UNIVERSITY OF NAIROBI

SCHOOL OF COMPUTING AND INFORMATICS

AGENT-BASED SECURITY INFORMATION MONITOR

BY

NGUGI ANTHONY MWANGI

THIS RESEARCH PROJECT REPORT IS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE OF UNIVERSITY OF NAIROBI

SEPTEMBER 2014

Declaration

I ANTHONY MWANGI NGUGI do declare that this research project report is my own original work and has not been presented anywhere for any academic award.

Signature: ___________________________________ Date:_____/_____/________

ANTHONY MWANGI NGUGI

P58/63741/2011

I DR. ELISHA T. OPIYO OMULO as the University supervisor do confirm that this research project report was presented for evaluation by my approval as partial fulfillment of the requirements for the award of the degree of Master of Science in Computer Science of the University of Nairobi.

Signature:_____________________________ Date: ______/______/______

DR. ELISHA T. OPIYO OMULO

Dedication.

I dedicate this project to my lovely wife Faith and my beloved son Destiny

Abstract

In the recent times, computer systems have been employed intensively to solve physical security challenges. This has been catalyzed by the nature of security that is dynamically changing on a daily basis. Numerous intelligence security systems have been developed, however they lack capacity to collect process and exchange intelligence needed for timely action. The systems cannot detect crimes before they happen because they are only made to be reactive in nature. The data is presented in different formats making the process of analysis and exchange a complex endeavor. The purpose of this project is to address physical security issues by providing and exchanging sensitive security information quickly for prompt action and provide an information exchange platform that can be adopted by any certified security organ. This is facilitated by recording of intelligence collected into the database as soon as it is obtained and disseminate it immediately to the people interested. This study embarked on Multi agent system for monitoring security intelligence information. The main function was broken into five main functions namely:- Authentication of users, uploading of new intelligence, Information parsing, sending of email alerts, searching for intelligence Several agents were identified which includes and not limited to Database agent, communication agent, authentication agent, user and email agent. Prometheus methodology was adopted for analysis and design of the multi agent system. The implementation was done using Java Agent development Environment (JADE Platform) and MySql database. The prototype was evaluated by a group of security officers from a state corporation. In terms of overall system functionality 95% of Officers rated it excellent while 5% rated it good. Future work proposed is application of agents for business competitive advantage and combination of data mining tools and techniques to pull information from rich data repository

Acknowledgement

I Thank the Almighty God for his sufficient grace, love and care throughout the project. May the Lord be glorified. I would like to extend my gratitude to my supervisor Dr. Elisha T. Opiyo Omulo who gave me unwavering support, guidance, correction and time. I would like to appreciate Prof. Peter Wagacha and Mr. Ogutu (Panel 5) for their relentless support and professional advice. Last but not least to all security officers who participated in evaluation of the project. I will forever be grateful

Contents

LIST OF FIGURES

Definition of Important terms

Department 1 refers to section that is responsible in electronic transmissions that can be collected by ships, planes, ground sites, or satellites. Also has a role of the interception of communications between two parties

Department2 - this is intelligence collected from publicly available sources. Such sources includes:- newspaper, journals, radio, television and the internet.

Department3 - is an intelligence gathering discipline which collects information via satellite and aerial photography

Department4 is intelligence gathered by means of interpersonal contact. Typical activities consist of interrogations and conversations with persons having access to information.

IM – Intelligence monitoring

CHAPTER ONE: INTRODUCTION

1.1 Background Information

National security is a high priority in Kenya today. Kenya is a common and attractive targets to terrorists due to many factors that includes but not limited to its geographical location, ethnic composition, political stability, unstable neighbors, poverty, Islamic fundamentalism and lax of law enforcement (Aronson, 2013)

In 1998, the American Embassies in Nairobi and Dar es Salaam (Tanzania) were attacked taking the lives of hundreds and destruction of millions and millions of property. In 2002 terrorists widely believed to be affiliated with the perpetrators of Embassy attack detonated a bomb at Kikambala hotel at Kenyan coast while simultaneously shooting a surface to air missile at an Israel commercial aircraft, missing the target by a whisker. Most recently a September, 2013 insurgent attack on the Westgate shopping mall in Nairobi made international headlines and took the lives of 67 individuals from many countries around the world.

