THE UNIVERSITY OF NAIROBI

COLLEGE OF BIOLOGICAL AND PHYSICAL SCIENCES

SCHOOL OF COMPUTING AND INFORMATICS

PROTECTING MOBILE AGENTS FROM MALICIOUS HOSTS IN A DISTRIBUTED NETWORK

BY NGEREKI ANTHONY MBUGUA

P53/65839/2013

SUPERVISOR: DR. ANDREW M. KAHONGE

A RESEARCH WITH PROTOTYPE PROJECT REPORT SUBMITTED TO THE SCHOOL OF COMPUTING AND INFORMATICS IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF AN MSC. IN DISTRIBUTED COMPUTING

JANUARY, 2015

Declaration

This research project report is my original work and has not been presented in any other academic institution for similar reasons.

Signed:

……. **……..**

Anthony M. Ngereki **Date**

This research project report has been submitted for examination with my approval as University Supervisor.

Signed:

…….. **……**

Dr. Andrew M. Kahonge **Date**

Dedication

To my mum Mary, my wife Rachel and my lovely daughter, Zaneta.

Acknowledgement

I wish to express my gratitude first and foremost to the almighty God. Father, your love, grace and favor has brought me this far.

To my supervisor, Dr. Andrew M. Kahonge. Your insights, exemplary guidance, valuable feedback and constant encouragement throughout the duration of the project have been outstanding. I have drawn a lot of confidence and learnt more from your patience and humility. Am so grateful.

To my employer, Chuka University and in particular, the Vice Chancellor, Prof. E. Njoka, his Deputy, Prof. S. M Kagwanja and the entire Staff Development and Education Committee members for granting me this opportunity.

To my friend Eric Gitonga and Miriam Njoki, yours was more than support, but true friendship. My workmates Kamweru, Nish, Dennis, Osero, Dr. Mukuthuria, Grace, Naomi, I cant mention you all but thanks for your encouragement and support. Special gratitude to Okello Nelson whose advice and guidance helped me complete this project.

To my course mates. It was tough but with your positive criticism, support and true comrade spirit, we made it. Special gratitude to Kariuki and Waweru, you remind me of the slogan; "you'll never walk alone".

To my parents, Mr. and Mrs. Ngereki, for giving me the gift of education. I cannot ask for more.

Finally, not least though, to my wife Rachel, for being patient, understanding and supportive in my time away, studying, you are more than I could ever seek for in a wife. And to our daughter Zaneta, the joy, fulfillment and completion that you bring to the family fuels our desire to do more than just succeed, I so love you.

Abstract

Distributed applications provide challenging environment in today's advancing technological world. To enhance the aspects of better performance and efficiency in real scenario, mobile agent's concept has been brought forward. Mobile agents (MAs) are autonomous computing entities that have the capability of moving (migrating) from one host to another and resuming execution in the new host. MAs are an extension of the mobile object concept and allow the movement of an agent's code, data and state. Of concern however are the security threats that this exciting paradigm is associated with.

We have used the Tropos methodology to design a security framework for mobile agent systems. We further demonstrate this security solution by use of a credit bureau use case. In our design, the security mechanism protects tasks, sub-tasks, goals and soft goals of each agent from other agents. Only an authenticated and authorized agent can access tasks or goals of another agent. This security hierarchy uses a multi-faceted approach to protect mobile agents and must be incorporated from the design stage of agent systems. The security supervisor assumes the dual role of authentication and authorization. The status monitoring agent, on the other hand, monitors the status of each agent in the environment.

Our results showed that a multi-agent system with no security at all factored into its design will always be vulnerable. This is due to the fact that multi-agent systems are inherently loosely coupled. A secure multi-agent system is one that has both a good monitoring and security supervision framework. These two frameworks complement each other. Where the security might fail, the monitoring framework would report an exception.

Definition of Terms

Mobile Agent - Mobile agents (MAs) are autonomous computing entities that have the capability of moving (migrating) from one host to another and resuming execution in the new host.

Agent Platform – An agent platform provides the computational environment in which an agent operates.

Mobility - Is the core property in a mobile agent concept whereby the agent has the ability to migrate or transport itself from one node to another within the same environment or from node to another node in a different environment autonomously.

Ragged - MAs should be able to deal with errors when encountered and during their (errors) occurrence.

Actors – Interchangeably used with the term agent. An actor is an autonomous software component.

Financial Regulator – An institution mandated by the government to regulate finance industry.

Credit Reference Bureau - An institution, licensed by a government, to offer credit reference services like Credit Scoring, for instance.

Abbreviations and Acronyms

MA	Mobile Agent
MAS	Mobile Agent System
TLS	Transport Layer Security
ACL	Agent Communication Language
OHA	One Hop Agent
MHA	Multiple Hop Agent
PMM	Partial Mobility Mechanism
CRB	Credit Regulatory Bureau
API	Application Programming Interface
SSL	Secure Sockets Layer
FTP	File Transfer Protocol

Table of Contents

List of Figures

CHAPTER ONE

1.0 Introduction

1.1 Background of the Study

The mobile agent paradigm is a shift in the evolution of computing and the distributed computing environment in particular. Previous works show that agent concepts and mobile agents have become important building blocks of the architecture of new networks, systems and services. Mobile agents (MAs) are autonomous computing entities that have the capability of moving (migrating) from one host to another and resuming execution in the new host. MAs are an extension of the mobile object concept and allow the movement of an agent's code, data and state. Although mobility is the core property of MAs, they should also be autonomous, interactive, adaptive, proactive, intelligent, coordinative, learning, cooperative, ragged and ability to act as proxy.

The nature of mobile agents leads to many benefits. By allowing users to package and migrate their operations to be carried out locally, computation is moved to the data rather than the data to the computation thus reducing the network load. Another benefit is that their mobility and adaptive properties enables MAs to respond to real-time systems. These benefits along with conservation of bandwidth have seen MAs fronted as a replacement to the client/ server model in the building of computer networks. However, security for MAs remains a significant obstacle towards their full adoption.

Security is one of the major important issues that should be carefully planned when developing a MA model. The model must protect all parties in the system such as: MAs, Hosts, the mobility and communications. This area has taken a wide range of researchers' attention. And it could be classified as one of the biggest challenges. The importance of the security comes from the nature of the MAs itself; that contains distributed entities any gape of security will affect overall the system. The MA needs protection against other malicious MAs and hosts. Mobile agent systems have not only incorporated security issues that have often incurred in conventional distributed systems, it also possesses some new security threats. Security threats in mobile agent systems are classified into four main categories: agent-to-platform, platform-to-agent, agent-to-agent, and others-to-agent platform.

1.2 Statement of the Problem

One of the major concerns towards the reliability of mobile agent is its security/protection from various types of threats like authentication & authorization of the user, malware threats, communication of private information, denial of service attack, logic bomb or event-triggered attacks, compound attacks, security threats from malicious hosts and malicious logic (Mishra and Singh, 2014).

Security is the biggest concern which darkens the advantageous side of mobile agent infrastructure (Srivastava and Nandi, 2014). The ability of a mobile agent to migrate from it's home platform to another execution environment exposes to various attacks (eavesdropping, network sniffing, malicious execution environment, etc.).

Ebietomere and Ekuobase (2014) posit that ensuring security for a mobile agent against a malicious host appears to be the most serious of the fundamental security issues of mobile agents. They further state that no solution has been found for the problem yet, though effort has been made by researchers to solve the problem.

Although the mobile agent paradigm promises a great revolution in distributed networks and applications, its' adoption is still low largely because of the security risks that introduces.

1.3 Research Objectives

1. To propose a multi-agent system security mechanism.
2. Design and build a prototype that implements the proposed security mechanism.
3. Evaluate the prototype to establish level of security threats mitigation in the prototype.
4. Demonstrate secure multi-agent system operation using a credit bureau use case.

1.4 Scope of the Research

Although, there a number of models that can be used to describe agent systems, a simple model consisting of two components: the agent and the agent platform is sufficient to discuss security in mobile agents. From this model, security threats to mobile agents can be categorised as: agent to agent, agent to platform, platform to agent and other to agent platform.

This study seeks to propose a solution to security threats associated with agent platform to mobile agent. Such threats are expounded on in section 2.6.

1.5 Justification/ Benefits of the Research

The Mobile Agent (MA) technology is gaining importance in the distributed management of networks and services for heterogeneous environments. MA-based management systems could represent an interesting alternative to traditional tools built upon the client/server model. The acceptance of MA solutions is limited by the request for security. Without security, applications cannot suit global untrusted environments, such as the Internet.

The mobile agent paradigm provides a very important feature in developing high performance, decentralized software applications in distributed environments. As agents (executable code) can travel on the network, data transfer between the communicating parties is drastically reduced, which increases communication performance by utilizing network bandwidth.

It is therefore imperative to develop solutions to the challenges of the MA paradigm especially security threats.

CHAPTER TWO

2.0 Literature Review

2.1 Mobile Agents and state of the art

Ahmed (2013) and Paulino (2002) both define mobile agents as independent objects capable to achieve tasks in heterogeneous networks on behalf of users. The mobile agent paradigm is an extension of the mobile – object concept in which the object (code and data) is moved from one machine to another. Mobile agents (MA's) moves code, data and state from one computing environment to another autonomously. This mobility known as migration is achieved by having the MA saving its state and execution in one host that does not have the resources that the MA needs to execute, the MA then moves to another host in the network that have the required resources and resumes execution from the previously saved state.

The MA's body consists of three parts: first part is a code which represents the behavior or tasks of the MA that will be executed in the hosts. The second part represents the MA's data space which is updated according the execution of the first part. The third part is the execution state that keeps a execution start point in each host. (Ahmed, 2013).

2.2 The Mobile Agent Lifecycle

The MA lifecycle according to the aglet platform consists of creation, cloning, dispatching (migration), retraction, activation, deactivation and disposal. (Aneiba and Rees, 2004).

Creation is the first period where a mobile instance is created and eqipped with the desired parameters so as to achieve its goal. Cloning creates a copy of the original mobile object when the need for another agent with the same features and properties to do the same or other job in conjuction with the original agent arises. Dispatching (migrating) is responsible for moving the agent fro one node to another within the network environment by specifying the destination address(e.g. URL) to the agent. The retraction function is done where the agent's source node requires its agent to be returned to the original host or node. Activation and deactivation are operations done when the mobile agent is required to start or to stop only within certain time of its lifetime. Finally, the dispose operation is done where the agent life comes to the end. Fig. 2.2 explains the above mobile agent operations as suggested by an Aglet system.

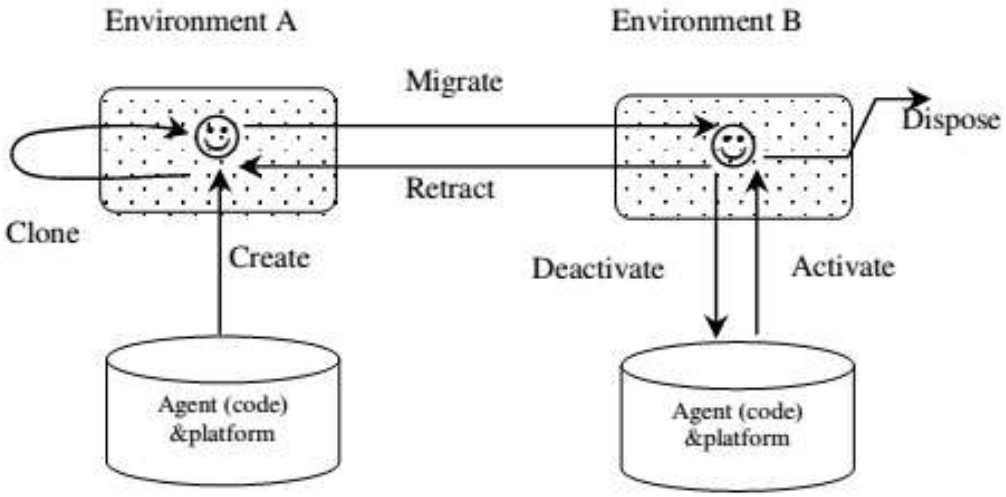

Figure 2.1: Mobile Agent Lifecycle (Aglet perspective)

2.3 Advantages of Mobile Agents

Satoh (2008) outlines various advantages of MA's in the development of various services in smart environments in addition to distributed applications. These include:

- **Asynchronous execution**: After migrating to the destination-side computer, a mobile agent does not have to interact with its source-side computer. Therefore, even when the source can be shut down or the network between the destination and source can be disconnected, the agent can continue processing at the destination. This is useful within unstable communications, including wireless communication, in smart environments.
- **Direct manipulation**: A mobile agent is locally executed on the computer it is visiting. It can directly access and control the equipment for the computer as long as the computer allows it to do so. This is helpful in network management, in particular in detecting and removing device failures. Installing a mobile agent close to a real-time system may prevent delays caused by network congestion.
- **Reduced communication costs**: Distributed computing needs interactions between different computers through a network. The latency and network traffic of interactions

often seriously affect the quality and coordination of two programs running on different computers. As illustrated in Figure 2.3, if one of the programs is a mobile agent, it can migrate to the computer the other is running on and communicate with it locally. Thus, mobile agent technology enables remote communications to operate as local communications.

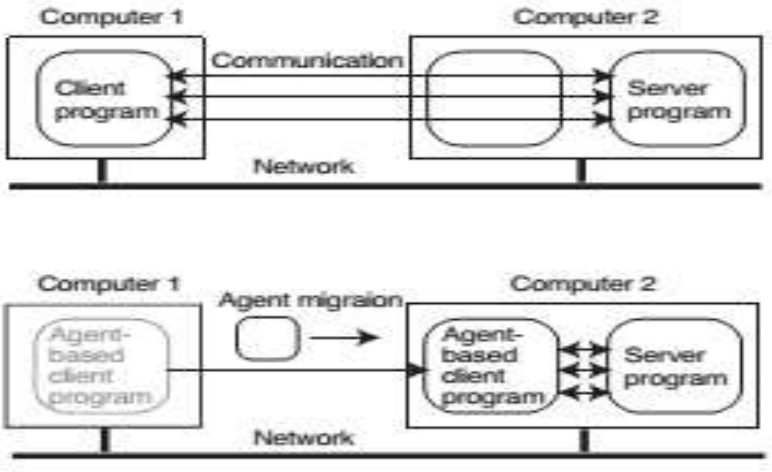

Figure 2.2: Reduced communication with mobile agents

- **Dynamic-deployment of software**: Mobile agents are useful as a mechanism for the deployment of software, because they can decide their destinations and their code and data can be dynamically deployed there, only while they are needed. This is useful in smart environments, because they consist of computers whose computational resources are limited.
- **Easy-development of distributed applications**: Most distributed applications consist of at least two programs, i.e., a client-side program and a server side program and often spare codes for communications, including exceptional handling. However, since a mobile agent itself can carry its information to another computer, we can only write a single program to define distributed computing. A mobile agent program does not have to define communications with other computers. Therefore, we can easily modify standalone programs as mobile agent programs.
- A mobile agent enables dynamic service customization and software deployment because it encapsulates some services or protocols into its mobility entity.

