance("SHA1PRNG","SUN");

      keyGen.initialize(1024, random);

       KeyPair pair = keyGen.generateKeyPair();
          PrivateKey priv = pair.getPrivate();
           PublicKey pub = pair.getPublic();

        Signature dsa = Signature.getInstance("SHA1withDSA", "SUN");

        dsa.initSign(priv);
```

Figure A. 4: Key Generator Code

Figure A.5 shows the code to verify the signature on the service_doc that has been signed by DA using the key generated in figure A.4.

```
X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec(encKey);

      KeyFactory keyFactory = KeyFactory.getInstance("DSA", "SUN");
         PublicKey pubKey = keyFactory.generatePublic(pubKeySpec);

          FileInputStream sigfis = new FileInputStream(args[1]);
           byte[] sigToVerify = new byte[sigfis.available()];
           sigfis.read(sigToVerify );

           Signature sig = Signature.getInstance("SHA1withDSA", "SUN");
            sig.initVerify(pubKey);
```

Figure A. 5: Signature Verifier Code

APPENDIX B

UML DESIGN DOCUMENTATION

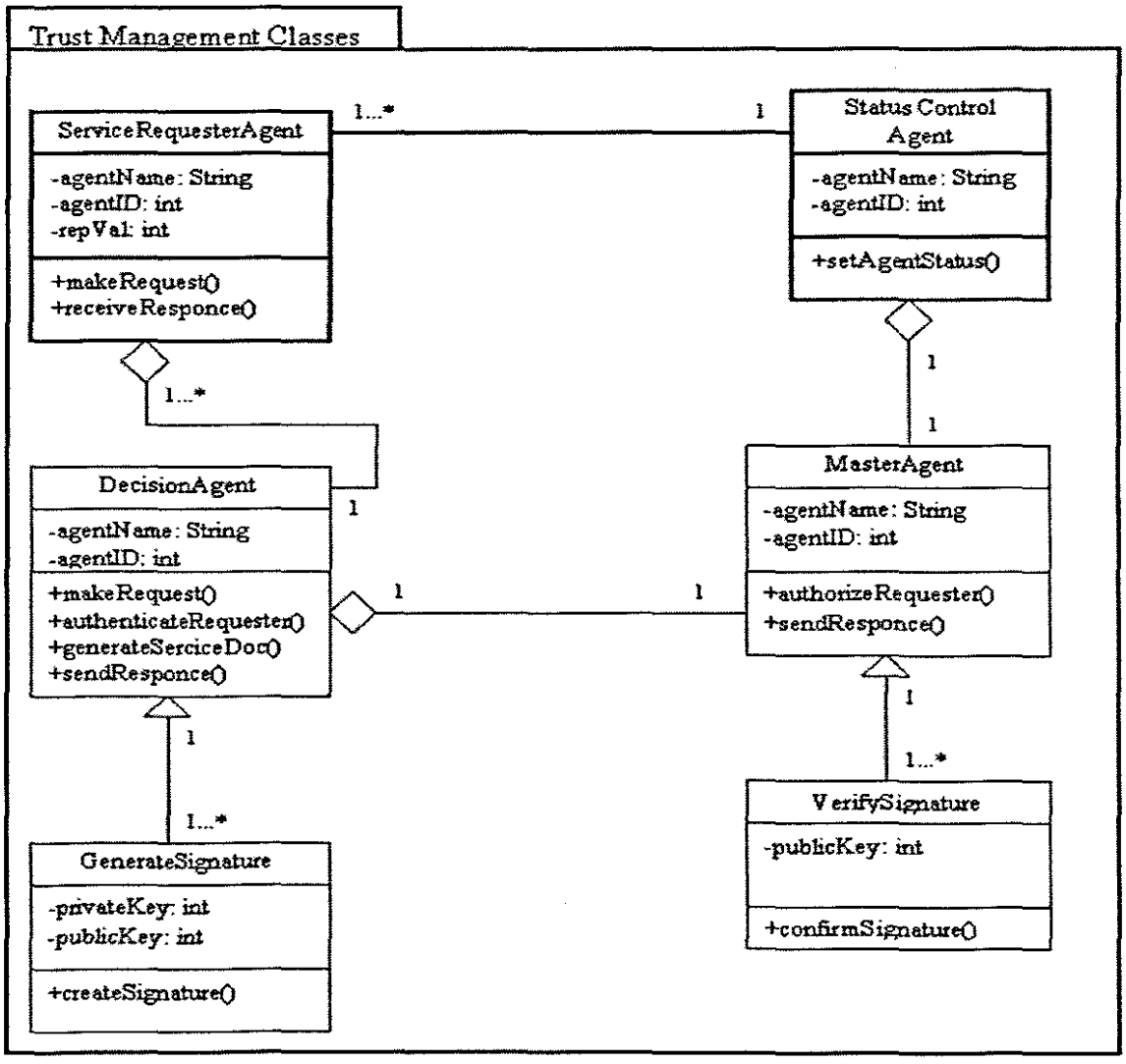

Figure B. 1: UML Class Diagram

APPENDIX C

CODE

Main Server Class

```
package agent2agenttrust;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Rectangle;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextArea;

public class FrmMainServer extends JFrame {

    private BorderLayout layoutMain = new BorderLayout();
    private JPanel panelCenter = new JPanel();
    private JMenuBar menuBar = new JMenuBar();
    private JMenu menuFile = new JMenu();
    private JMenuItem menuFileExit = new JMenuItem();
    private JMenu menuHelp = new JMenu();
    private JMenuItem menuHelpAbout = new JMenuItem();
    private JLabel statusBar = new JLabel();
    private JTextArea jTextArea1 = new JTextArea();
    private JButton jButton1 = new JButton();