Terrorism is a global challenge. its solution cannot be fought single handedly by a particular continent, nation or even security organ individually. It requires cooperation, coordination, information exchanges among all the stake holders

The security intelligence organization relies on different sources of information in order to combat terrorism. Such sources are organized into departments which includes:- Department 1, Department 2, Department 3 and Department 4 specialized systems for Intelligence gathering. These systems collect and exchange this information in digital format using secure systems that are highly encrypted. This information exchange solution is incapacitated in the sense that it lacks proactivity, reactivity and speed for decision making.

Today's society is referred to as global village, the use of ICT has completely changed the lifestyle of people. Any kind of information can be found in digital format and it is inherently distributed across physical and logical locations. The world is also characterized by many computerized systems interconnected through networks such as intranets, extranets and the open internet.

Software agents are autonomous entities in the sense that they can migrate to other systems working on behalf of users to collect and analyze information. Agents are also proactive entities because they can process data on behalf of the human or other agents they are designed to collaborate with. The development of agent based system would particularly be very essential since they work in very complex and distributed environments. Agents will interact through coordination, corporation and negotiations.

In this study, agent based system for monitoring and exchange of terrorism information between the various systems in an intelligence organization was developed. Each user of the system accesses the system through the web application platform.

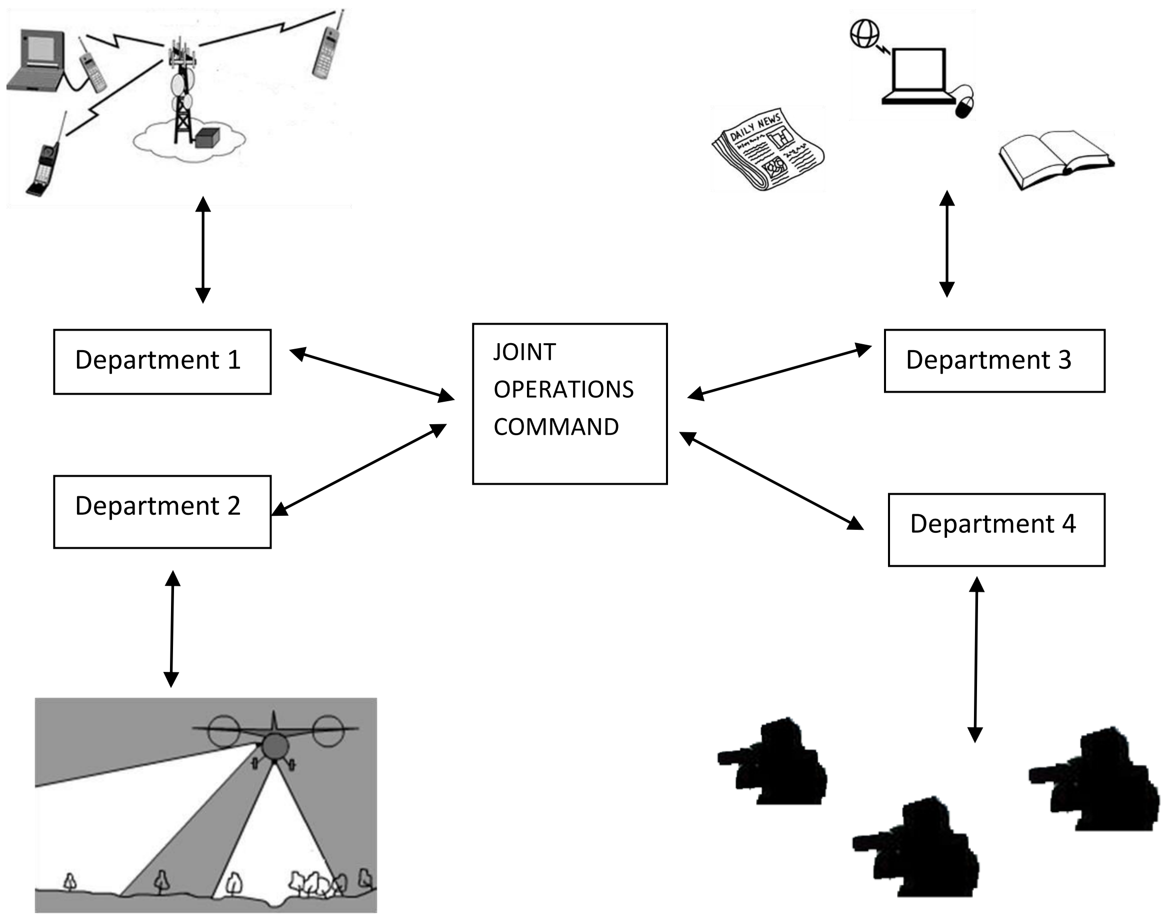


Figure 1: The existing Intelligence department

1.2 Problem statement

Despite of the fact that security intelligence organization has intensified the span of terrorism intelligence collection by implementing numerous intelligence systems, the information collected lacks the capacity to be processed and exchanged quickly for action. The system also lacks ability to detect crimes before they happen and reactivity processing of critical information capability essential to prevent terrorism activities. Many of these systems rely on secure electronic mail instead of databases which makes hard to process and retrieve information. The data is presented in different format and standards making it a complex endeavor to analyze and exchange. This invalidates most of the data collected.

1.3 The purpose of the project