2.4 Characteristics of Mobile Agents

Aneiba and Rees (2004) have higlighted several characteristics that a mobile agent should have. MAs should be *autonomous,* having the ability to act without direct external interferences. In other words, they have some degree of control over their data and states. MAs should be *Interactive* in communicating with the environment and other agents. MAs should be *adaptive*. In other words, they have ability to respond to other agents or theirenvironment.

Mobility is the core property in a mobile agent concept whereby the agent has the ability to migrate or transport itself from one node to another within the same environment or from node to another node in a different environment autonomously. *Proxy*, MAs may act on behalf of someone or for the benefit of some entities (e.g. software systems). In order to act on behalf of others, mobile agents must have at least a minimal degree of autonomy. *Proactive,* MA should be a goal-oriented entity, and take an initiative in responding to an environment.

Intelligent, MAs may have certain degree of intelligence, based on knowledge in order to act efficienciently. *Coordinative,* MAs should be able to perform data transfer activities in sharing with other agents within the givenenvironment. *Learning* refers to a mobile agent's ability in gaining information about the current environment, which will help MAs to modify its behaviour. *Cooperative,* MAs should be able to coordinate with other agents to achieve a common purpose. *Ragged,* MAs should be able to deal with errors when encountered and during their (errors) occurrence.

2.5 Security Threats in Mobile Agents

Pai P, Shinde S.K and Khachane (2012) and Varnamkhasti M.M and Mahmoodi M (2013) observe that threats to security generally fall into three main classes: disclosure of information, denial of service, and corruption of information. Although, there a a number of models that can be used to describe agent systems, a simple model consisting of two components: the agent and the agent platform is sufficient to discuss security in mobile agents. The agent platform provides the computational environment in which an agent operates. The platform from which an agent originates is referred to as the home platform, and normally is the most trusted environment for an agent.

From this model, we can categorise security threats as:

Agent-to-Platform: this category category represents the set of threats in which agents exploit security weaknesses of an agent platform or launch attacks against an agent platform. This set of threats includes masquerading, denial of service and unauthorized access.

Agent-to-Agent: The agent-to-agent category represents the set of threats in which agents exploit security weaknesses of other agents or launch attacks against other agents. This set of threats includes masquerading, unauthorized access, denial of service and repudiation. Many agent platform components are also agents themselves.

Platform-to-Agent: The platform-to-agent category represents the set of threats in which platforms compromise the security of agents. This set of threats includes masquerading, denial of service, eavesdropping, and alteration.

Other-to-Agent Platform: The other-to-agent platform category represents the set of threats in which external entities, including agents and agent platforms, threaten the security of an agent platform. This set of threats includes masquerading, denial of service, unauthorized access, and copy and replay.

2.6 Security threats on Mobile Agents by Mobile Agent Platforms

The mobility factor of mobile agents poses the greatest of exposure of a mobile agent to threats. Mobility is regarded as either weak or strong. Strong migration means that the mobile agent can carry itself, its data and its state while weak migration means that mobile agent can carry only itself and its data (e.g. mobile object). In case of strong mobility of mobile agent all its code, data and state are exposed to the mobile agent platform in which it migrates for execution of operation. Because of this, a mobile agent faces more severe security risks. Following are possible attacks by malicious platforms (Dadhich et al., 2010)

2.6.1 Leak out/ modify mobile agent's code

Since the mobile agent's code has to be read by a guest platform, so this malicious platform can read and remember instructions going to be executed to infer the rest of the program based on that knowledge .By this process, platform knows the strategy and purpose of mobile agents. If mobile agents are generated from standard building libraries , the malicious platform knows a complete picture of mobile agent's behavior and it finds out the physical

address and can access its code memory to modify its code either directly or by insertion of virus. It can even change code temporarily , execute it and finally resuming original code before the mobile agent leaves.

2.6.2 Leak out/ modify mobile agent's data

A lot of data is security sensitive like security keys, electronic cash, e.t.c. If the malicious platform get to know the original location of data it can modify the data in accordance with the semantics of data. Above tasks can lead to severe consequences. Even if data is not sensitive, malicious platform can attack on normal data like traveling data of person and leaking it to somebody.

2.6.3 Leak out/ modify mobile agent's execution flow

By knowing the mobile agents physical location of program counter, mobile agent's code and data the malicious platform can predict what will be set of instructions to be executed next and deduce the state of that mobile agent. By help of this process, it can change the execution flow according to its will to achieve its goal. It can even modify mobile agent's execution to deliberately execute agent's code in wrong way.

2.6.4 Denial of Service(DoS)

This attack causes mobile agent to miss some good chances if agent can finish its execution on that platform in time and travel to some other platform. DoS causes not to execute the mobile agent migration and put it in waiting list carrying delays.

2.6.5 Masquerading

Here malicious platform pretends as if it is the platform on which mobile agent has to migrate and finally becomes home platform where mobile agent returns. By this mechanism, it can get secrets of mobile agents by masquerading and even hurts the reputation of the original platform.

2.6.6 Leak out/ Modify the interaction between a mobile agent and other parties

Here malicious platform eavesdrop on the interaction between a mobile agent and other parties like another agent or other platforms. This leads to extraction of secret information about mobile agent and third party. It can even alternate the contents of interaction and expose

itself as part of interaction and direct the interaction to another unexpected third party. By this way, it can perform attacks to both mobile agent and third party.

2.6 Related Work

The executing host in mobile agent systems has complete control over executing programs. This makes agent protection difficult. Several approaches have been proposed to solve this shortcoming. We survey some approaches some of them,

Shrivastava & Mehta, (2012) propose an algorithm to protect agents. In their algorithm, there's a monitoring agent and a dummy agent with the same script but with dummmy data. The original agent sends the monitoring agent and dummy agent to the the next node in the network to chech its behaviour. If the node is suspicious, the monitoring agent sends an alert acknowledgement and an OK acknowledgement if otherwise. Though this algorithm protects the agent, it still adds to some overhead in the network due to the cteation of the dummy agent and its data.

Ahmed, (2013) proposes partial mobility mechanism (PMM) to protect mobile agents integrity and privacy against malicious hosts. In PMM the MA has two types: the first one is an One_Hop_Agent (OHA) which can visit only one host. The second is a Multi-Hop-Agent (MHA) which can visit multiple hosts. The MHA can contain multiple of OHAs. The main idea behind PMM is to allow to the One-Hop-Agent to visit untrusted hosts only. So, the MA will not visit any host that is classified as untrusted host. In PMM, all hosts will be visit by MAs are classified in two categories, trusted and untrusted hosts. The problem with this method is that untrusted host has to be known prior to the design of the system. This may be particularly difficult in a distributed environment.

The Ajanta mechanism: This mechanism proposes three approaches for protecting the MA. The first is to allow the programmer to define parts of the MA's state as Read-Only and if any modification occurs to these parts, the MA's user can detect using the digital signature mechanism. The second approach is let the MA creates append-only data states container where the data stored in this container can not be deleted or altered without detection by MA's user. The third approach is to let programmers to define data states to specific hosts and no other hosts can deal with these data states. These mechanisms use the encryption, the decryption and the digital signature.

A Secure Mobile Agents Platform: By using access control and authentication, this mechanism protects the MAs. The host controls all the resources available on it. Each MA defines its own control policy for other MAs by using an Interface Definition Language (IDL).

KeyLets mechanism: This mechanism based on partitioning a MA as units according to the task type. By using secret keys, it encrypts each unit to protect them. The distribution of keys to different hosts is done through the execution of specific type of a MA that is termed a Keylet. The disadvantages of this approach: Propagation requires a third party code producer that can supply the MA by a template the MA's owner. Also, a large number of transactions related to the keylet and a host may not be willing to support the increased of computation. Moreover, key revocation is not good in quality. In addition, it requires a complicated mechanism to categorize tasks of the MA. Also, this mechanism does not protect the MA code completely.

2.6 Agent Hierarchy and Inter-Agent Migration

We used an approach where a hierarchy of agents exists. The parent agent supervises and keeps track of all children in its hierarchy. If the children fork other children in the course of their execution, then the child is in supervision of the children forked.

This hierarchy of command and control ensures that any activity by the agent is tracked as well as any activity or intent to tamper with an agent. If this happens, then the parent agent has a decision to make whether to kill the agent or deploy in another host

The model is achievable through introduction of the following concepts as borrowed from (Satoh, 2008.)

1. Agent Hierarchy: Each mobile agent can be contained within one mobile agent.
2. Inter-agent Migration: Each mobile agent can migrate between mobile agents as a whole with all its inner agents.

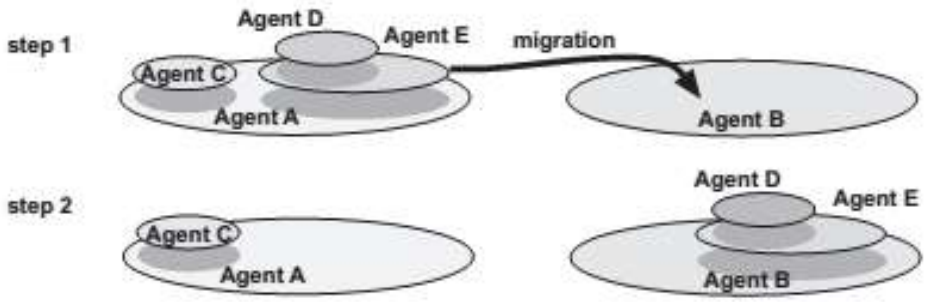

Figure 2.3: Agent hierarchy and Inter-agent migration

The agent hierarchy is given as a tree structure in which each node contains a mobile agent and its attributes. The runtime system is assumed to be at the root node of the agent hierarchy. Agent migration in an agent hierarchy is performed just as a transformation of the tree structure of the hierarchy. In the runtime system, each agent has direct control of its inner agent. That is, a container agent can instruct its embedded agents to move to other agents or computers, serialize and destroy them. In contrast, each agent has no direct control over its container agent. Instead, each container can offer a collection of service methods which can be accessed by its embedded agents.

CHAPTER THREE

3.0 Research Methodology

3.1 Introduction

In this chapter, we discuss the general methods, tools and instruments that we used to achieve the objectives we set out in chapter one. The methodologies discussed are directly matched with the tasks that we undertook to solve the problem which included; formulation of a security solution for mobile agents, development of a prototype that demonstrates the security solution and use of a test case to verify effectiveness of the security solution.

3.2 Research Design

The nature of the research favored an experimental design. Much of the substantial gain in knowledge in sciences comes from performing experiments (Seltman, 2014). In an experimental design, the researcher interferes with the conventional order of doing a thing by introducing a selected condition or change. Observations or measurements are then planned to illuminate the effect of the changes. Experimental designs help infer about causes or relationships on than just simply describe. An experiment may be either exploratory or confirmatory; in our case it was exploratory.

Seltman, (2014) describes scientific learning as an iterative process in which experimentation is one of the steps. Analysis and interpretation of information got from experiments leads to possible modification of the current state of knowledge. This process is shown in the figure below.

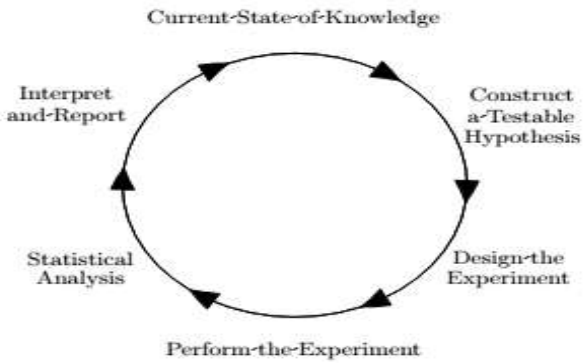

Figure 3.1: The circular process of scientific learning

3.3 System Development Methodology

We used the Tropos methodology in developing the system prototype. Tropos is an agent-oriented software engineering (AOSE) methodology that covers the whole software development process. Tropos is based on two key ideas. First, the notion of agent and all related mentalistic notions (for instance goals and plans) are used in all phases of software development, from early analysis down to the actual implementation. Second, Tropos also covers the very early phases of requirements analysis, thus allowing for a deeper understanding of the environment where the software must operate, and of the kind of interactions that should occur between software and human agents.

Tropos spans four phases:

- **Early requirements**, concerned with the understanding of a problem by studying an organizational setting; the output of this phase is an organizational model which includes relevant actors, their respective goals and their inter-dependencies. Early requirements include two main diagrams: the actor diagram and the goal diagram. The latter is a refinement of the former with emphasis on the goals of a single actor.
- **Late requirements**, where the system-to-be is described within its operational environment, along with relevant functions and qualities. The system-to-be is represented as one actor which has a number of dependencies with the other actors of the organization. These dependencies define the system's functional and non-functional requirements.
- **Architectural design**, where the system’s global architecture is defined in terms of subsystems, interconnected through data, control and other dependencies. This phase is articulated in three steps:
    - Definition of the overall architecture.
    - Identification of the capabilities the actors require to fulfill their goals and plans.
    - Definition of a set of agent types and assignment to each of them one or more capabilities.

- **Detailed design**, where behavior of each architectural component is defined in further detail. Each agent is specified at the micro-level. Agents' goals, beliefs and capabilities are specified in detail, along with the interaction between them

3.4 Prototyping

Prototyping is the process of building a model of a system. A prototype is designed to test a new design to enhance precision by system analysts and users. Prototyping serves to provide specifications for a real, working system rather than a theoretical one. The choice of prototyping in this research was informed by several factors, key among them being the limited time available for the completion of the project. By prototyping, we were able to simply demonstrate functions relevant to our research using available software tools.

We used a proof of principle (proof of concept) prototype to test the main aspect of our intended design without attempting to exactly simulate the actual implementation. Proof of concept prototypes are can be used to prove out a potential design approach to demonstrate its feasibility. An advantage of proof of concept prototypes is that they are usual small and may or may not be complete.