    String msg;

    public FrmMainServer() {
        try {
            jbInit();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void jbInit() throws Exception {
                this.setJMenuBar( menuBar );
      this.getContentPane().setLayout( layoutMain );
      panelCenter.setLayout( null );
      this.setSize(new Dimension(481, 401));
      this.setTitle( "Agent2Agent Trust Management (Server)" );
      menuFile.setText( "File" );
      menuFileExit.setText( "Exit" );
      menuFileExit.addActionListener( new ActionListener() { public void
      actionPerformed( ActionEvent ae ){fileExit_ActionPerformed( ae );}} );
      menuHelp.setText( "Help" );
      menuHelpAbout.setText( "About" );
```

```
        menuHelpAbout.addActionListener( new ActionListener() { public void
        actionPerformed( ActionEvent ae ){helpAbout_ActionPerformed( ae);}} );
        statusBar.setText( "" );
        jTextArea1.setBounds(new Rectangle(10, 10, 445, 275));
        jTextArea1.setEditable(false);
        jButton1.setText("Close App");
        jButton1.setBounds(new Rectangle(285, 290, 125, 25));
         jButton1.addActionListener(new ActionListener() {
                                public void actionPerformed(ActionEvent e) {
                                          jButton1_actionPerformed(e);
                                    }
                                }
         );
         menuFile.add( menuFileExit );
         menuBar.add( menuFile );
         menuHelp.add( menuHelpAbout );
         menuBar.add( menuHelp );
         this.getContentPane().add( statusBar, BorderLayout.SOUTH);
         this.getContentPane().add( panelCenter, BorderLayout.CENTER);
         panelCenter.add(jButton1, null);
         panelCenter.add(jTextArea1, null);
         jTextArea1.setText("");
         jTextArea1.setLineWrap(true);

    }

    void fileExit_ActionPerformed(ActionEvent e)
    {
        System.exit(0);
    }

    void helpAbout_ActionPerformed(ActionEvent e)
    {
       FrmMainServer_AboutBoxPanel1(), "About",
    }

    private void jButton1_actionPerformed(ActionEvent e)
    {
        System.exit(0);
    }

    public void setMessage(String info)
    {
        jTextArea1.append(info +"\n");
    }
}
```

Agent Class

```
package agent2agenttrust;

import java.net.*;
import java.util.*;
import java.io.*;
import java.net.URL;
import java.awt.*;

public class Agent implements Serializable
{
  private boolean haveMove = false;
  private String whereTo = "";
  private final int agentID = 65;
```

```
FrmMainServer frame;

public Agent()
{
   frame = new FrmMainServer();
}
public void run ()
{
  if (!haveMove)
  {
    haveMove=true;

    go();   //move to the next machine
  }
  else
    {
      monitor();
    }
  }

public int getAgentID()
{
  return agentID;
}

public void go()
{
  try
  {

    InetAddress add = InetAddress.getByName(whereTo);
    Socket client = new Socket(add,7555);
    ObjectOutputStream output =
    new ObjectOutputStream(client.getOutputStream());
    ObjectInputStream input  = new
    ObjectInputStream(client.getInputStream());

    //serialize agent

    output.writeObject(this);
    output.flush();
    client.close();

  }
    catch(IOException e)
    {
       System.out.println("IO Exception ");e.printStackTrace();
    }
    catch(Exception e)
    {
      System.out.println("Some unidentified flying exception!");
      e.printStackTrace();
    }
  }

  private void monitor()
  {

    frame.setMessage("Mobile agent is now accessing service...");

  }

  public void setNextMachine(String node)
```

```
    {
        whereTo = node;
    }
}
```

Decision Agent Class

```
package agent2agenttrust;

import java.io.*;
import java.util.*;
import java.security.*;

public class DecisionAgent
{

  GenSig gen ;  // the signature class for digital signature...
  public DecisionAgent()
  {
     gen = new GenSig();
  }

  public boolean reputationChecker(int repVal)
  {

    if (repVal > 5)         //if reputation is sufficient
    {
      //create a service document for Master Agent to use
      try
      {

         FileOutputStream out = new FileOutputStream("Servicedocument.txt");
         String agentName = "Agent2" +"\n";
         String serviceType = "Weather Calculator" +"\n";
         String timeRequired = "10" +"\n";
         out.write(agentName.getBytes());
         out.write(serviceType.getBytes());
         out.write(timeRequired.getBytes());
         out.close();

      }catch(IOException ie)
       {
          ie.printStackTrace();
       }
          gen.createSig();
          return true;
    }
    else
    {
        return false;
    }
  }
}
```

Master Agent Class

```
package agent2agenttrust;

import java.io.*;
import java.util.*;
```

```
public class MasterAgent
{
  private String agentName;
  VerSig verifier;
  public MasterAgent()
  {
    verifier = new VerSig();
  }

  public boolean authorizeService(int polReq)
  {
     if(polReq >= 8)
     {

       try
       {
           BufferedReader inbr = new BufferedReader(new FileReader(new
           File("Servicedocument.txt")));

           String agentName;
           String serviceType;
           String timeRequired;

           agentName =  inbr.readLine();
           serviceType = inbr.readLine();
           timeRequired = inbr.readLine();

           System.out.println ("AgentName: " + agentName);
           System.out.println ("Time is: " + timeRequired);

           inbr.close();

        }
        catch(Exception e)
         {
            e.printStackTrace();
         }
             verifier.confirmSig();
             return true;
      }else
        {
           return false;
        }