This research expounded on agent based system for monitoring security information. The goal is to provide and exchange timely and sensitive information for prompt action. In addition it provides an information exchange framework that can be adopted by any certified security organ. In this study the agent based system and the preferred methodology were evaluated and the information exchange was developed.

1.4 Objectives of the Study

Project Objective

1) To find out the existing security information monitoring systems, tools and their limitations

System Objectives

1) To design an agent based information monitoring system
2) To develop a prototype
3) To test and implement the prototype
4) To evaluate the prototype

Research Objective

1) To assess the impact of agent based system as interveners of security monitoring

1.5 Significance of the study

This research study develops an information monitoring framework that will assist in collaboration, sharing and exchange of information between security agencies. The study will assist the Department1 section in identifying the critical portion for aerial mapping and surveillance. To the Department2 section, the study will assist in identifying the areas to concentrate in signal interception and signal jamming, to the Department3 section, the study will help in identifying the people, digital equipment, weapon, vehicle etc to research on. To the Department4 section, the study will help in interrogation, foot surveillance and investigations of the suspected terrorists.

1.6 Research outcomes

At the end of this research study, the deliverables will include a prototype that acts as a platform for the exchange of security information amongst the security organs, a project report and research paper that will contribute significantly to the body of knowledge

1.7 Assumptions and limitations of the study

1. The system will not handle exchange of security and intelligence information for all crimes. It will only deal with terrorism crimes and threats.
2. An assumption that all the security agencies are well connected through a national wide area network.
3. An assumption that all the security agencies store their data and information in databases.
4. To solve the security information exchange, one has to rely on data from security agencies, this data is highly confidential, as such this study has relied on anonymous data as we could not obtain real data.

CHAPTER TWO: LITERATURE REVIEW

2.1 Intelligence Monitoring (IM)

According to (Kassel, 2001) Intelligence Monitoring (IM) is a process of discovering hidden truth by collecting clues from open sources such as internet and other sources such as cooperate intranets and document management systems.

IM is applied to many disciplines such as business intelligence, political intelligence, national security intelligence, military intelligence etc.

IM has yielded new concepts beyond search engines, portals, online databases, extranets and news aggregators. The latest technology is (IM) software giving full control to the users as to what to access automatically and what to download for analysis.