CHAPTER FOUR

4.0 Technical Design, Analysis and Implementation

4.1 Introduction

In this chapter, we describe the system prototype in detail. We discuss the security mechanisms employed in the prototype. We then, describe the CRB use case in detail using flowcharts, use case diagrams and activity diagrams. We have relied heavily on open source tools to achieve this objective. Open source tools are distributed for free and are suitable in academic research because the researcher can modify as per system requirements without the need for user licenses which are in most cases expensive to acquire.

4.1.1 A Multi-Faceted Mobile Agent Security Mechanism

Most mobile agent security mechanisms proposed only detect rather than protect (Jansen and Karygiannis, 2003). Literature reveals that a single approach cannot protect a mobile agent from all the security challenges that dog this exciting paradigm (Shrivastava and Mehta, 2012). We therefore propose a multi-faceted approach to dealing with security threats in mobile agent systems. Such a solution must be implemented right from the design stage to implementation if a mobile agent system has to be truly secure. The use of the Tropos methodology helps us achieve this. This approach effectively deals with all known threats by malicious hosts to mobile agent systems. Sections 4.1.2 to 4.1.4 describe the algorithms that we have embedded in our mobile agent code to mitigate against threats posed by a malicious host.

4.1.2 Protection Against Blocking and Denial of Service

Using time to live (TTL) and heart beat mechanisms, a mobile agent can be protected against blocking or denial of the service by a malicious host. The following steps describe how the mechanism detects and protects against blocking or denial of service

While not Successful;

1. Parent Agent
    a) Creates a new agent (Either in the present or alternative host)
    b) The created Agent:

a. Checks resource availability (If enough resources, Acknowledge ability to execute else dies)

c) Transfer data to the created agent

d) Sets TTL and Starts timer

2. While child Agent Not Done;
    a) Sends heartbeat to parent
3. While Timer < TTL;
    a) Waits for result from child agent
4. If Timer >= TTL;
    a) Attempt to destroy child agent
5. Else If Not Successful;
    a) Cut off communication from child agent
6. If there is a free agent;
    a) Assign task of child agent to free agent
7. Else
    a) Create another child agent in another host

4.1.3 Protection Against Masquerading

We have achieved this by centralizing the security requirements of any agent to itself. This implies that when an agent desires to migrate, it creates its instance in a new host and kills itself in the current host. It also informs its parent host of this move. This completely abstracts the use of host identity in the migration process making it difficult for a host to masquerade as another. Further to this requirement, a parent agent assigns a child agent ID for each of the children it creates. This ID is not replicable as it uses a one-way hashing algorithm. The parent agent keeps a repository of all IDs of the children it has created and tries to match this with the ID of any agent that tries to communicate with it. If a match cannot be found, communication is denied. Communication between a parent agent and a child is that a child must identify itself when communicating with the parent or any other agent in the network while a parent agent must not necessarily identify itself with its children.

The following steps describe the process of generating an irreplicable child ID:

1. Call a hashing function (We propose a 256 bit SHA1 algorithm)

2. Call a timestamp retrieval function
    - Generate current timestamp (Computed to nanosecond precision)
    - Host time zone
3. Call an agent ID generator function (Concatenates Hash value + Timestamp + Time zone)

This process creates a unique ID for any agent created. The reasoning here is that even in a synchronized environment, it is difficult for a malicious host to precisely fake an ID with precision to the nanosecond.

4.1.4 Protection Against Eavesdropping and Alteration

We use encryption and decryption mechanisms to achieve this. Eavesdropping is only possible in the host environment since in our mechanism, communication between remote hosts is via SSL/ TLS. We further make eavesdropping on an agent's code by using a production ready tool (Scala) in development of our agents. This implies that the agents' code is compiled and not visible to the host environment.

The files and data created during execution is however visible by the host. To counter eavesdropping, we use an encryption algorithm to encrypt the temporary files and data before writing to memory of the host environment and a decryption algorithm whenever we need to read them.

We further protect against alteration by adding a CRC based on our encrypted data and compare it when retrieving data to detect alteration and take corrective actions. The steps below describe how we achieved this:

Writing Temporary Files to Host

1. Encrypt data before writing to host file system.
2. Generate CRC based on data in step 1 above.
3. Write out the data to the file system

Reading Data from Temporary Files

1. Retrieve data.

2. Calculate CRC.
3. Calculate CRCs to see if data is modified
4. If CRC1 is NOT equal to CRC2
    a. Inform the parent
    b. Cease processing

The parent agent then decides whether to transfer task to a free agent or to start the agent in a new host.

4.2 System Description

As mentioned in the previous chapter, we used the Tropos methodology in the development of the prototype. The Tropos methodology spans four phases namely: Early Requirements, Late Requirements, Architectural Design and the Detailed Design Phases.

In line with our objectives as set out in Section 1.3, we chose a use case to demonstrate our secure multi-agent system operation. The use case is a credit bureau which is described in the sections that follow using the Tropos methodology.

4.2.1 Current System Model (Early Requirements Phase)

The current credit bureau system is as described below. The relevant actors (agents), their respective goals and their interdependencies are described.

4.2.1.1 Main Actors

An actor represents:

- A physical or a software agent.
- A role, meant as an abstract characterization of the behavior of one or more agents taken in a specific context.
- A position, i.e., a set of roles generally assumed by a single agent, e.g., Data Analyst.

In our survey of the current system, we identified the following actors:

Financial Institution:
All financial institutions are required by laws of Kenya to submit their credit data to a licensed credit reference bureau. This actor refers to a representative in each financial institution charged with the responsibility of ensuring that data is formatted properly, in

accordance with the industry regulator's template, before it's submitted to a licensed credit reference bureau through the system. Goals of a financial institution actor are:

- Prepare credit information records that comply with the industry regulator's provided document template.
- Submit the credit information records to the credit reference bureau.
- Review processing reports generated by the credit reference bureau and revise the credit information records as needed.
- Resubmit revised credit information records to the credit reference bureau.

Credit Reference Bureau:

This refers to an institution, licensed by a government, to offer credit reference services like Credit Scoring, for instance. It's this institution that reports to the industry regulator the compliance level of each financial institution. In a generic notion, it's the responsibility of a credit reference bureau to report to the regulator, the compliance level of each financial institution. This, it does by generation reports that shows statistics and detailed results of processing data it receives from financial institutions. This actor models the credit reference bureau personnel charged with the responsibility of generating and manually verifying the accuracy and presentation of statistical and detailed processing reports. The personnel also have a responsibility to dispatch the reports to the regulator in the processing workflow. The credit reference bureau has the following goals:

- Process credit information records received from financial institutions by enforcing rules provided by the industry regulator.
- Generate processing summary statistics and detailed reports.
- Send generated summary statistics and detailed reports to financial institutions using in-house portal or e-mail.
- Send generated summary statistics to the industry regulator using a provided online portal or e-mail.

Financial Industry Regulator:

This refers to an institution mandated by a government to regulate the finance industry. The regulator defines rules and templates used by financial institution to submit credit data to the

system. This actor models a regulator personnel charged with the responsibility of going through generated reports and advising relevant stakeholders as needed. The goals of an industry regulator actor are:

- Prepare document submission specification to be used by the financial institution to submit credit information to the credit reference bureau.
- Review reports it receives from credit reference bureau about the compliance level of the financial institution and act accordingly.
- Revise the document submission specification and train the credit reference bureau and the financial institution on its usage.

In our methodology, intentional elements include goals, soft goals, tasks and resources. These can either be internal to an actor, or define a dependency relationship between actors. A goal is a condition or state of affairs in the world that the actor would like to achieve. For example, a credit reference's bureau to codify rules it receives from the industry regulator is modeled as a goal.

A soft goal is typically a non-functional condition, with no clear-cut criteria as to when it's achieved. For example, the fact that all processing should be completed within 24 hours of receiving financial institution data, regardless of volume, is modeled as a soft goal.

A task specifies a course of action that produces the desired effect. A resource, represented is a physical or information entity. For example, a financial institution waits for file processing reports to be availed by the credit reference bureau.

Some intentional elements are internal to each actor. While some are delegated from one actor to another. For example, a financial institution depends on a credit reference bureau to access credit score of a credit applicant. The resource is modeled as a dependency, using dependency links, from financial institution to the credit reference bureau. Dependency links are used to represent inter-actor relationships.

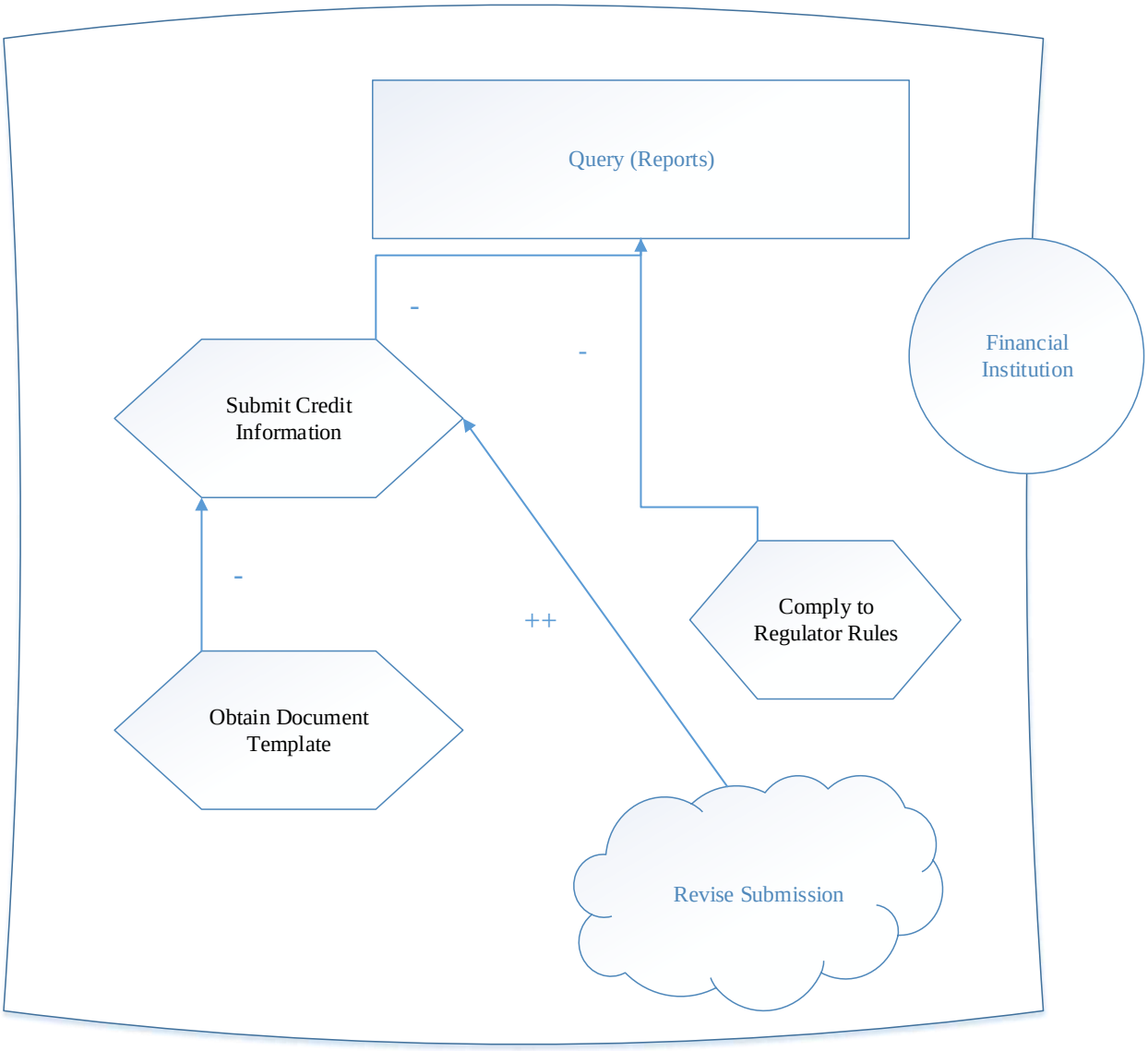

Figure 4.1: High level Tropos model focusing on the Financial Institution

Figure 4.1 zooms into one of the actors of this domain, the financial institution. It shows how the high level intentional elements of the financial institution are refined and operationalized. The refinements and relationships among intentional elements are represented with these intentional links: means-ends, decomposition and contribution links. Each element connected to a goal by a means-ends link is an alternative way to achieve the goal. Decomposition links define a refinement for a task. For example, if a financial institution wants to access credit information for an applicant, they must have submitted their own credit records to a credit

reference bureau and the credit records they submitted must have been conformant to the industry regulator rules.

A contribution link describes the impact that an element has on another. The impact can be positive or negative. Positive impact is represented by a plus sign (+) while a negative impact is represented by a minus sign (-). For example, if the financial institution revises its submission on receiving a processing report from the credit reference bureau, there will be a positive impact on the quality of a resubmission, and hence the compliance level of the financial institution to the industry regulator rules.

4.2.2 Proposed System Model (Late Requirements Phase)

In this stage, we are going to focus on the requirements in the target system, i.e., the system-to-be. The goal of this phase is to provide a set of functional and non-functional requirements of the target system. Contrary to early requirements phase, we are going to model the system itself by introducing it as an actor system and model its dependencies with the other actors of the organization.

Figure 4.2 shows a late requirements Tropos model of the target system. The system is code named SCIPD, for System for Distributed Credit Information Processing and Querying System. The system simplifies the current processing of credit information, as explained in [4.2] by introducing a distributed system that automates much of the processing and communication needed. It also makes it easy to create a Query API. The Query API would be used by all subscribed financial institutions to query credit applicant's credit score or credit history.

Of note in the target system is the security and distributed processing requirements. As shown in [4.2], centralizing processing to credit bureaus' premises may lead to delays in communicating compliance feedback to financial institutions. Also, industry regulator may find it hard to mine trends in compliance by the financial institutions. By adopting a distributed approach, financial institutions have control over reviewing of their compliance level and only commit their credit information to the credit reference bureau when they are satisfied with the compliance level.