   }
}
```

Generate Signature Class

```
package agent2agenttrust;

import java.io.*;
import java.security.*;

class GenSig
{
  GenSig(){}

  public void createSig()
  {

    /* Generate a DSA signature */
```

```
    try{
       /* Generate a key pair */

       KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA", "SUN");
       SecureRandom random = SecureRandom.getInstance("SHA1PRNG", "SUN");

       keyGen.initialize(1024, random);

       KeyPair pair = keyGen.generateKeyPair();
       PrivateKey priv = pair.getPrivate();
       PublicKey pub = pair.getPublic();

       /* Create a Signature object and initialize it with the
       private key */

       Signature dsa = Signature.getInstance("SHA1withDSA", "SUN");

       dsa.initSign(priv);

       /* Update and sign the data */

       FileInputStream fis = new FileInputStream("Servicedocument.txt");
       BufferedInputStream bufin = new BufferedInputStream(fis);
       byte[] buffer = new byte[1024];
       int len;
       while (bufin.available() != 0)
       {
         len = bufin.read(buffer);
          dsa.update(buffer, 0, len);
         };

            bufin.close();

            /* Now that all the data to be signed has been read in,
                    generate a signature for it */

            byte[] realSig = dsa.sign();

            /* Save the signature in a file */
            FileOutputStream sigfos = new FileOutputStream("sig");
            sigfos.write(realSig);

            sigfos.close();


            /* Save the public key in a file */
            byte[] key = pub.getEncoded();
            FileOutputStream keyfos = new FileOutputStream("suepk");
            keyfos.write(key);

            keyfos.close();

        } catch (Exception e) {
            System.err.println("Caught exception " + e.toString());
        }

    };

}
```

Verify Signature Class

```
package agent2agenttrust;

import java.io.*;
import java.security.*;
import java.security.spec.*;

class VerSig {

     VerSig(){}

    public void confirmSig() {

        /*Verify a DSA signature*/

         try{

            /*import encoded public key*/

            FileInputStream keyfis = new FileInputStream("suepk");
            byte[] encKey = new byte[keyfis.available()];
            keyfis.read(encKey);

            keyfis.close();

            X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec(encKey);

            KeyFactory keyFactory = KeyFactory.getInstance("DSA", "SUN");
            PublicKey pubKey = keyFactory.generatePublic(pubKeySpec);

            /* input the signature bytes */
            FileInputStream sigfis = new FileInputStream("sig");
            byte[] sigToVerify = new byte[sigfis.available()];
            sigfis.read(sigToVerify );

            sigfis.close();

            /*create a Signature object and initialize it with the
            public key*/
            Signature sig = Signature.getInstance("SHA1withDSA", "SUN");
            sig.initVerify(pubKey);

            /* Update and verify the data */

            FileInputStream datafis = new
            FileInputStream("servicedocument.txt");
            BufferedInputStream bufin = new BufferedInputStream(datafis);

            byte[] buffer = new byte[1024];
            int len;
            while (bufin.available() != 0) {
                len = bufin.read(buffer);
                sig.update(buffer, 0, len);
                };

            bufin.close();


            boolean verifies = sig.verify(sigToVerify);
```

```
            //System.out.println("signature verifies: " + verifies);

        } catch (Exception e) {
            System.err.println("Caught exception " + e.toString());
        };

    }

}
```

Security Application Class

```
package agent2agenttrust;

import java.awt.*;
import javax.swing.*;
import java.lang.*;

public class SecurityApp
{
    public SecurityApp()
    {
        JFrame frame = new FrmMain();
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        if (frameSize.height > screenSize.height) {
            frameSize.height = screenSize.height;
        }
        if (frameSize.width > screenSize.width) {
            frameSize.width = screenSize.width;
        }
        frame.setLocation( ( screenSize.width - frameSize.width ) / 2, (
screenSize.height - frameSize.height ) / 2 );
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        frame.setVisible(true);
    }

    public static void main(String[] args)
    {
        try
        {

UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
        }
        catch (Exception e)
         {
            e.printStackTrace();
        }
          new SecurityApp();
    }
}
```

Security Application Server Class

```
package agent2agenttrust;

import java.awt.Dimension;
import java.awt.Toolkit;
```

```
import javax.swing.JFrame;
import javax.swing.UIManager;
import java.io.*;
import java.net.*;
import java.util.*;
import java.net.URL;
import java.awt.*;

public class SecurityAppServer extends Thread
{

   //public String myName;
   boolean reputationSufficient;
   boolean serviceAllowed;

   public static final int PORT = 7555;

   FrmMainServer frame;

   public SecurityAppServer()
   {
        frame = new FrmMainServer();
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        if (frameSize.height > screenSize.height)
        {
            frameSize.height = screenSize.height;
        }
        if (frameSize.width > screenSize.width)
        {
            frameSize.width = screenSize.width;
        }
            frame.setLocation( ( screenSize.width - frameSize.width ) / 2, (
            screenSize.height - frameSize.height ) / 2 );
            frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
            frame.setVisible(true);

            reputationSufficient = false;
            serviceAllowed = false;

            frame.setMessage("Server Agent Started...," +"\n" +"waiting for
            client connection....");
    }

    public void run()
    {
       while(true)
       {
         try
           {

            ServerSocket server = new ServerSocket(PORT);
            Socket soc = server.accept();

            //receiveBroadCast(soc);
            //System.out.println("Accept a connection:"+soc);
            frame.setMessage("Accept a connection:"+ soc);