2.1.1 Intelligence monitoring steps

(Kassel, 2001) formulates six interrelated steps for intelligence monitoring solution. They begin with mission planning and end up with intelligence report, notification or alarm that form the feedback to the process. The middle process is meant for evaluation and keeping on track.

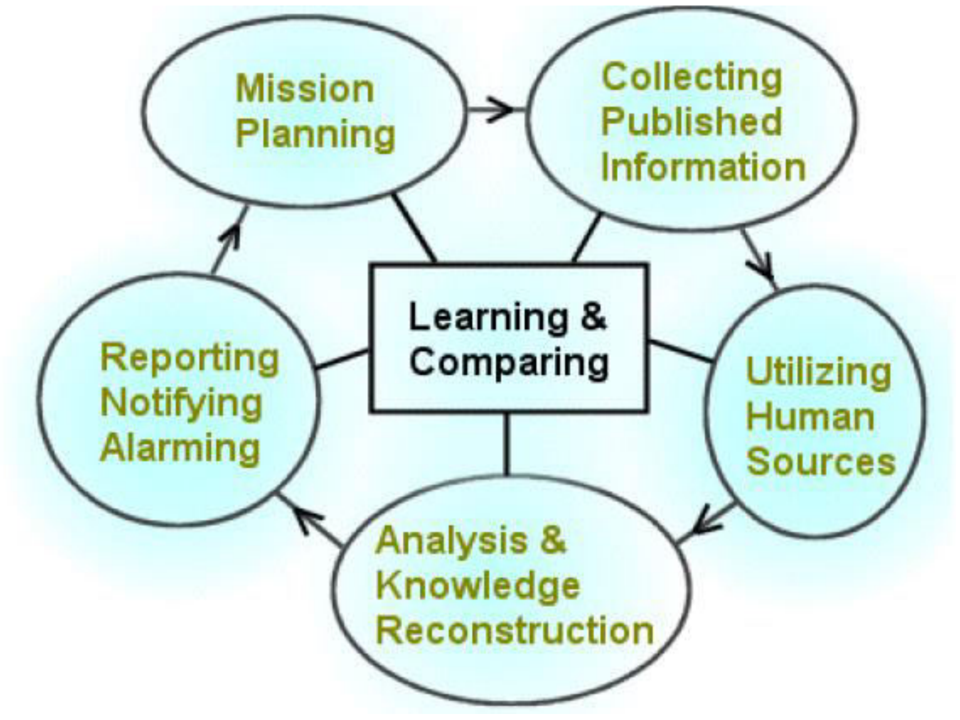

Mission Planning
Collecting Published Information
Learning & Comparing
Reporting Notifying Alarming
Utilizing Human Sources
Analysis & Knowledge Reconstruction

Figure 2: Monitoring cycle (Kassel, 2001)

a) Mission Planning

This step helps in identification of questions that will drive the intelligence gathering phase. If the planning is wrong, the questions will also be wrong leading to tapping the wrong information pool. The intelligence teams with the decision makers are to define the intelligence requirements.

b) Collecting published information

Published information is the information on the internet making (Open Source) it is easily accessible. This information is easily searched through search engines such as Google. A true search goes beyond normal internet sources but to extranets, specialized databases, government filings, journals, local news etc

Intelligence Monitoring Software selection criteria

- The automatic collection of timely information using software agents
- The ability to search from the internet, corporate intranets for information from websites and internal databases.
- The organization of collected information in a manner that would facilitates document retrieval

c) Utilizing human resource

This was the most emphasized step traditionally. Human resource involves interviews, email exchanges, collection of field data by humans. A lot of information remains beyond the research of (IM) software.

d) Analysis and reconstruction of knowledge

This step involves the user converting the collected information into a meaningful assessment to discover both implications and possible outcomes. The analysis methods may include correlation, similarities to known patterns and anomalies. Advanced software packages may also be applied. The main objective of this is to deliver designed outcomes in response to the needs of decision makers using the (IM) program.