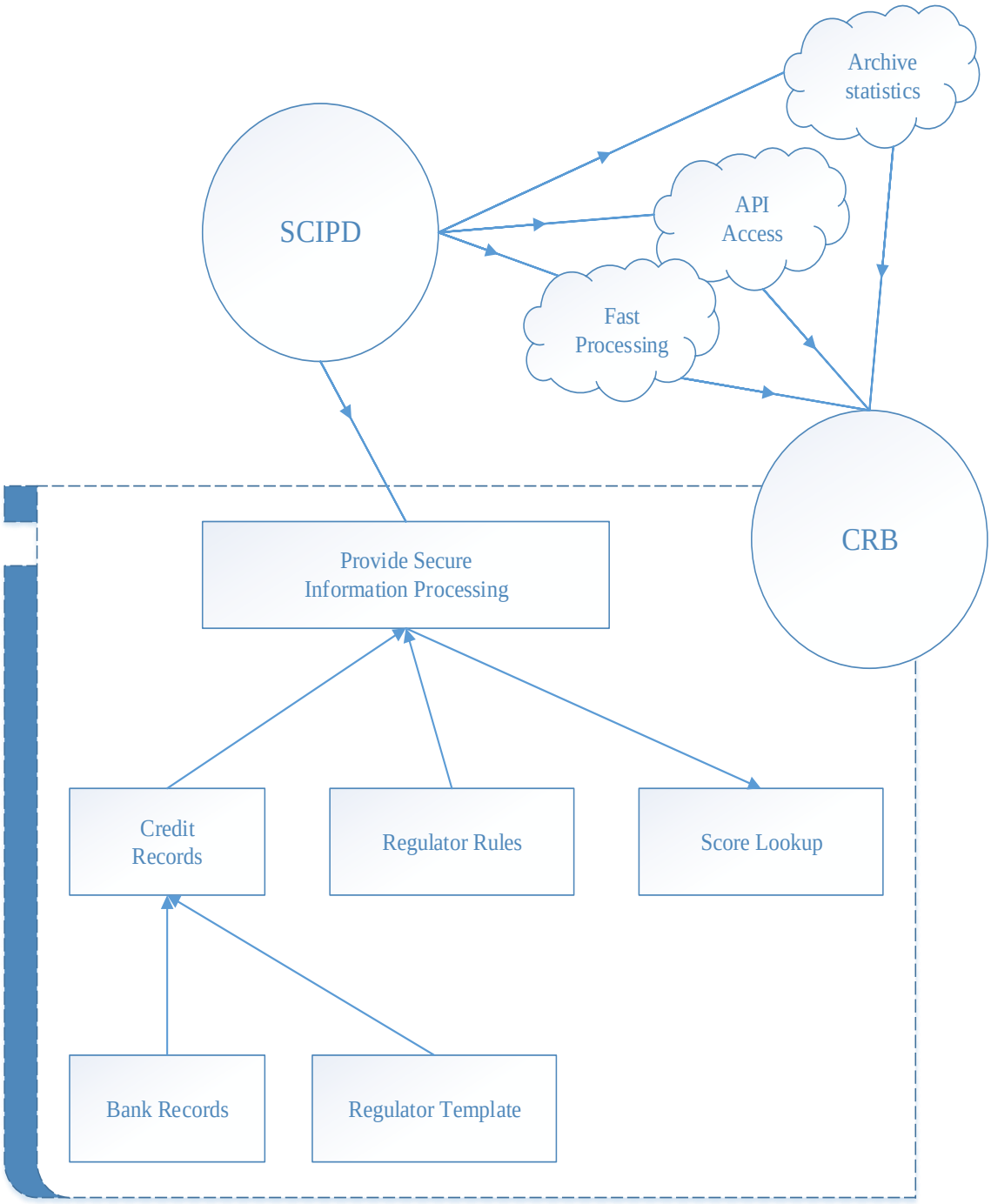

Figure 4.2: Proposed System Model (SCIPD)

Given the sensitive nature of credit information, security of the data is of paramount importance. To this end, modeling of the system takes into account security patterns from this early stage in the software development process. Whereas most software development

processes normally consider security after design of the system, we adopted Tropos development methodology because it integrates security concerns throughout the development stages. In modeling the security requirements of the system, we've used Secure Tropos, an extension of the Tropos software development methodology, to model and analyze security requirements alongside functional requirements.

4.2.3 Security Requirements Engineering

Security requirements engineering is geared at detecting and analyzing security issues in the software development process. Security mechanisms are of utmost importance because systems are subject to threats, which may influence organizational assets. As shown in figure 4.3, every communication between agents goes through a security supervisor. The security mechanism protects tasks, sub-tasks, goals and soft goals of each agent from other agents. Only an authenticated and authorized agent can access tasks or goals of another agent. In the figure, a high level representation of the interactions of the main agents in the system is shown. For example, if a financial institution agent wants to retrieve a processing report, it has to go through the security supervisor agent to be authenticated first. Every request to a credit reference bureau agent must be explicitly authorized by the parent of such agent.

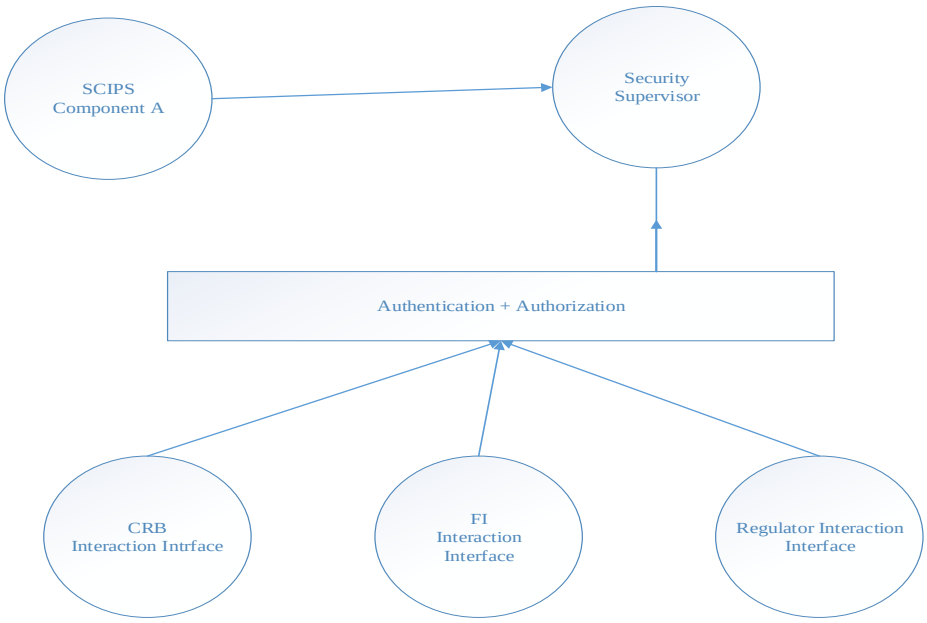

Figure 4.3: Security Architecture

It’s worth noting that there’s not just a single security supervisor agent. Each agent that creates other agents also creates a security supervisor agent to control access to its children. This is to ensure that security is localized to the parent agent. It also minimizes cross network traffic. Cross network communication will only happen if an agent wishes to communicate with a remote agent. When remote communication is necessary, then the parent of the target agent, the one whose services are being sought, controls access to the agent’s goals and tasks.

4.3 Architectural Design

In this stage of our software development methodology, we define the system’s global architecture in terms of actors interconnected through data and control flows, represented as dependencies. Additionally, we are going to map actors into a set of software agents, each characterized by its specific capabilities. This stage in our software development methodology is achieved in three steps: Overall architecture definition, identification of actor capabilities to fulfill their goals and plans, and definition of agent types and capabilities assignment to each agent.

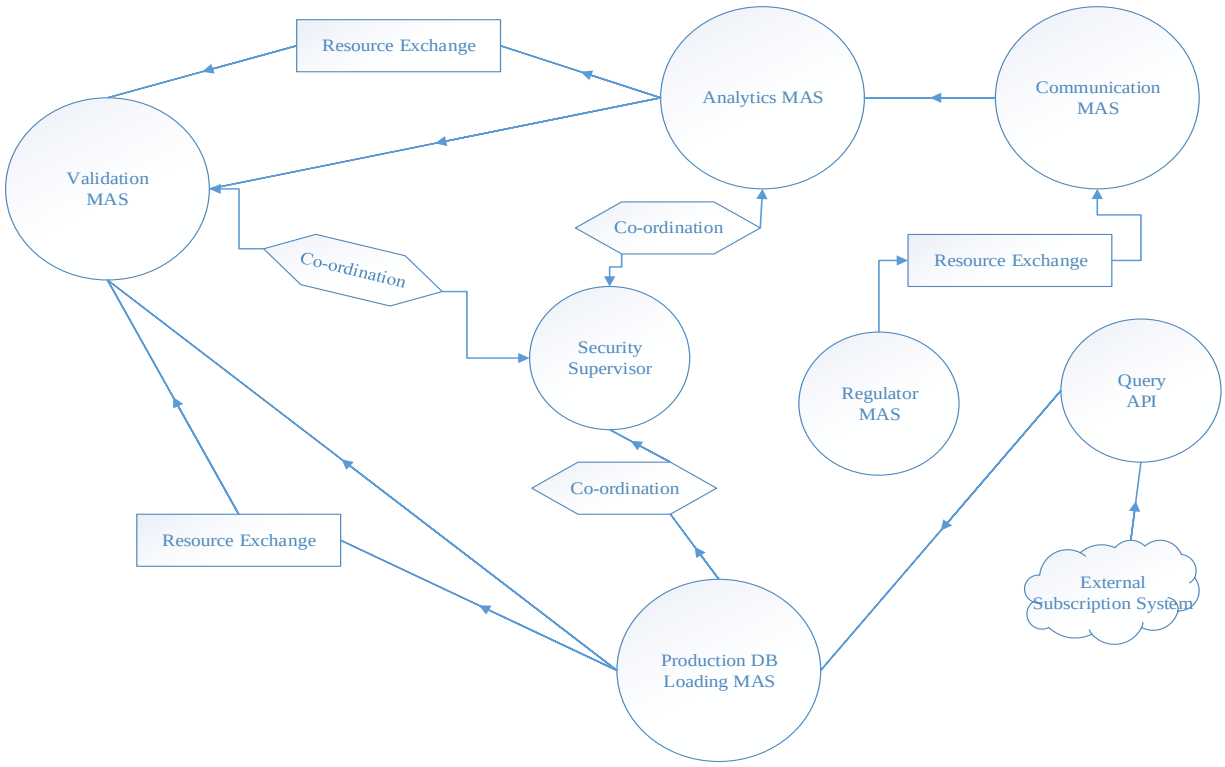

Figure 4.4: Overall Architectural Design

As shown in figure 4.4, there are five main components in the SCIPD system:

- Validation multi-agent system component.
- Analytics multi-agent system component.
- Production database multi-agent system component.
- Communications multi-agent system component.
- Regulator multi-agent system component.

As shown in the figure, the data flow is from the validation multi-agent system component to the analytics multi-agent system and the production database multi-agent system components. The validation multi-agent system component resides at the financial institution's local area network. The analytics multi-agent system and the production database multi-agent system components reside at the credit reference bureau's local area network.

Communication between the validation multi-agent system and the analytics and production database multi-agent system components take place across the Internet. To protect the integrity of the data and ensure that the data is not compromised en-route, data must be encrypted. Also, transfer of the data must take place through a secure dedicated channel for the duration of the transfer. To achieve this, SCIPD uses Transport Layer Security to encrypt communication as shown in figure 4.5. Communication between the validation multi-agent system and production database loading multi-agent systems components uses the same protocol and encryption algorithm. The same applies to communication between the analytics and the regulator multi-agent system components.

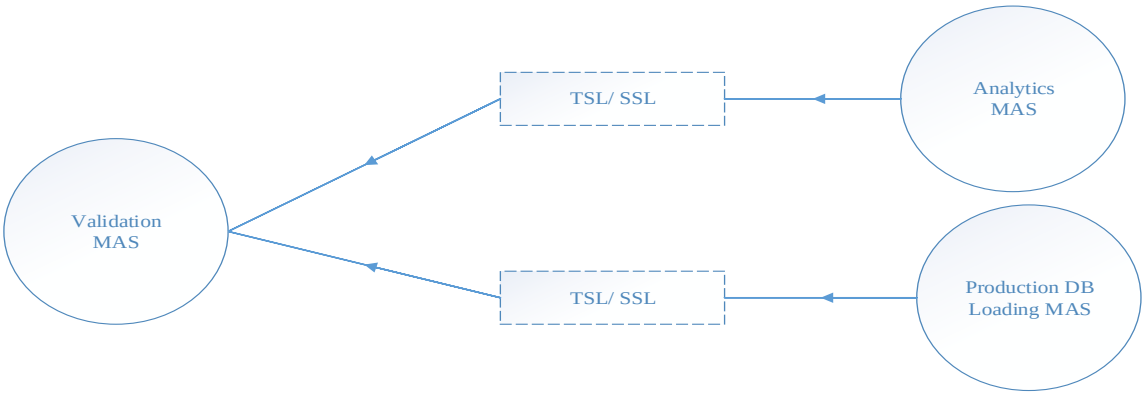

Figure 4.5: Secure communication via TSL/ SSL

Internally, each agent and sub-agent must protect its goals, plans and tasks. To do so, each agent that creates another agent also creates its own security supervisor that controls access to its internal data and state from other agents and from the host operating environment. Whenever an agent receives a message to perform a task or communicate its status from another agent, it delegates authentication and authorization to the security supervisor before fulfilling the request. If authentication and authorization fails, the request is not fulfilled and fails silently along with the security supervisor, a state monitoring agent is created for each agent with children.

The goal of a state monitoring agent is to periodically check and validate the state of the parent agent and its children. If an inconsistent is discovered, the agent hanged and its parent is informed of the state. The decision will be on the parent agent to determine whether and when to restart or resume the agent either on the same or in an alternative host. Figure 4.6, 4.7 and figure 4.8 illustrate this.