            //Need a code to save all clients that are connected

            ObjectOutputStream output = new
            ObjectOutputStream(soc.getOutputStream());
            ObjectInputStream input =  new
```

```
ObjectInputStream(soc.getInputStream());
Agent agent = (Agent) input.readObject();
System.out.println("Agent ID: "+ agent.getAgentID());
soc.close();
server.close();

//creating the other agents

DecisionAgent dAgent = new DecisionAgent();
MasterAgent mAgent = new MasterAgent();


frame.setMessage("Calling Decision Agent for
Authentication...\n");
int repV = (int)(1 + Math.random()* 20);
System.out.println("Reputation value is " + repV);
reputationSufficient = dAgent.reputationChecker(4);//repV);

int polReq = (int)(2 + Math.random()*15);
System.out.println("Number of policy requirements received is "+
polReq);

if (reputationSufficient)
{

   frame.setMessage("Creating and signing service
   document...\n");

   Thread.sleep(5000);

   frame.setMessage("Transfering control to Master Agent for
   Authorization...\n");

   serviceAllowed = mAgent.authorizeService(polReq);

   Thread.sleep(5000);

   frame.setMessage("Master agent checks the signature of
   Decision
   agent from service document...\n");

   Thread.sleep((long)(Math.random()*5000));

   if (serviceAllowed)
   {

      // agent can now run when service is allowed...

      frame.setMessage("Service Authorized, Agent now access
      CACIP...\n");

      agent.run();

   }
    else
      {

        frame.setMessage("Service is not allowed, call killer
        agent
        to terminate...\n");
        frame.setMessage("The agent "+ agent.getAgentID()+" has
        been isolated");
      }
```

```
            }
              else
              {
                 frame.setMessage("Reputation insufficient\n\n The
                 Deceision
                 agent has broadcasted the query for reputation\n");
                 InetAddress []  addresses =
                 {InetAddress.getByName("10.56.200.237")};
                 int [] ports  = {8600};
                 int [] trustValues =
                 broadCastAgent(agent.getAgentID(),addresses,ports);
                 evaluateTrust(trustValues);

              }

          }catch(IOException ip)
          {
             System.out.println("IO Exception here");
             ip.printStackTrace();
          }
            catch(InterruptedException inexc)
            {
               System.out.println("There is a problem with the sleeping
               thread");
            }
            catch(ClassNotFoundException e)    .
          {
             System.out.println("Class not found exception here");
          }
      }
  }
public int[] broadCastAgent(int agentID,InetAddress [] address,int []
ports) throws IOException
{

   ArrayList arr = new ArrayList();
   for(int i = 0; i < address.length;i++)
   {
      Socket s = new Socket(address[i].getHostAddress(),ports[i]);
      PrintWriter out = new PrintWriter(s.getOutputStream(),true);
      out.println("AgentID"+ agentID);
      out.flush();

      BufferedReader  r =  new BufferedReader(new
      InputStreamReader(s.getInputStream()));

      String in = null;

      while((in = r.readLine()) != null)
      {
         if(in.startsWith("TrustValue"))
         {
           int trustValue =
           Integer.parseInt(in.trim().substring(11,in.length()));
           arr.add(new Integer(trustValue));
           frame.setMessage("Received trust value of agent "+agentID+ "
           is "+ trustValue);
         }

       }

    }
        arr.trimToSize();
```

```
        int [] trustValues = new int[arr.size()];
      for(int i =0;i < trustValues.length;i++)
      {
         trustValues[i] = ((Integer)arr.get(i)).intValue();
      }
         return trustValues;
}

 public void evaluateTrust(int [] trustValues)
 {
    for(int i = 0; i < trustValues.length; i++)
    {
       if(trustValues[i] < 50)
       {
         System.out.println("The agent " + trustValues[i] + " is
         untrusted");
       }
        else
         {
           System.out.println("Agent authenticated through the use of
           foreign reputation");
         }
    }
  }

public void receiveBroadCast(Socket soc)
{

 try
 {
   BufferedReader r  = new BufferedReader(new
   InputStreamReader(soc.getInputStream()));
   PrintWriter writer  = new PrintWriter(soc.getOutputStream(),true);
   String s = null;
   while((s = r.readLine()) != null)
   {
      int trustValue = (int)(Math.random()*100);
      if(s.trim().startsWith("AgentID"))
      {
         int agentID =
         Integer.parseInt(s.trim().substring(7,s.length()));
         System.out.println("Agent ID: "+ agentID);
         //Does checking
         System.out.println("Trust value for " + agentID + " is "+
         trustValue);
         writer.println("TrustValue "+  trustValue);
         writer.flush();
      }
    }
  }
   catch(IOException ioe)
   {
      System.out.println("There was a problem in receiving broadcast");
   }

}

public static void main(String[] args)
{
   try {


      UIManager.setLookAndFeel
```

```
            (UIManager.getCrossPlatformLookAndFeelClassName());
        }
          catch (Exception e)
          {
             e.printStackTrace();
          }
             SecurityAppServer sec = new SecurityAppServer();

             sec.start();
    }
}
```

Main Interface Form Class

```
package agent2agenttrust;

import java.awt.*;
import java.awt.event.*;
import java.awt.event.ActionEvent;
import javax.swing.*;
import java.net.URL;
import java.sql.*;

public class FrmMain extends JFrame {
    private BorderLayout layoutMain = new BorderLayout();
    private JPanel panelCenter = new JPanel();
    private JMenuBar menuBar = new JMenuBar();
    private JMenu menuFile = new JMenu();
    private JMenuItem menuFileExit = new JMenuItem();
    private JMenu menuConfig = new JMenu();
    private JMenuItem menuConfigSettings = new JMenuItem();
    private JMenu menuHelp = new JMenu();
    private JMenuItem menuHelpAbout = new JMenuItem();
    private JLabel statusBar = new JLabel();
    private JPanel jPanel1 = new JPanel();
    private JLabel jLabel1 = new JLabel();
    private JLabel jLabel2 = new JLabel();
    private JLabel jLabel3 = new JLabel();
    private JLabel jLabel4 = new JLabel();
    private JPanel jPanel2 = new JPanel();
    private JList jList1;