e) Reporting, Notification and Alarming

This step involves the delivery and presentation of critical intelligence in a coherent and convincing manner to decision makers. Accuracy and timeliness is very crucial in this stage for the timely and correct decision making. Notification frequency also needs to be properly analyzed and designed.

f) Learning and comparison

In this step, if it is well established, it will provide continuous feedback to other processes and will suggest changes in the overall mission

2.2 Types of information monitored

The scale and type of the exchange of data may vary from one institution to another depending on its scope of work and its responsibilities, custom authorities may emphasis on the exchange of financial data while police authority may more often exchange information and criminal intelligence on persons and vehicles. On the other hand the national authorities exchange all information and intelligence shared from all other authorities. The following categories of data are often exchanged:-

i. Data about persons:- perpetrators, suspects, unidentified persons (name, date of birth, job, residence, fingerprints, verification of personal data, criminal convictions, passports, identity cards, photographs)

ii. Data about vehicles:- vehicles used to transport suspects, perpetrators, vehicles located at the crime scene, record by the surveillance cameras, registration details, owner or operation of the vehicle, chassis numbers, purchase documents, export and import documents

iii. Communication data:- subscribers details (particular for mobile), outgoing or incoming calls, emails, wiretapping and interceptions

iv. Data about objects:- confiscated objects, objects related to crimes are often exchanged while data about firearms (licensing data, lost weapon, weapon used in crimes) and other explanations data

2.3 Real-time system

A real-time system is one that must process information and produce a response within a specified time, else risk severe consequences, including failure. That is, in a system with a real-time constraint it is no good to have the correct action or the correct answer after a certain deadline: it is either by the deadline or it is useless (Srivastava et al, 2012)

Advantages of real time systems for security monitoring

i. The systems are crucial for timely gathering of intelligence information and other activities to prevent and counter acts of terrorism.
ii. The system makes it possible for gathering of evidence required for prosecuting terrorist very speedily.
iii. The systems increase the availability of electronic data online which is compiled and analyzed for counter terrorism purpose
iv. The system are very essential for proactively prevent, detect and deter the terrorists activities.

2.4 Existing systems

2.4.1 Integrated drug monitoring system

Makkai (1999) from the Australia Institute of Criminology conducted a study on drug monitoring system. It emphasized the collection and monitoring of empirical data in a view to link drugs and criminal activities. The study was aimed to show that research precedes intervention mechanisms. It indicates that to move towards evidence based policy making and policing, the collection of the basic research data is essential. The following is architecture of an integrated drug monitoring system.