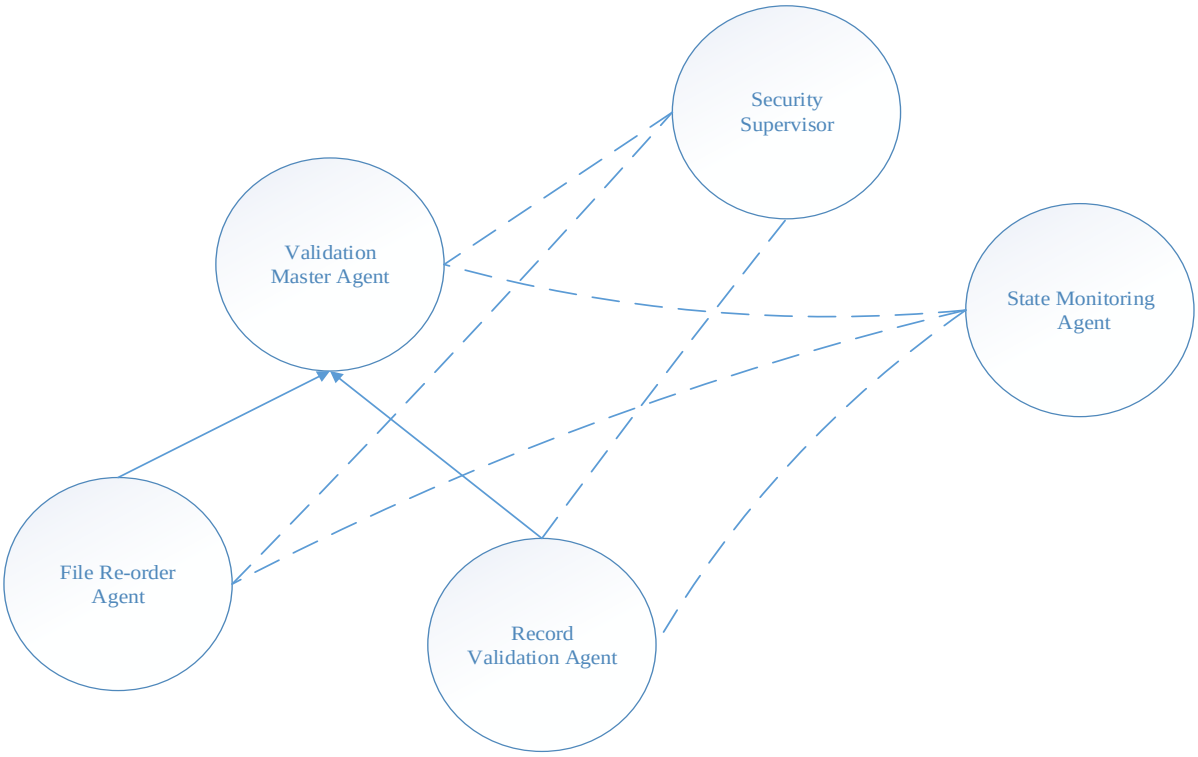

Figure 4.6: Monitoring and Security Supervision Hierarchy

Figure 4.6 shows a combined monitoring and security supervision hierarchy. It shows the communication hierarchy for security supervision and state monitoring. Figure 4.7 shows monitoring hierarchy only. It illustrates the flow of monitoring messages in the event of an exception from the lowest agents in the hierarchy to the highest working level in the hierarchy. Figure 4.8 shows the security supervision hierarchy. If, for example, the top level agent in the analytics multi-agent system component has a security breach, the top level agent in the validation multi-agent system is informed of the exception. The validation multi-agent system then resumes or restarts the analytics multi-agent system in the same host or in an alternative host.

Internally, each multi-agent system has its own supervision and monitoring hierarchy which is created dynamically.

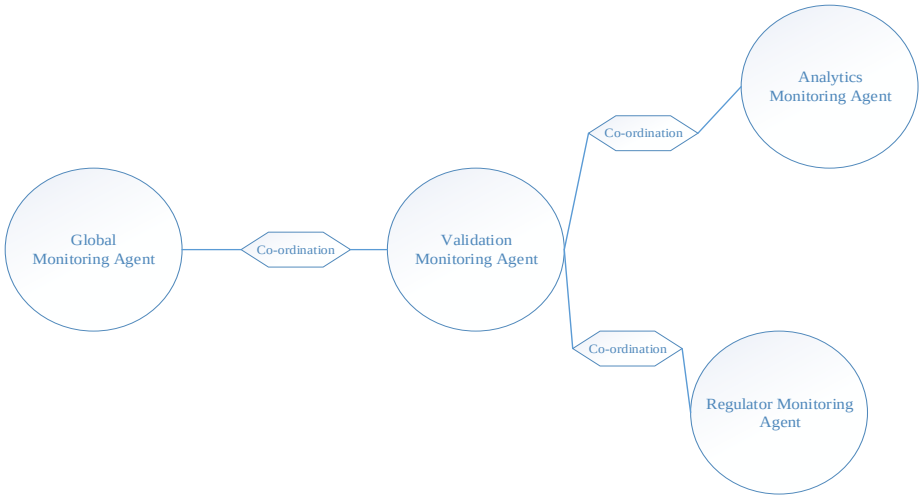

Figure 4.7: Monitoring Hierarchy

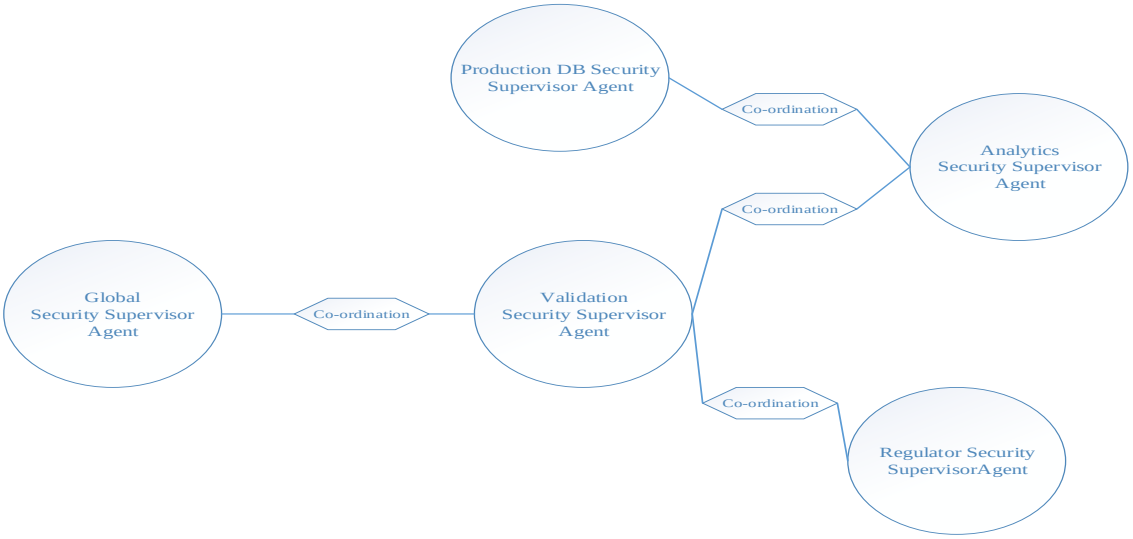

Figure 4.8: Security Supervision Hierarchy

4.3.1 Actors And Their Capabilities

The main actors (agents) in the SCIPD system are:

- Validation main actor
- Analytics main actor
- Production database loading main actor
- Global security supervision actor
- Global state monitoring actor

4.3.1.1 Validation Main Actor

The goals of this actor are:

- Read text files from the file system. It should be able to read very large files. To do this, it needs to implement reading of continuous streams of data instead of loading entire files into the main memory.
- Split text files into individual records.
- Differentiate between the different file types being loaded from the file system.
- For each file type read, it should be able to read corresponding configuration settings. Configuration settings are divided into two: Regulator rules for the file types and bureau's defined rules meant to ensure better data quality.

- It should be able to apply defined validation routines to each record read from a file of a specific type.
- It should be able to write out processing logs to the file system or insert into a database as it processes records for each file type.

To achieve these goals, the validation main actor has the following sub-actors:

- File reader agent: This actor reads files from the file system or from an FTP server by streaming. For every file streamed from the file system or from an FTP server, this actor creates sub-actor to work on the files.
- Configuration loading agent: This actor is responsible for loading configuration settings for each file type. Some of the configurations are stored locally to where the agent is running. Others are loaded from a remote server. This actor also ensures that the local copies of configuration files are up to date.

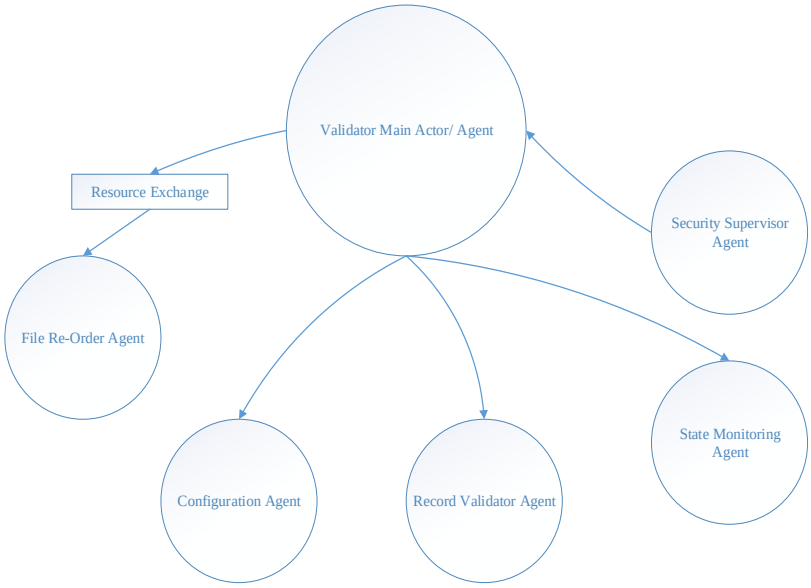

Figure 4.9: Validation Agent Hierarchy

- Record validator agent: There's a validator agent for every defined file type. The goal of a validator agent is to perform actual validation of a record. The output of such an agent are validation logs.
- Security supervisor agent: This agent's task is authentication and authorization whenever another agent requests for the main validation

agent's services. If a remote agent requires a service offered by the main validation agent or any it's child agents, authentication and authorization are delegated to the security supervisor agent.

- State monitoring agent: An agent's and its children's states are continuously monitored to know when and whether the agent's state has been compromised.
- Communication agent: All external communication, i.e., communication to other agents is through one specialized agent. This agent will choose the protocol of communication based on the type of communication to be performed. Communication can include requesting other agent's to perform some task, creating an agent in a remote host, sending out e-mail address, retrieving data from a remote FTP server, among others.

4.3.1.2 Analytics Main Actor

The goals of this actor are:

- Receive compressed validation processing logs from the validation multi-agent system component.
- Generate processing statistics based on the data it has received.
- Generate processing summary reports targeted at both the financial institution and the industry regulator.
- Generate detailed processing reports targeted at the financial institution;
- Load report templates from configuration files.
- When statistics and reports generation is complete, sends the data to the archiving multi-agent system component.

To achieve these goals, this actor has the following sub-actors:

- Report templates loading agent: This agent is charged with the responsibility of loading templates for reports to be generated from a configuration file. If necessary, loads configuration files from a remote host.
- Statistics aggregation agent: This agent is charged with the responsibility of generating aggregation statistics based on the data received from the

validation agent. To do this effectively, this agent implement a light-weight aggregation pattern to track the following measures: total number of records processed thus far, total number of valid records, total number of valid records and total value, in money terms, of all valid records processed thus far.

- Report generation agent: This agent is charged with the responsibility of generating configured reports, as loaded by the reports templates loading agent.
- Communication agent: This coordinates and chooses the appropriate protocol for communication with other agent systems and other interfaces. Communication can include requesting services from other agents, sending an e-mail notification, or retrieving a remote resource.
- Security supervision agent: This agent is charged with the responsibility of restricting to this agent system's goals, tasks and plans. All incoming requests are vetted by this agent to determine whether or not they can be fulfilled or not based on the prevailing security configuration.
- State monitoring agent: This agent monitors the state of the analytics agent system and reports any security exception to the global security supervising agent.

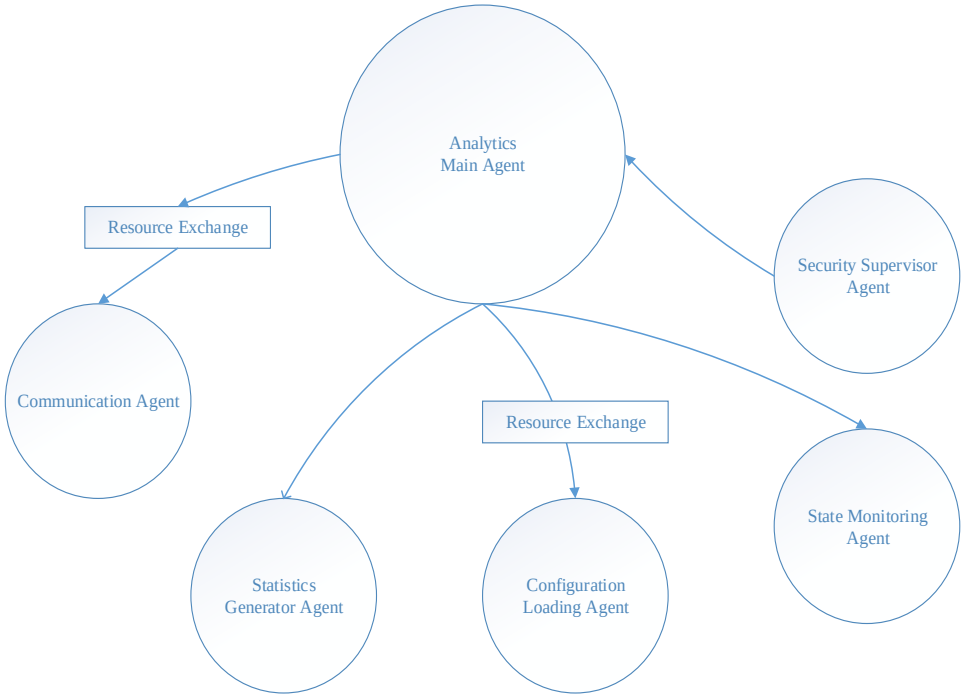

Figure 4.10: Analytics Agent Hierarchy

4.3.1.3 Production Database Loading Main Actor

Production database loading main actor has the following goals and tasks:

- Receive valid records from the validation actor system.
- Load production database configuration either from the local file system or from a remote configuration hosting server.
- Perform search to establish whether a record is to be created a new or updated;
- Generate production loading report.

To achieve these goals, the agent has the following sub-actors:

- Communication agent: The responsibility of this agent is to coordinate communicate with remote agents and remote server hosts. For example, when receiving valid records from the validation agent system.
- Database configuration loading agent: This agent is charged with the responsibility of loading all production database configurations and managing connection to them.
- Search agent: This agent performs search on existing records to determine whether a record needs to be created or updated.

- Production logs generating agent: This agent generates logs as production loading is happening. It then coordinates with the communication agent to dispatch the logs to be reviewed by the credit reference bureau staff.