    Agent agent = new Agent();

    private JButton btnSend = new JButton();

    DefaultListModel listModel;

       FrmConfigure config;

    public FrmMain() {
        try {
            jbInit();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void jbInit() throws Exception
    {

       listModel = new DefaultListModel();
```

```
Connection dbConn = null ;

try
{
  Driver d = (Driver)Class.forName
  ("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
  String URL = "jdbc:odbc:" + "MyDbs2";
  dbConn = DriverManager.getConnection( URL,"nouser","nopassword");

  Statement stmt;
  ResultSet rs;
  stmt = dbConn.createStatement();

  rs = stmt.executeQuery ("SELECT * FROM MachineList");

  while (rs.next())
  listModel.addElement(rs.getString ("MachineName"));
  dbConn.close();

}
 catch (Exception e)
 {

    JOptionPane.showMessageDialog(this,"Can't Open Node Database",
    "Program error",JOptionPane.ERROR_MESSAGE);
    e.printStackTrace();
    }
        //end database


        jList1 = new JList(listModel);
        //jList1.setVisibleRowCount(-1);

        JScrollPane listScroller = new JScrollPane(jList1);
        listScroller.setAutoscrolls(true);
        //listScroller.setPreferredSize(new Dimension(250, 80));

        this.setJMenuBar( menuBar );
        this.getContentPane().setLayout( layoutMain );
        panelCenter.setLayout( null );
        this.setSize(new Dimension(438, 370));
        this.setTitle( "Agent2Agent Security" );
        menuFile.setText( "File" );
        menuFileExit.setText( "Exit" );
        menuFileExit.addActionListener( new ActionListener() { public
        void actionPerformed( ActionEvent ae ) {
        fileExit_ActionPerformed( ae ); } } );

        menuConfig.setText( "Configure" );
        menuConfigSettings.setText( "Settings" );
        menuConfigSettings.addActionListener( new ActionListener() {
        public void actionPerformed( ActionEvent ae ) {
        configSettings_ActionPerformed( ae ); } } );

        menuHelp.setText( "Help" );
        menuHelpAbout.setText( "About" );
        menuHelpAbout.addActionListener( new ActionListener(){ public
        void actionPerformed( ActionEvent ae ) {
        helpAbout_ActionPerformed( ae ); } } );
        statusBar.setText( "Application Started..." );
        jPanel1.setBounds(new Rectangle(5, 25, 200, 160));
        jPanel1.setLayout(null);
        jPanel1.setBorder(BorderFactory.createEmptyBorder(0,0,0,0));
```

```
            jPanel1.setBackground(new Color(255, 239, 214));
            jLabel1.setText("Agent2Agent");
            jLabel1.setToolTipText("null");
            jLabel1.setFont(new Font("Tempus Sans ITC", 1, 22));
            jLabel1.setForeground(new Color(16, 16, 16));
            jLabel1.setBounds(new Rectangle(20, 5, 150, 30));
            jLabel2.setText("TRUST");
            jLabel2.setToolTipText("null");
            jLabel2.setFont(new Font("Trebuchet MS", 1, 21));
            jLabel2.setBounds(new Rectangle(50, 50, 90, 25));
            jLabel3.setText("MANAGER");
            jLabel3.setToolTipText("null");
            jLabel3.setFont(new Font("Trebuchet MS", 1, 21));
            jLabel3.setBounds(new Rectangle(45, 75, 95, 25));
            jLabel4.setText("(Client)");
            jLabel4.setToolTipText("null");
            jLabel4.setFont(new Font("Times New Roman", 1, 20));
            jLabel4.setForeground(new Color(231, 0, 0));
            jLabel4.setBounds(new Rectangle(55, 130, 65, 25));
            jList1.setBounds(new Rectangle(210, 30, 210, 155));
            jList1.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
            jList1.setBorder(BorderFactory.createTitledBorder
            ("Click to select a node..."));
            jList1.setValueIsAdjusting(true);
            btnSend.setText("Send");
            btnSend.setBounds(new Rectangle(215, 210, 210, 35));
            btnSend.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent e) {
                      btnSend_actionPerformed(e);
                                  }
                              }
            );
            menuFile.add( menuFileExit );
            menuBar.add( menuFile );
            menuConfig.add( menuConfigSettings );
            menuBar.add( menuConfig );
            menuHelp.add( menuHelpAbout );
            menuBar.add( menuHelp );
            this.getContentPane().add( statusBar, BorderLayout.SOUTH);
            this.getContentPane().add(panelCenter, BorderLayout.CENTER);
            jPanel1.add(jLabel4);
            jPanel1.add(jLabel3);
            jPanel1.add(jLabel2);
            jPanel1.add(jLabel1);
            panelCenter.add(btnSend, null);
            panelCenter.add(jList1, null);
            panelCenter.add(jPanel1, null);
            this.getContentPane().add(jPanel2, BorderLayout.NORTH);
             config = new FrmConfigure();


          int size = listModel.getSize();

          if (size == 0) //No Machine, disable send button .
            btnSend.setEnabled(false);