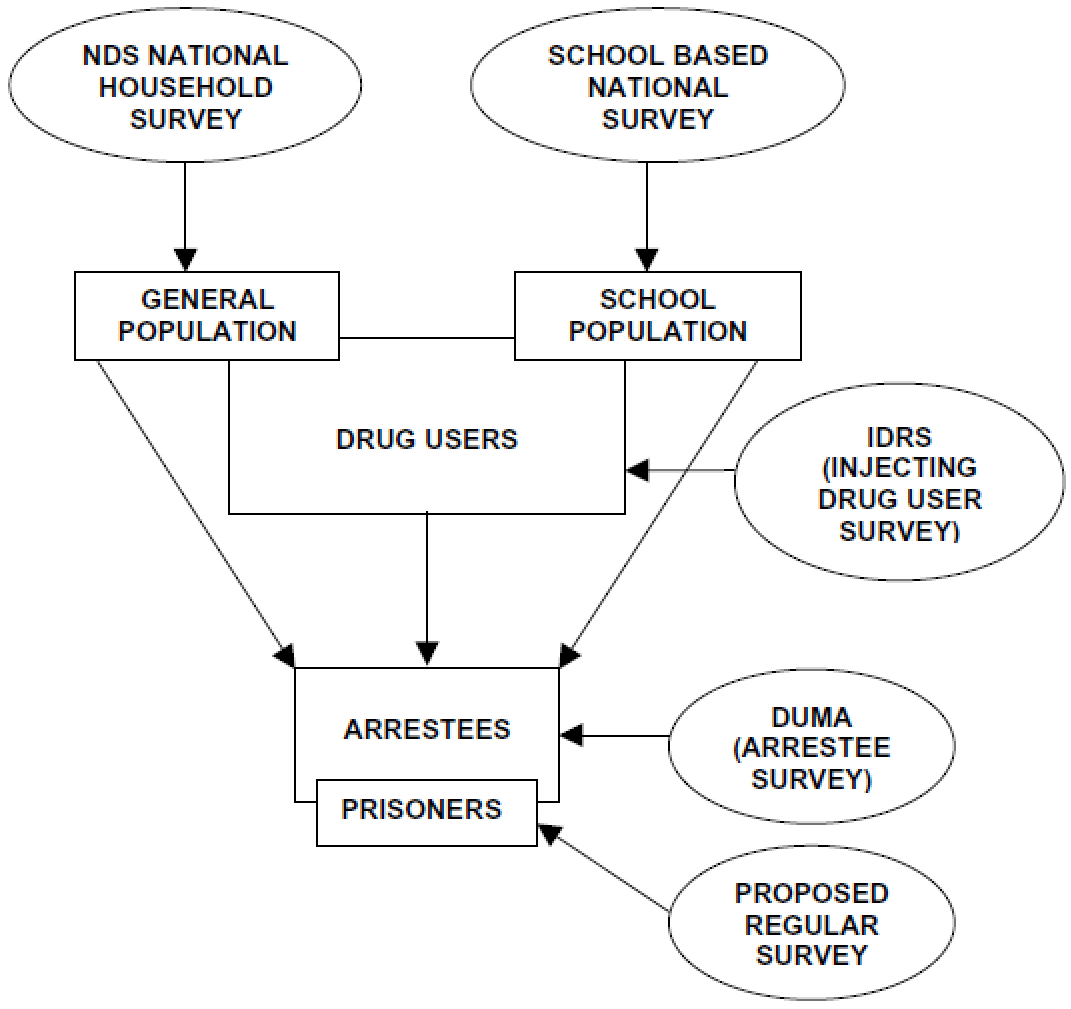


Figure 3: Integrated drug monitoring system – Makkai (1999)

The above architecture shows the way in which data collection system could potentially form an integrated monitoring program providing data to enable national policy initiatives to be developed and evaluated. It can further be decomposed into the following sections:-

NDS – National Household Survey – conducted by market research companies targeting general population which involves face to face interviews and self completion.

School based Survey – targeted young people and involved self completion

IDRS (Illicit Drug Reporting System) – targeted injecting drug users and involves face to face interviews.

DUMA (Drug Use Monitoring in Australia) - targeted arrestees and involved face to face interviews.

Benefits of the drug monitoring system

- ❖ The system will show the level of illicit drug use and its changes over time
- ❖ The factors that differentiate between those who have never used, those who have used but now ceased and those who continue to use.
- ❖ The important risk preventive factors

Weaknesses of the drug monitoring system

Targeting of resources in the criminal justice system to prevent, deter, reduce and control crime requires access to better and more complete data. Further access of real time data for drug monitoring is fundamental for quick action. This system fails to offer that

2.4.2 MNSight – Automatic On-line Media Monitoring

Matiaško, (2011) conducted a study on MNSight. It is a fully automated system and it collects daily over 2000 articles from internet media sources. It is easy to use, cheap and it is designated not only for companies but also for general public. It can also serve not only for monitoring purpose but also as news service which can also be set according to users needs and the areas of user interests.

How it works

For user there exists two interfaces

- ❖ Web based application for setting and viewing real time results
- ❖ Easy portable static html report and email address to which it will be sent. A report contains all matched articles clearly ordered by category and time.