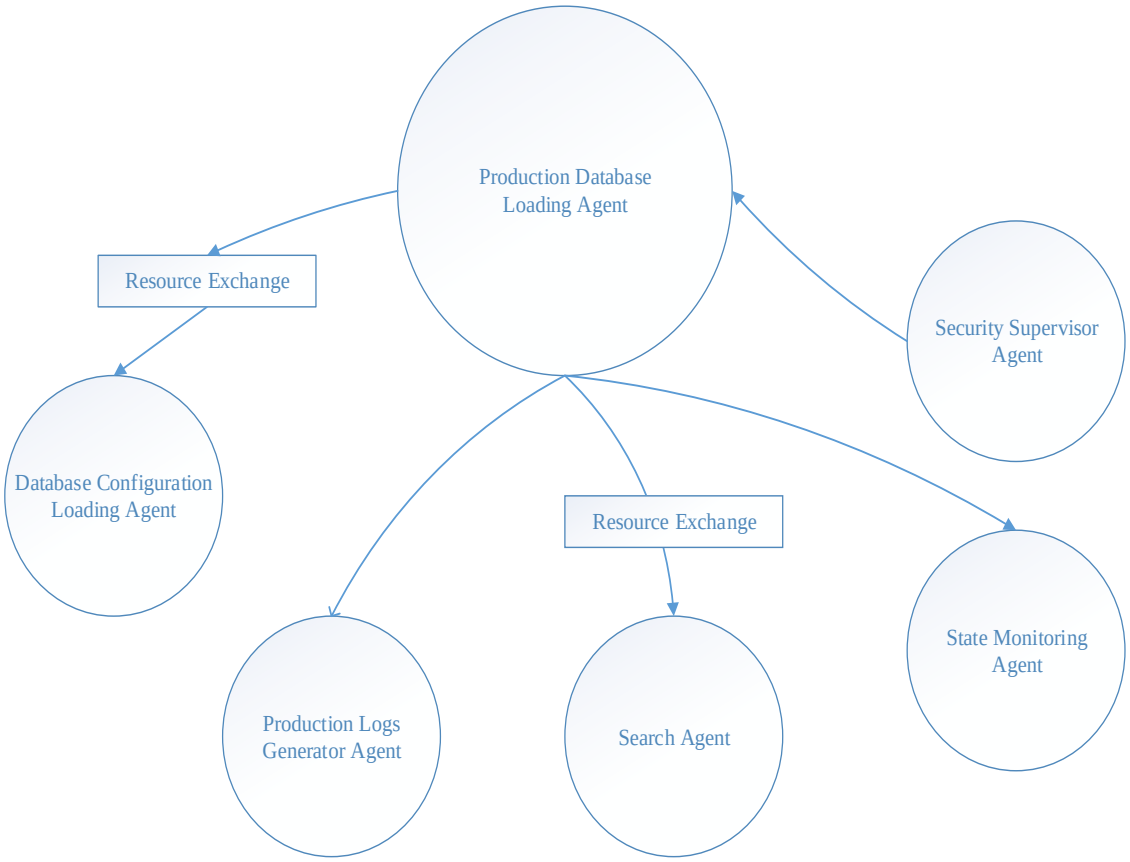

Figure 4.11: Production Database Loading Main Agent.

4.4 Detailed Design

In this stage, we are going to specify, in detail, the goals of main agents, their beliefs, capabilities and communication between the agents. Figure 4.4 shows an agent interaction protocol focusing on dialogue among the agents. In the following sections, we are going to show how the capabilities of each agent is fulfilled by defining tasks, attributes and measures for each agent and it's interaction with other agents. We've divided this section by agent system types.

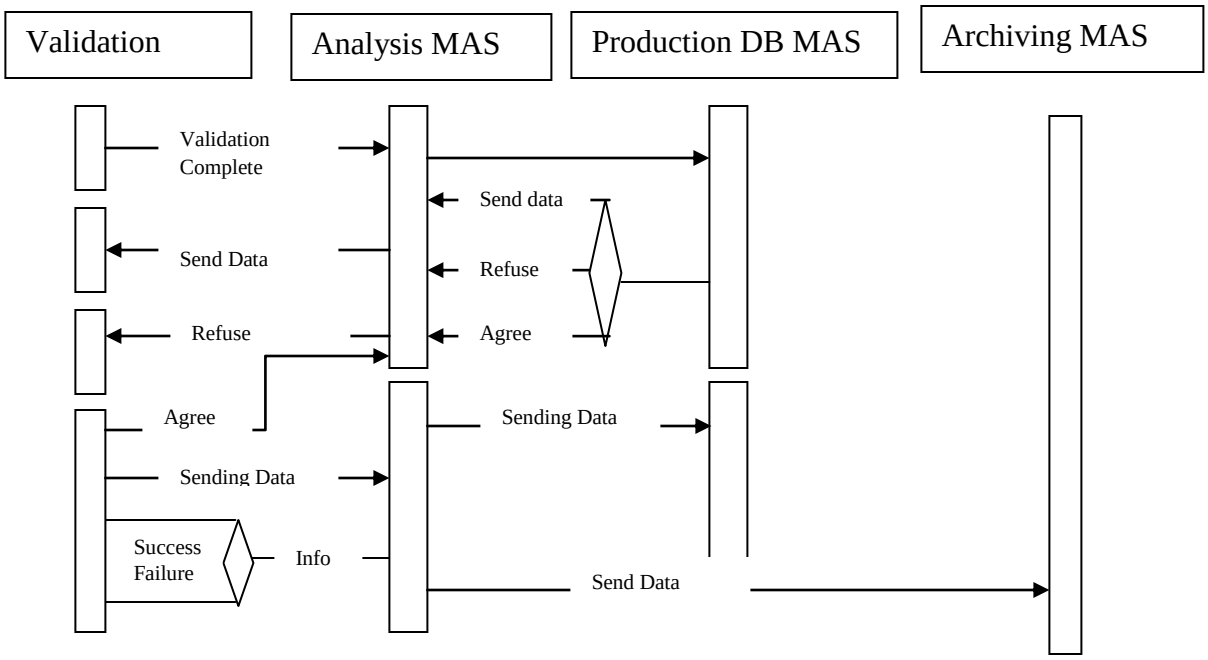

Figure 4.12: Agent Interaction Protocol

4.4.1 Validation Agent System Attributes

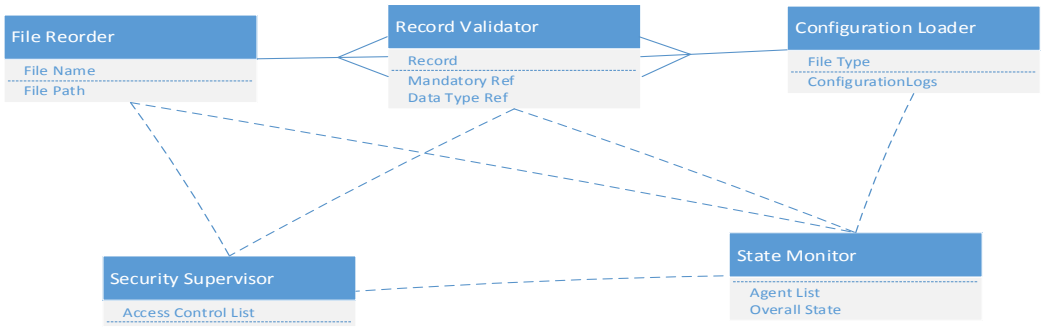

Figure 4.13: class diagram for the validation multi-agent system component.

4.4.1.1 File Reader Agent

- File name
- Absolute file path
- File type
- File size
- Batch ID
- Processing start date
- Processing end date
- FTP server host (Optional)
- FTP server username (Optional)
- FTP server password (Optional)

4.4.1.2 Configuration Loading Agent

- File Type
- Configuration base location
- Configuration up-to-date
- Configuration handler

4.4.1.3 Record Validator Agent

- Record
- Mandatory rules
- Data type rules
- Business type rules
- Validation errors
- Valid flag

4.4.1.4 Security Supervisor Agent

- Access control list
- Global security supervisor address
- Authentication time to live manager
- Authorization token generator

- Authorization validity monitor

4.4.1.5 State Monitoring Agent

- List of agents to monitor
- Overall status
- Monitoring schedule
- State criticality

4.4.1.6 Communication Agent

- Network communication protocol
- Intra-agent communication protocol
- Encryption algorithm

4.4.2 Analytics Agent System

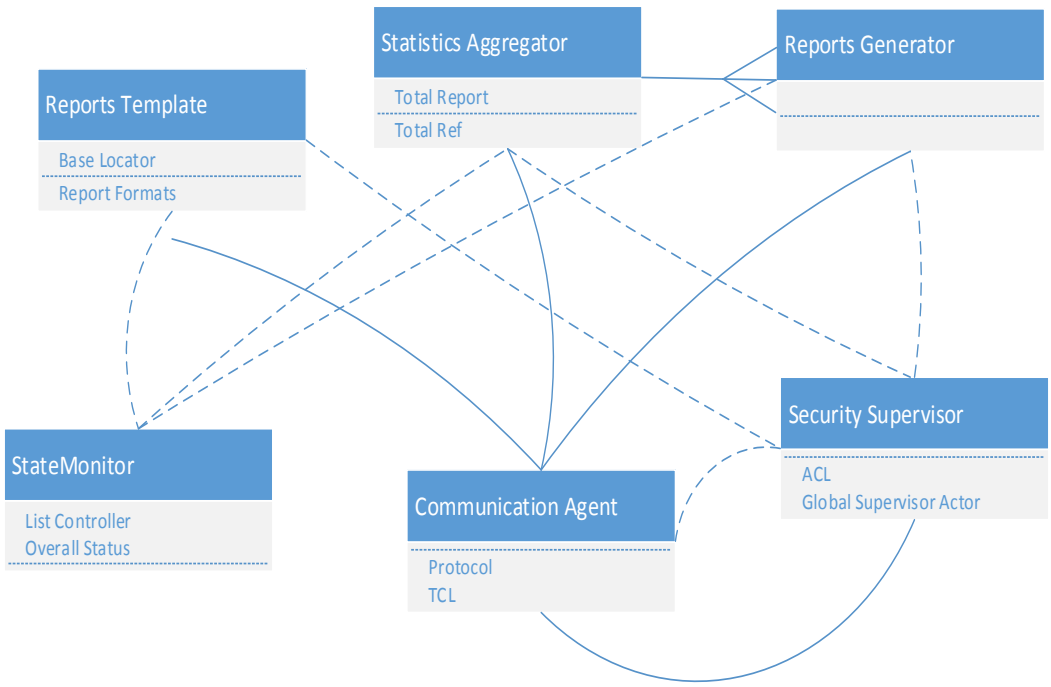

Figure 4.14: Analytics class diagram

The Analytics class diagram will be used to create the analytics multi-agent system component. The attributes and measures for each of the sub-actors of this component are shown below.

4.4.2.1 Report Templates Loading Agent

- Report template base location
- List of available report formats
- List of report recipients
- Determine recipients from metadata
- Report language

4.4.2.2 Statistics Aggregation Agent

- Total records processed
- Total number of validation errors
- Total value of all valid records

4.4.2.3 Report Generation Agent

- Pre-generate reports
- Report submission mode
- Report publishing base

4.4.2.4 Communication Agent

- Network communication protocol
- Intra-agent communication protocol
- Data encryption algorithm

4.4.2.5 Security Supervisor Agent

- Access control list
- Global security supervisor address
- Authentication time to live manager
- Authorization token generator

- Authorization validity monitor

4.4.2.6 State Monitoring Agent

- List of agents to monitor
- Overall status
- Monitoring schedule
- State criticality

4.4.3 Production Loading Agent System

The production loading multi-agent sub-system is responsible for database data loading. Its goals can be thought of as two-pronged: Search a record to determine where to create new record or update an existing record. The attributes and measure definition of this sub-system is shown in the corresponding sub-actor sections.

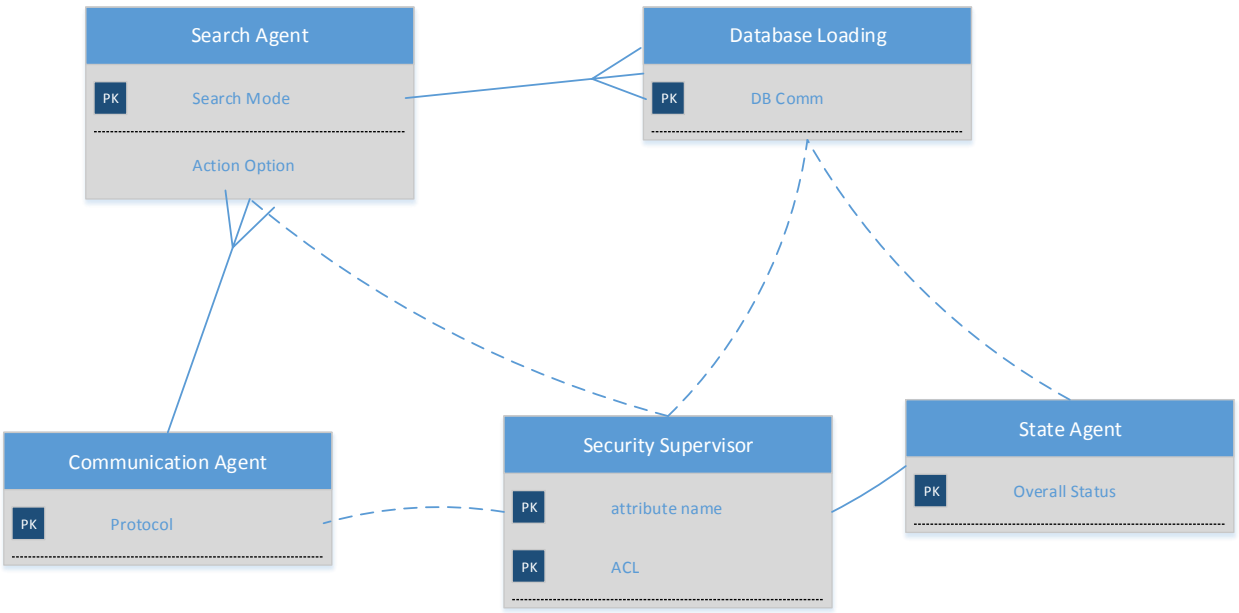

Figure 4.15: Database Loading Class Diagram.

4.4.3.1 Database Configuration Loading Agent

- List of database connection strings to interact with
- Database selection criteria
- Database connection handler

4.4.3.2 Search Agent

- Search mode
- Search result action options
- Matching function

4.4.3.3 Production Processing Logs Generating Agent$

- Log base location
- Log distribution mode
- List of log distribution recipients
- Log file split format
- Compression flag
- Compression algorithm
- Encryption flag
- Encryption algorithm

4.4.3.4 Security Supervisor Agent

- Access control list
- Global security supervisor address
- Authentication time to live manager
- Authorization token generator
- Authorization validity monitor

4.4.3.5 State Monitoring Agent

- List of agents to monitor
- Overall status
- Monitoring schedule
- State criticality

4.4.3.6 Communication Agent

- Network communication protocol
- Intra-agent communication protocol
- Data encryption algorithm

CHAPTER FIVE

5.0 Evaluation and Discussion of Results

5.1 Introduction

The test cases outlined here are meant to test the security of the designed system. These are black box tests meant to test whether or not the designed system is secure. This system has been designed to address most of security threats that dog multi-agent systems. Security threats in a multi-agent system come in a number of ways; a hosting environment may try to exploit agents resident on it, a malicious person may launch a man-in-the-middle attack on a multi-agent system, a hosting environment may attempt, deliberately or otherwise, to corrupt an agent's data or state, and a malicious agent may deliberately feed an agent the wrong data on which to act on.