}

void fileExit_ActionPerformed(ActionEvent e) {
    System.exit(0);
}

void configSettings_ActionPerformed(ActionEvent e) {
```

```
        statusBar.setText("Setting Configuration");
        config.setVisible(true);
    }

    private void btnSend_actionPerformed(ActionEvent e)
    {

          String node;
          String thisMachine;

          node = (String)jList1.getSelectedValue();
          statusBar.setText("Agent is going to: "+ " \t" + node);

          thisMachine = FrmConfigure.myMachine;

          agent.setNextMachine(node);

          agent.run();

    }
}
```

Machine Configuration Class

```
package agent2agenttrust;

import java.net.URL;
import java.sql.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class FrmConfigure extends JFrame
{
  BorderLayout borderLayout1 = new BorderLayout();
  JPanel jPanel1 = new JPanel();
  JLabel jLabel1 = new JLabel();
  JTextField txtThisMachine = new JTextField();
  JButton btnSave = new JButton();
  JSeparator jSeparator1 = new JSeparator();
  JTextArea jTextArea1 = new JTextArea();
  JLabel jLabel2 = new JLabel();
  JLabel jLabel3 = new JLabel();
  JTextField txtAddRemove = new JTextField();
  JButton btnAdd = new JButton();
  JButton btnRemove = new JButton();
  JButton btnBackMain = new JButton();
  JButton btnRefresh = new JButton();

  public static String myMachine ="";
  public static String myMachineName;

  public FrmConfigure() {
    super( " Configure panel");

    try  {
      jbInit();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
```

```
Connection dbConn = null ;

   try
   {

       Driver d = (Driver)Class.forName
       ("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();

       String URL = "jdbc:odbc:" + "MyDbs2";

       dbConn = DriverManager.getConnection( URL,"nouser","nopassword");

       Statement stmt;

       Statement stmt2;

       ResultSet rs;
       ResultSet rs2;

       stmt = dbConn.createStatement();

       stmt2 = dbConn.createStatement();

       rs = stmt.executeQuery ("SELECT * FROM MachineList");

       rs2 = stmt2.executeQuery("SELECT thisMachine FROM LocalMachine");

        while (rs.next())
       jTextArea1.append(rs.getString ("MachineName") + "\n");

       while (rs2.next())
       txtThisMachine.setText(rs2.getString("thisMachine"));

       dbConn.close();

     }
      catch (Exception e)
      {

        JOptionPane.showMessageDialog(this,"Problem during database
        operation, program exits", "Program
        error",JOptionPane.ERROR_MESSAGE);
        e.printStackTrace();
        System.exit(0);
      }

}

 private void jbInit() throws Exception {
 this.getContentPane().setLayout(borderLayout1);
 this.setSize(new Dimension(400, 441));
 jLabel1.setText("This Machine");
 jLabel1.setBounds(new Rectangle(24, 25, 86, 25));
 txtThisMachine.setBounds(new Rectangle(115, 20, 217, 35));
 btnSave.setText("btnSave");
 btnSave.setLabel("Save Name");
 btnSave.setBounds(new Rectangle(148, 64, 125, 35));
 btnSave.addActionListener(new ActionListener() {

   public void actionPerformed(ActionEvent e) {
     btnSave_actionPerformed(e);
   }
```

```
    });
    jSeparator1.setBounds(new Rectangle(25, 108, 341, 18));
    jTextArea1.setEditable(false);
    jTextArea1.setFont(new Font("Dialog", 1, 12));
    jTextArea1.setBounds(new Rectangle(22, 152, 103, 179));
    jLabel2.setText("Machine List");
    jLabel2.setBounds(new Rectangle(25, 133, 85, 17));
    jLabel3.setText("Add / Remove  Machine");
    jLabel3.setBounds(new Rectangle(188, 154, 161, 25));
    txtAddRemove.setBounds(new Rectangle(173, 179, 159, 28));
    btnAdd.setText("btnAdd");
    btnAdd.setLabel("Add");
    btnAdd.setBounds(new Rectangle(165, 218, 87, 31));
    btnAdd.addActionListener(new ActionListener() {

      public void actionPerformed(ActionEvent e) {
        btnAdd_actionPerformed(e);
      }
    });
    btnRemove.setText("Remove");
    btnRemove.setBounds(new Rectangle(257, 216, 82, 33));
    btnRemove.addActionListener(new ActionListener() {

      public void actionPerformed(ActionEvent e) {
        btnRemove_actionPerformed(e);
      }
    });
    btnBackMain.setText("Close Window");
    btnBackMain.setBounds(new Rectangle(225, 285, 123, 37));
    btnBackMain.addActionListener(new ActionListener() {

      public void actionPerformed(ActionEvent e) {
        btnBackMain_actionPerformed(e);
      }
    });
    btnRefresh.setText("Refresh");
    btnRefresh.setLabel("Refresh List");
    btnRefresh.setBounds(new Rectangle(20, 339, 117, 31));
    btnRefresh.addActionListener(new ActionListener() {

      public void actionPerformed(ActionEvent e) {
        btnRefresh_actionPerformed(e);
      }
    });
    jPanel1.setLayout(null);
    this.setTitle("Agent2Agent Trust Management");
    this.getContentPane().add(jPanel1, BorderLayout.CENTER);
    jPanel1.add(jLabel1, null);
    jPanel1.add(txtThisMachine, null);
    jPanel1.add(btnSave, null);
    jPanel1.add(jSeparator1, null);
    jPanel1.add(jTextArea1, null);
    jPanel1.add(jLabel2, null);
    jPanel1.add(jLabel3, null);
    jPanel1.add(txtAddRemove, null);
    jPanel1.add(btnAdd, null);
    jPanel1.add(btnRemove, null);
    jPanel1.add(btnBackMain, null);
    jPanel1.add(btnRefresh, null);
  }

  void btnSave_actionPerformed(ActionEvent e) {
```

```
  myMachineName = txtThisMachine.getText();