MNSight System Architecture - Matiaško, (2011)

It contains three parts namely:-

Web server – responsible for users and admin web pages access

Database – maintains user data and collected articles

Services – media crawler and media services

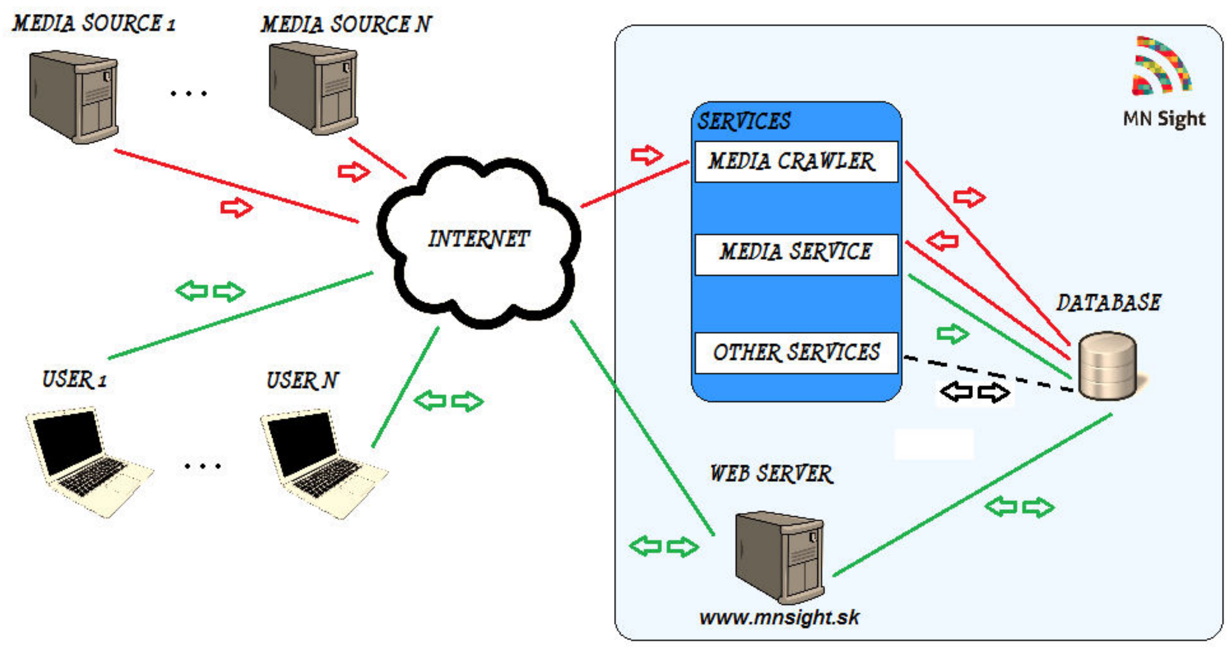


Figure 4: MNSight System Architecture - Matiaško, (2011)

Weaknesses of MNSight System

The system mainly deals with collection of information from the internet. It fails to integrate with other information sources such as human, radio, television

2.4.3 An automatic system for monitoring police records for a crime profile

Brown, (2001) studies an automated system that monitors police records on an ongoing basis for matches to predefined crime profiles. It notifies the police officer or a group of police officers when a match of the profile appears.