A secure multi-agent system is the one that can protect both its internal state, data its working on, and the data it produces. Internal state of an agent refers to the agent's attribute values. If these values can be changed by an external program in an unpredictable ways, then such an agent is not secure and, therefore, is not reliable Agents act on data and produce data that correspond to results of their computation. Once an agent gets data, it's the agent's responsibility to guard that data for the duration of its computation. Any results from computation should also be protected from accidental or deliberate manipulation from any other agent. If an agent is incapable of guaranteeing the security of the data it's acting on, then such an agent cannot be relied upon to produce correct computation results.

5.2 Threats Posed By A Hosting Environment

A hosting environment can deliberately or accidentally alter the state of agents running on it. This test is three-pronged:

- Deliberate manual injection of data to an agent's processing queue.
- Automated agent deliberately feeding an agent wrong information.
- An agent, that's not intended to be part of the multi-agent system running on the host deliberately attempting to alter one of the attributes of an agent.

5.3 Threats Posed By Man-In-The-Middle Attacks

This test is targeted at investigating the vulnerability of cross-network agent communication. Our system uses transport layer security (TLS) and secure sockets layer (SSL) for communication. As a control, we have a channel that does not use any channel encryption algorithm for communication. We first attempt to detect the agent communication traffic by using a packet sniffer. Once we discover the traffic, we then write a script to interfere with the communication by altering the packet header. One such alteration is changing destination host address. Another is to bring the whole communication to a standstill by consuming the packets before they reach the intended host.

5.4 Threats Posed By A Remote Malicious Agent

A multi-agent system running in one host might pose threats to a multi-agent system running on another host. The threat can be on the agent's state or data. To perform this test, we've constructed a separate multi-agent system whose work is solely to corrupt other multi-agent systems' states and data. This, it does by injecting random data and setting the agents' states to random values.

5.5 Test Results Discussion

Our results showed that a multi-agent system with no security at all factored into its design will always be vulnerable. This is due to the fact that multi-agent systems are inherently loosely coupled. To achieve security protection of some level, security must be factored into a multi-agent system design right from the requirements stage. Like any piece of software where security is important, software implementation should use defensive programming to improve the security of the system.

Without use of a monitoring system, it's impossible to know if an agent has been compromised unless one reviews the results of the agent's computation. Implementing a monitoring framework is especially very important for maintaining the integrity of a multi-agent system running on a host that is likely to corrupt the agent's state and/or data.

A monitoring framework alone is not enough. Whereas as such a framework informs us of the inadequacies in our system, what it doesn't do is prevent these inadequacies from being there in the first place. One way we found is most effective at preventing these vulnerabilities is a

well-designed security framework. The security framework includes a requirement that each agent requesting for a service must be authenticated and authorized before a service can be rendered to it. The security framework also protects the monitoring framework from malicious agents and/or hosts.

In conclusion, a secure multi-agent system is one that has both a good monitoring and security supervision framework. These two frameworks complement each other. Where the security might fail, the monitoring framework would report an exception. Coupled with these frameworks, the implementation must be use defensive programming techniques.

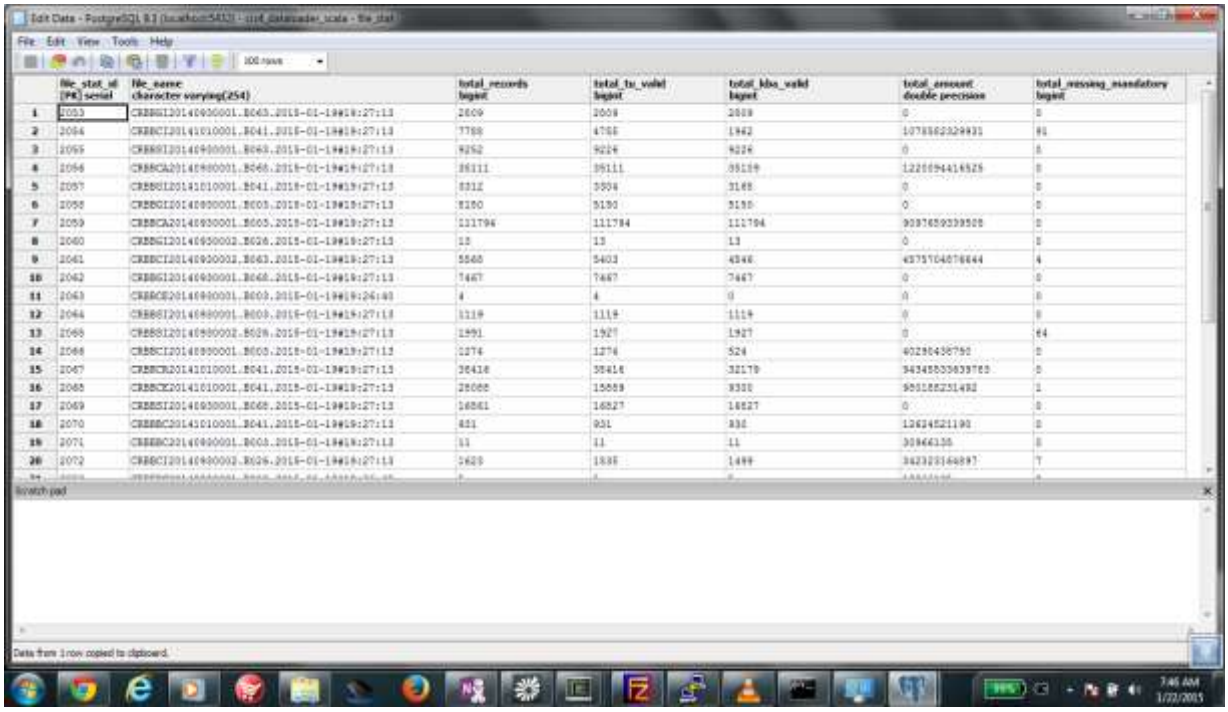

Figure 5.1: Results of computation in a secure MAS

Figure 5.1 shows the results of computation on files performed by a completely secure multi-agent system. Completely secure means the multi-agent system implements both the state monitoring and security supervision frameworks.

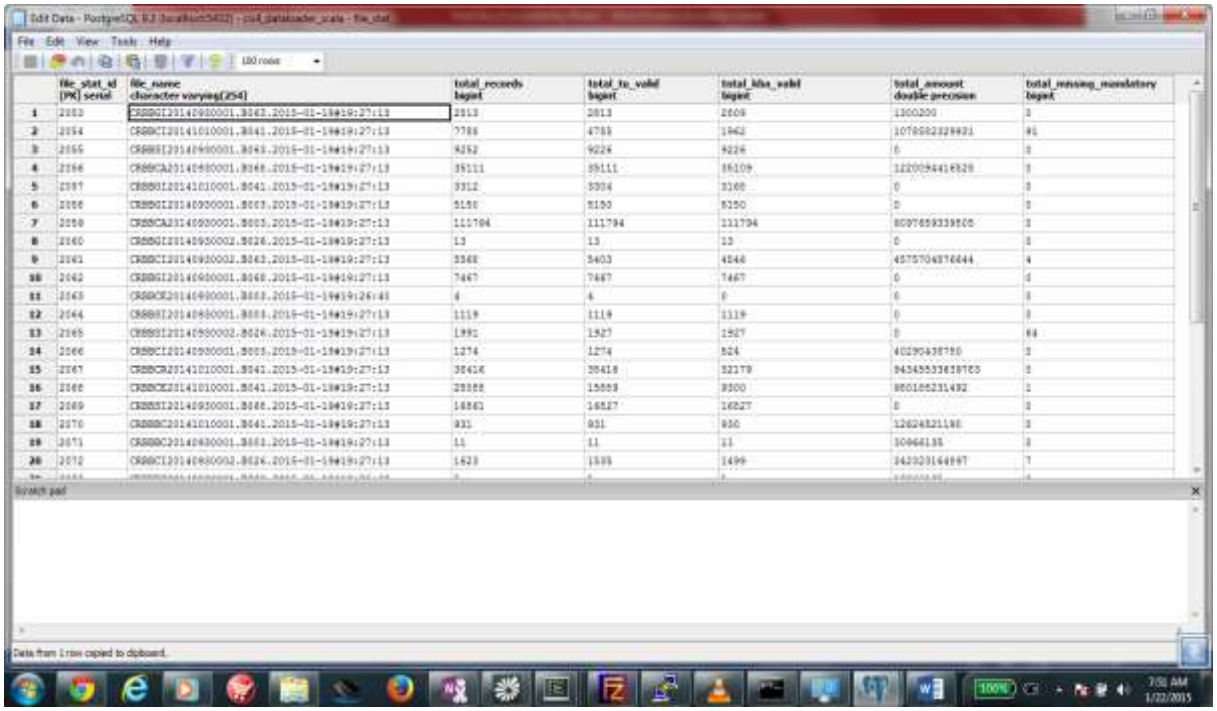

Figure 5.2: Results of computation in unsecured MAS

Figure 5.2 shows the result of computation on the same sets of files as those shown in figure 5.1. Unlike in figure 5.1, however, the multi-agent system that produced this output did not implement either the monitoring or the security supervision framework. As such, another multi-agent system was able to introduce new records in the processing pipeline of the multi-agent system. The result, as illustrated by record 1 in figure 5.2, is an output that includes records and balance amount measures that were not contained in the original files. Multi-agent systems are inherently loosely coupled. So, while it should be possible for multi-agent systems to interact, that interaction should be an expected one. A security supervision framework would ensure that if an agent cannot be identified then its request is not serviced. Also, the monitoring agent would detect unexpected state and escalate the exception to the parent agent which would execute the appropriate mitigation strategy.

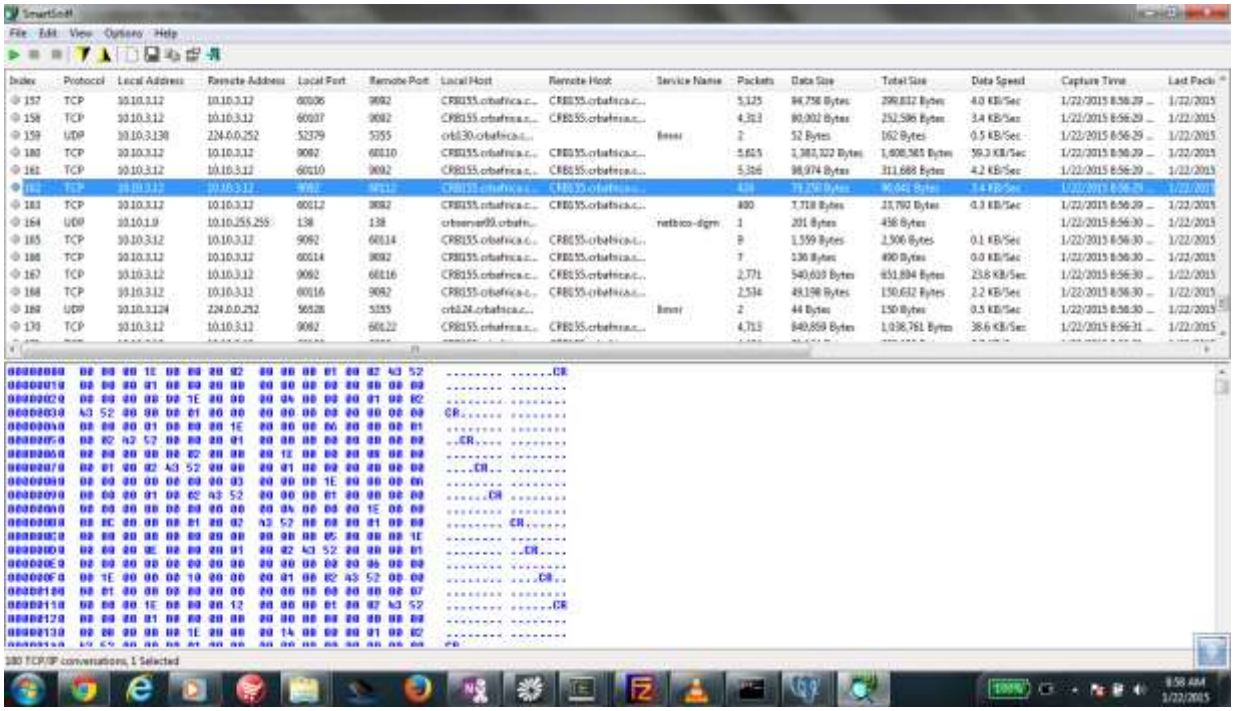

Figure 5.3: Communication traffic in unsecured MAS

Figure 5.3: Shows intercepted communication traffic in a multi-agent system which has not implemented the security supervision and state monitoring frameworks. As shown, the data can be easily converted into readable format with the appropriate tools; there are a myriad of tools available on the Internet for interpreting this. This particular screenshot was produced using Smart Sniff TCP traffic sniffer.