  Connection dbConn = null ;

  try
  {

      Driver d =
      (Driver)Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();

      String URL = "jdbc:odbc:" + "MyDbs2";

      dbConn = DriverManager.getConnection( URL,"nouser","nopassword");

      Statement stmt;

      stmt = dbConn.createStatement();

     stmt.executeUpdate("UPDATE LocalMachine SET thisMachine = '" +
      myMachineName +"'");

      String myMsg;
      myMsg = "This Machine Name updated" ;

      JOptionPane.showMessageDialog(this,
      myMsg,"Informtion",JOptionPane.INFORMATION_MESSAGE);

      dbConn.close();
 }
  catch (Exception ix)
  {
    JOptionPane.showMessageDialog(this,"Error Encountered,the program will
    close","Program error", JOptionPane.ERROR_MESSAGE);

    ix.printStackTrace();
    System.exit(0);
  }
      myMachineName

}

void btnAdd_actionPerformed(ActionEvent e) {

    String myMachine;

    myMachine = txtAddRemove.getText();

    Connection dbConn = null ;

    try
    {

        Driver d =(Driver)Class.forName
        ("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();

        String URL = "jdbc:odbc:" + "MyDbs2";

        dbConn = DriverManager.getConnection( URL,"nouser","nopassword");

        Statement stmt;

        stmt = dbConn.createStatement();
```

```
            stmt.executeUpdate("INSERT INTO MachineList(MachineName)VALUES ('"+
            myMachine+ "')");

            String myMsg;
            myMsg = myMachine + " added" ;
            JOptionPane.showMessageDialog(this,
            myMsg,"Informtion",JOptionPane.INFORMATION_MESSAGE);

            dbConn.close();

          }
           catch (Exception ix)
           {

            JOptionPane.showMessageDialog(this,"Error Encountered, the program
            will close","Program error", JOptionPane.ERROR_MESSAGE);

            ix.printStackTrace();
            System.exit(0);
          }

          //end database

}


 void btnRemove_actionPerformed(ActionEvent e) {

      String myMachine;

      myMachine = txtAddRemove.getText();

      Connection dbConn = null ;


      try
      {


         Driver d = (Driver)Class.forName
         ("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();

         String URL = "jdbc:odbc:" + "MyDbs2";

         dbConn = DriverManager.getConnection( URL,"nouser","nopassword");

         Statement stmt;
         stmt = dbConn.createStatement();

         stmt.executeUpdate("DELETE FROM MachineList WHERE MachineName= '" +
         myMachine +"'" );


         String myMsg;

         myMsg = myMachine + "   removed";

         JOptionPane.showMessageDialog(this,myMsg,"Information",
         JOptionPane.INFORMATION_MESSAGE);
         dbConn.close();

       }
       catch (Exception ix)
```

```
        {

           JOptionPane.showMessageDialog(this,"Unspecified Error Encountered,
           program terminates","Error",JOptionPane.ERROR_MESSAGE);
           ix.printStackTrace();

           System.exit(0);
         }
         //end database
  }

  void btnRefresh_actionPerformed(ActionEvent e) {

    jTextArea1.setText(" ");

    // database

    Connection dbConn = null ;

    try
    {
       Driver d =
       (Driver)Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();

       String URL = "jdbc:odbc:" + "MyDbs2";

       dbConn = DriverManager.getConnection( URL,"nouser","nopassword");

       Statement stmt;
       ResultSet rs;

       stmt = dbConn.createStatement();
       rs = stmt.executeQuery ("SELECT * FROM MachineList");

       while (rs.next())
       jTextArea1.append(rs.getString ("MachineName") + "\n");

       dbConn.close();

     }
      catch (Exception px)
      {

         JOptionPane.showMessageDialog(this, "Unspecified Error Encountered,
         program terminates ","Error",JOptionPane.ERROR_MESSAGE);
         px.printStackTrace();
         System.exit(0);
      }
         //end database
  }