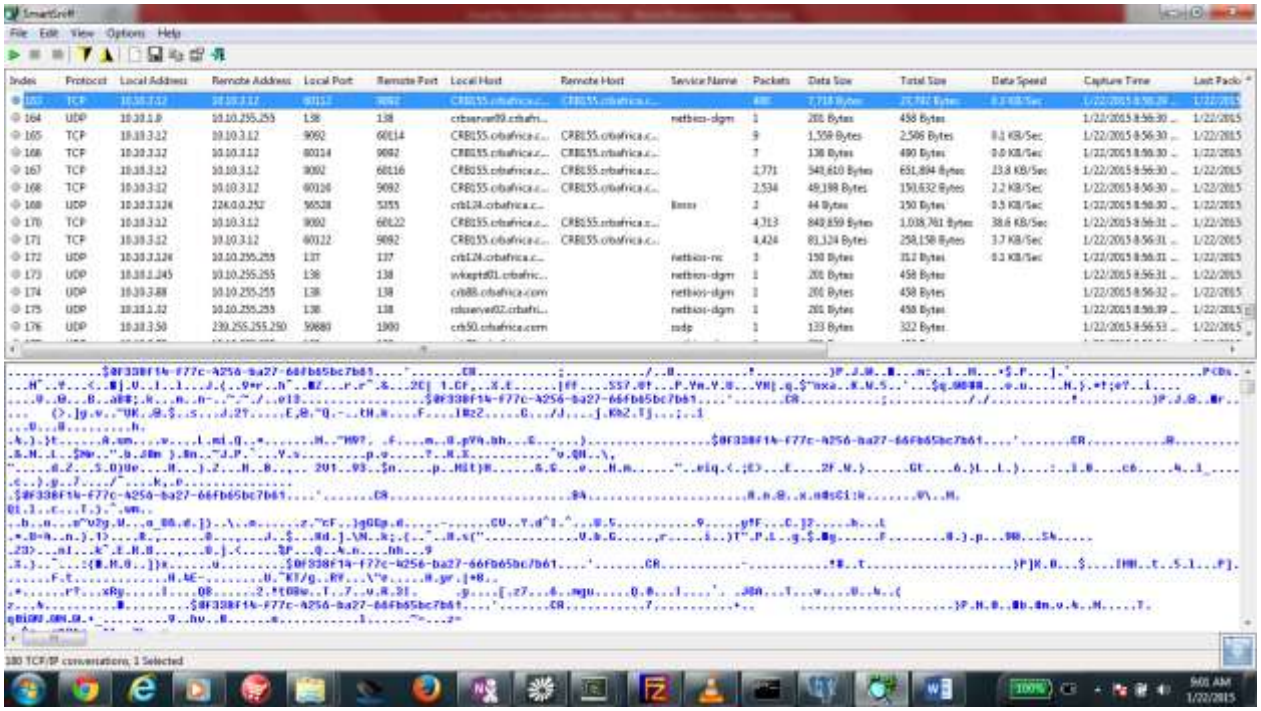

Figure 5.4: Communication in a secure MAS

Figure 5.4 shows the same traffic but in an environment that has implemented the security supervisor and the state monitoring frameworks. As shown, traffic data is encrypted and thus cannot be easily interpreted.

CHAPTER SIX

6.0 Conclusions and Future Work

6.1 Introduction

In this chapter, we present high level conclusions of our work as well as recommendations for future work. We organize our conclusions based on the four objectives that guided the project. This include formulation of a multi-agent security mechanism, design and development of a prototype implementing the proposed mechanism, evaluation of the security properties of the prototype and demonstration of a secure multi-agent system using a use case.

6.2 A Multi-Agent Security Mechanism

Previous works have suggested security approaches for multi-agent systems but most of these works address just one or two of these requirements. To achieve a completely secure multi-agent system, a multi-faceted approach to security must be adopted. Such an approach is proposed in this work. We suggested separate algorithms to counter security threats in mobile agents by malicious hosts and incorporated them in the agent's code effectively protecting our agents from the different forms of attacks.

6.3 Design and Development of a Secure Multi-Agent System Prototype

A number of agent oriented software engineering (AOSE) methodologies exist. Most of these methodologies however normally consider security requirements after design of the system. We adopted Tropos development methodology because it integrates security concerns throughout the development stages. In modelling the security requirements of the system, we've used Secure Tropos, an extension of the Tropos software development methodology, to model and analyze security requirements alongside functional requirements. We then developed the prototype with Scala which is a production ready tool. This helped us to produce a compiled code for our agents therefore minimizing the ability of the executing platform to read and possibly alter the agent's code.

6.4 Evaluation of the Prototype

For any computer system to be said to be secure, it must meet several security requirements i.e. confidentiality, integrity, authentication, authorization, non-repudiation and availability. We conducted black box tests to test our prototype using this metrics and compared the results

with parallel systems that incorporated other security mechanisms. Computation results in the other systems revealed inadequacies while our system produced expected results. While multi-agent systems are inherently loosely coupled, their interactions are expected and thus can be monitored. Our mechanism provides for such monitoring without adding to computational overheads and as such makes it easier to trace security violations while making them difficult to happen in the first place.

6.5 A Use Case Demonstrating A Secure Multi-Agent System

We chose a Credit Reference Bureau (CRB) use case to demonstrate our security mechanism. We enhanced the functionality of the CRB by introducing a secure multi-agent system to run its operations. The CRB handle highly confidential transactions using traditional file transfer between executing hosts. While transforming the CRB to be truly distributed and reducing communication overheads between hosts, our mechanism also enhanced integrity of the system. Proper security considerations for mobile-agent systems thus enable conversion of legacy systems into trusted distributed applications.

6.6 Future Work

This research project concentrated on securing MAS to mitigate against security threats imposed by malicious executing environments. The approach used mitigates against such threats but has however no mechanism of logging or marking insecure hosts so as to notify other agents not to visit them. Further work can be done to include such an option.

The prototype can also be enhanced to include functionality such that in the event that a previously insecure host is solved, that host should not be marked as insecure indefinitely.

There are many cryptographic algorithms available. In the prototype, we have used selected encryption algorithms like SHA1 with 256 bits and security mechanisms like TLS. Experiments can be done using current algorithms as well as emerging hashing and cryptographic technologies to enhance mobile agent security

References

Ahn, J., 2013. Mobile Agent Group Communication Ensuring Reliability and Totally-ordered Delivery. 10(3).

Ahuja, P. & Sharma, V., 2012. A JADE Implemented Mobile Agent Based Host Platform Security. *International Institute for Science, ,* 3(7).

Ahuja, P. & Sharma, V., 2012. A Review on Mobile Agent Security. *International Journal of Recent Technology and Engineering (IJRTE),* June.

Aneiba, A. & Rees, S.J., 2004. Mobile Agents Technology and Mobility.

Bellavista, P. et al., 2004? *Security for Mobile Agents: Issues and Challenges*. Bologna: University of Bologna.

Bellavista, P., Corradi, A. & Stefanelli, C., 1999. *An Open Secure Mobile Agent Framework for Systems Management*. Bologna: Università di Ferrara.

Bosede, O., A.O, O. & Aderounmu, G.A., 2013. A framework for an Operating System-based Mobile Agent. *IOSR Journal of Computer Engineering (IOSR-JCE),* 14, pp.1 - 6.

Ebietomere, E.P. & Ekuobase, G.O., 2014. Issues on Mobile Agent Technology Adoption. *African Journal of Computing & ICT,* 7.

Groot, D.R.A.d., 2004. *Cross-Platform Generative Agent Migration; An Agent Factory Approach*. Master's Thesis. Vrije Universiteit Amsterdam.

Gupta, S. & Sinha, S., 2013. A Secure Architecture for Mobile Agent Based Communication System. *International Journal of Latest Trends in Engineering and Technology (IJLTET),* 2.

Herv´e, P., 2002. A Mobile Agent Systems' Overview.

Ichiro, S., 2008. Mobile Agents.

Karzan, A.A. & Erdoğan, N., 2014. Securing Mobile Agent Systems in Which the Agents Migrate via Publish/Subscribe Paradigm. *Lecture Notes on Software Engineering*, 2.

Lovrek, I. &Sinkovic, V., 2000? *Performance Evaluation of Mobile Agent Network*. University of Zagreb.

W. Jansen, T. Karygiannis, NIST Special Publication 800-19-Mobile Agent Security, Technical paper, National Institute of Standards and Technology, Computer Security Division.

Mahmoodi, M. & Varnamkhasti, M.M., 2013. A Secure Communication in Mobile Agent System. *International Journal of Engineering Trends and Technology (IJETT)*, 6.

Marikkannu, P., Jovin, J.J.A. & Purusothaman, T., 2012. Self-Protected Mobile Agent Approach for Distributed Intrusion Detection System against DDoS Attacks. *International Journal of Information and Electronics Engineering*, 4.

Mishra, P.K. & Singh, R., 2014. A Survey on Reliability Estimation Techniques for Mobile Agent based Systems. *International Journal of Advanced Computer Research*, 4.

Osero, B.O. & Mwathi, D.G., 2014. Implementing Security on virtualized network storage environment. *International Journal of Education and Research*.

Pai, P., Shinde, S.K. & Khachane, A.R., 2012. Security In Mobile Agent Communication. *International Journal of Advanced Engineering Research and Studies*, 1.

Persson, M., 2001. *Mobile Agent Architectures*. Scientific Report. LINKÖPING: Division of Command and Control Warfare Technology.

Seltman H. J., 2014. Experimental Design and Analysis.

Shrivastava, R. & Mehta, P., 2012. Securing Mobile Agent and Reducing Overhead Using Dummy and Monitoring Mobile Agents. *International Journal of Management, IT and Engineering* , 2.

Singh, Y., Gulati, K. & Niranjan, S., 2012. Dimensions and Issues of Mobile Agent Technology. *International Journal of Artificial Intelligence & Applications (IJAIA)*, 5.

Singh, A. & Malhotra, M., 2012. Agent Based Framework for Scalability in Cloud Computing. *International Journal of Computer Science & Engineering Technology (IJCSET)*.

Srivastava, S. & Nandi, G.C., 2014. Fragmentation based encryption approach for self protected mobile agent. *Journal of King Saud University - Computer and Information Sciences*, 26, pp.131- 142.

Upadhye, S. & Khot, P.G., 2013. Optimize Security solution for mobile agent security: A Review. *International Journal Of Engineering And Computer Science*, 2, pp.322-29.

Verma, G., 2012. Strategies of Mobile Agent for Handling a Task. *International Journal of Engineering and Innovative Technology (IJEIT)*, 2.

Virmani, C., 2012. A Comparison of Communication Protocols for Mobile Agents. *International Journal of Advancements in Technology*, 3.

Appendices

Appendix 1: Application Configuration Source Code

```
akka {
 actor {
  provider = "akka.remote.RemoteActorRefProvider"
 }
 remote {
                enabled-transports = ["akka.remote.netty.ssl"]
                secure-cookie = "090A030E0F0A05010900000A0C0E0C0B03050D05"
                require-cookie = on

  netty.tcp  = {
   hostname = "127.0.0.1"
   port = 8989
  }

  netty.ssl = {
      hostname = "127.0.0.1"
      port = 9898
      enable-ssl = true
      security {
          key-store = "C:/ws/scala/hello-akka-local/src/main/resources/KEYSTORE/keystore"
      key-store-password = "09040407050407080702010C0903090D0C0E0906"
      key-password = "09040407050407080702010C0903090D0C0E0906"
```

```
        trust-store = "C:/ws/scala/hello-akka-local/src/main/resources/KEYSTORE/truststore"
        trust-store-password = "09040407050407080702010C0903090D0C0E0906"
        protocol = "TLSv1"
        random-number-generator = "AES128CounterSecureRNG"
        enabled-algorithms = ["TLS_RSA_WITH_AES_128_CBC_SHA"]
        }
    }


  }
}
```

Appendix 2: Validation Utilities Source Code

```
package utils

import org.slf4j.LoggerFactory
import java.util.Properties
import scala.collection.mutable.Map
import java.io.FileInputStream
import kafka.producer._

object CfgUtils {
  val cfgBaseDir = "D:/sws/scipd-vs/cfg/"
  val remoteDataDir = "C:/tmp/others/raw"
  val localDataDir = "D:/tmp/data"
  val logBaseDir = "C:/tmp/others/logs"
  val logger = LoggerFactory.getLogger(getClass)

  // FTP Server Config
  val FTP_HOST = "localhost"
  val FTP_PORT = "21"
  val FTP_DIR = "ftp"
  val FTP_USER = "ftp_user"
  val FTP_PASS = "0k5LLO12"

  // File type codes
```

```
val IC = "ic"

val CI = "ci"

val CA = "ca"

val CR = "cr"

val GI = "gi"

val SI = "si"

val FA = "fa"

val BC = "bc"


// Institution type codes

val BANK = 'B'

val DPFB = 'B'

val MFB = 'M'


// Number of columns

val IC_COLNUM = 67

val CI_COLNUM = 58

val CA_COLNUM = 25

val SI_COLNUM = 37

val GI_COLNUM = 40

val CR_COLNUM = 26

val BC_COLNUM = 11

val FA_COLNUM = 15
```

```
// Validation error classes
val ERROR_TYPE_COLNUM = "INVALID NUMBER OF COLUMNS"
val ERROR_TYPE_CLASS1 = "VALIDATION ERROR CLASS 1"
val EMPTY_VALUE_REGEX = "(NA|NONE|NULL|[^A-Za-z0-9])"

// KAFKA communication config
val KAFKA_HOSTS = "127.0.0.1:9092"
val IC_TOPIC = "IC"
val IC_CONSUMER_GRP = "IC"
val CI_TOPIC = "CI"
val CI_CONSUMER_GRP = "CI"
val SI_TOPIC = "SI"
val SI_CONSUMER_GRP = "SI"
val GI_TOPIC = "GI"
val GI_CONSUMER_GRP = "GI"
val BC_TOPIC = "BC"
val BC_CONSUMER_GRP = "BC"
val CA_TOPIC = "CA"
val CA_CONSUMER_GRP = "CA"
val CR_TOPIC = "CR"
val CR_CONSUMER_GRP = "CR"
val FA_TOPIC = "FA"
val FA_CONSUMER_GRP = "FA"
```

```
  // Kafka Producers
  val IC_KAFKA_PRODUCER = new KafkaProducer(IC_TOPIC, KAFKA_HOSTS)
  val CI_KAFKA_PRODUCER = new KafkaProducer(CI_TOPIC, KAFKA_HOSTS)
  val SI_KAFKA_PRODUCER = new KafkaProducer(SI_TOPIC, KAFKA_HOSTS)
  val GI_KAFKA_PRODUCER = new KafkaProducer(GI_TOPIC, KAFKA_HOSTS)
  val CA_KAFKA_PRODUCER = new KafkaProducer(CA_TOPIC, KAFKA_HOSTS)
  val CR_KAFKA_PRODUCER = new KafkaProducer(CR_TOPIC, KAFKA_HOSTS)
  val BC_KAFKA_PRODUCER = new KafkaProducer(BC_TOPIC, KAFKA_HOSTS)
  val FA_KAFKA_PRODUCER = new KafkaProducer(FA_TOPIC, KAFKA_HOSTS)

  def loadProperties(instType: Char, fileType: String): Properties = {
    val props = new Properties
    props.load(new FileInputStream(cfgBaseDir +
      fileType.toLowerCase + "." + instType.toLower + ".properties"))

    val msgProps = new Properties
    msgProps.load(new FileInputStream(cfgBaseDir + "msg/en.properties"))
    props.putAll(msgProps)

    return props
  }
}
```