  void btnBackMain_actionPerformed(ActionEvent e) {
         this.setVisible(false);
  }
}
```

REFERENCES

1. Alfalayleh, M. and Brankovic, L. (1998). An Overview of Security Issues and Techniques in Mobile Agents, In 8th IFIP TC-6 TC-11 Conference on Communications and Multimedia Security, The Beech Hill Hotel, England.

2. Amazon Site. (http://www.amazon.com) World Wide Web. May 13, 2006

3. Blaze, M. Feigenbaum, J. and Lacy, J. (1996). Decentralized Trust Management, Proceedings of the 1996 IEEE Symposium on Security and Privacy, Washington, DC, USA, Page: 164, ISBN:0-8186-7417-2

4. Bonatti, P. Duma, C. Olmedilla, D. and Shahmehri, N. (2005). An integration of Reputation-based and Policy-based Trust Management. In Semantic Web Policy Workshop in Conjunction with 4th International Semantic Web Conference, Galway, Ireland, Nov

5. Cohen, P.R. and Levesque, H.J. (1990). Intention is choice with commitment. Artificial Intelligence, 42(2-3): 213-261

6. Conte, R. and Paolucci, M. (2002). *Reputation in artificial societies*. Kluwer Academic Publishers.

7. DeGroot, M. H. and Schervish, M. J. (2002). *Probability and statistics*. Place: Addison-Wesley.

8. Durfee, E.H. (1999). Practically coordinating. AI Magazine, 20. (1) 99-116.

9. eBay Site. (http://www.ebay.com) World Wide Web. June 9, 2006

10. Grandison, T. and Sloman, M. (2000). A survey of trust in internet applications. *IEEE Communications Surveys & Tutorials*, *3*(4).

11. Huynh, T.D. Jennings, N.R. and Shadbolt, N.R. (2004). FIRE: An integrated Trust and Reputation Model for Open Multi-Agent Systems, Proceedings of the 16th European Conference of Artificial Intelligence(ECAI) Volume 13 , Issue 2, Pages: 119 - 154 , Hingham, MA, USA

12. Jennings, N.R. (1993). Communications and conventions: The foundation of coordination in multi-agent systems. *The knowledge Engineering Review*, 8(3): 223-250

13. Jennings, N.R. Faratin, P. Lomuscio, A.R. Parsons, S. Sierra, C. and Wooldridge, M. (2001). Automated negotiation: prospects, methods and challenges. *International Journal of Group Decision and Negotiation*, 10 (2), pages: 199-215

14. Jurca, R. and Faltings, B. (2003a). An Incentive Compatible Reputation Mechanism. In Proceedings of IEEE Conference on E-Commerce, Newport Beach, CA, USA

15. Jurca, R. and Faltings, B. (2003b). Towards incentive-compatible reputation management. In R. Falcone, S. Barber, L. Korba, and M. Singh, (Eds.), *Trust, reputation and security: theories and practice.* Vol. 2631 of Lecture Notes in AI (pp. 138–147). Springer-Verlag, Berlin, Heidelberg.

16. Kraus, S. (2001). Strategic Negotiation in Multi-Agent Environments. Cambridge, MA: MIT Press.

17. Kritzinger, F. Truter, D. and McGregor, K. (2003). A secure End-to-End System for M-Commerce, Technical Report CS03-24-00, Department of Computer Science, University of Cape Town, October 12.

18. Maarof, M.A. and Krishna, K. (2002). "An Hybrid Trust Management Model For MAS Based Trading Society", 3rd International Symposium on Multi-Agent Systems, Large Complex Systems, and E-Businesses (MALCEB'2002), Erfurt/Thuringia, Germany, 8-10 October 2002.

19. Ping, Y. Yan, H. Yafei, H. Yiping, Z. and Shiyong, Z. (2004) Securing ad hoc networks through mobile agent, Proceedings of the 3rd International Conference on Information Security, ACM International Conference Proceeding Series; vol.85, Shanghai, China, pages: 125-129.

20. Pynadath, D. and Tambe, M. (2002). Multiagent Teamwork: Analysing key teamwork theories and models. In C. Castelfranchi and L. Johnson, Editors, Proceedings of the first International Joint Conference on Autonomous Agents and Multi-Agent Systems, Volume 2, pages 873-880.

21. Ramchurn, S.D. Huynh, D. and Jennings, N.R. (2004). Trust in Multi-Agent Systems. In Knowledge Engineering Review, 19(1) pp.1-25

22. Resnick, P. and Zeckhauser, R. (2002). Trust among strangers in internet transactions: Empirical analysis of eBay's reputation system. InM. R. Baye, (Ed.), *The economics of the internet and e-commerce.* Vol. 11 of Advances in Applied Microeconomics. Elsevier Science.

23. Rosenschein, J. and Zlotkin, G. (1994). Rules of Encounter: Designing Conventions for Automated Negotiation among Computers.Cambridge MA: MIT Press.

24. Sabater, J. (2003). *Trust and Reputation for Agent Societies*. PhD thesis, Universitat Autònoma de Barcelona.
Avilable: http://www.iiia.csic.es/~jsabater/Documents/Thesis.pdf

25. Sabater, J. and Sierra, C. (2001).REGRET: A reputation model for gregarious societies. In Proceedings of the fifth international conference on Autonomous agents (pp. 194 - 195). Montreal, Quebec, Canada. ACM Press

26. Salton, G. and McGill, M. (1983). *An introduction to modern information retrieval.* New York: McGraw-Hill.

27. Teacy, W.T.L. Patel, J. Jennings, N. R. and Luck, M. (2005). Coping with inaccurate reputation sources: Experimental analysis of a probabilistic trust model. In *proceedings of fourth international joint conference on autonomous agents and multiagent systems*, Netherlands: Utrecht, pp. 997–100.

28. Winsborough, W. Seamons, K. and Jones, V. (2002). *Automated Trust Negotiation.* In DARPA Information Survivability Conference and Exposition, Hilton Head, SC, January

29. Yu, B. and Singh, M. P. (2003). Searching social networks. In *Proceedings of the second international joint conference on autonomous agents and MultiAgent systems (AAMAS)* (pp. 65–72). ACM Press.

30. Yu, B. and Singh, M.P. (2002). An evidential model of distributed reputation management. In *Proceedings of first international joint conference on autonomous agents and multi-agent systems.* Vol. 1. (pp. 294–301). ACM Press.

31. Zacharia, G. and Maes, P. (2000). Trust management through reputation mechanisms. *Applied Artificial Intelligence, 14*(9), 881–908.

32. Zuma, S. M. and Adigun, M.O. (2006). CACIP: a pattern for interfacing components in a context-aware mobile environment, International Association Of Science And Technology For Development, Proceedings of the 17th IASTED international conference on Modelling and simulation, Montreal, QC , Canada, Pages: 416 - 423 , ISBN ~ ISSN:1021-8181 , 0-88986-592-2, May.