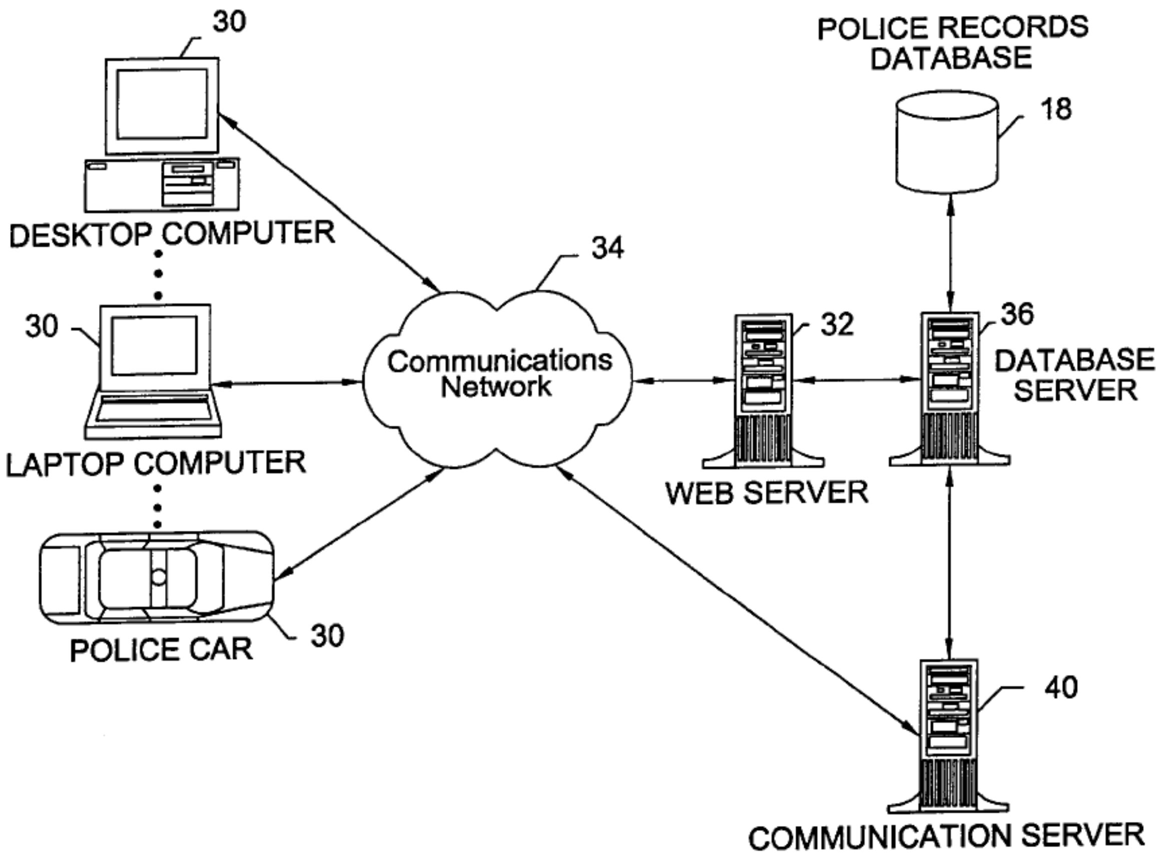


Figure 5: An architecture of system for monitoring police records for a crime profile – Brown, (2001)

The user interface device – a client side devices provided to execute the user interface module. These includes desktop computers, laptop computer, police cars etc

Communication network – an infrastructure under which notifications such as emails is sent to the police officer

Web server – computers that store WebPages

Police record databases – the system that stores police records

Database server – computer under which police database system are stored

Communication server – computer that facilitates communication of other network devices.

2.5 Agents

According to (Brenner et al, 1998) an agent is a software or software / Hardware that is autonomous (act relatively independently) and is characterized by: -

- *autonomy* - agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state
- *social ability* - agents interact with other agents (and possibly humans) via some kind of agent-communication language
- *reactivity*: agents perceive their environment and respond in a timely fashion to changes that occur in it
- *pro-activeness*: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking initiative.
- *Mobility* – ability to move around network platforms
- *Veracity* – avoid communicating false information knowingly
- *Rationality* – act in order to achieve its goals subject to belief
- *Personality* – have distinct behavior, name and role

2.6 Agents Architecture

(Maes, 1991) defines agent architecture as a particular methodology for building agents. This section tries to explain how the construction of agents can be decomposed into a construction of a set of component modules and how those modules should be made to interact

There are three main agent architecture namely:- Symbolic, reactive and hybrid agent architecture.

2.6.1 Symbolic reasoning

Deliberate agent architecture is the one that:-

- Contains explicitly represented, symbolic model of the world
- Make decisions (for example about what action to perform via symbolic reasoning)

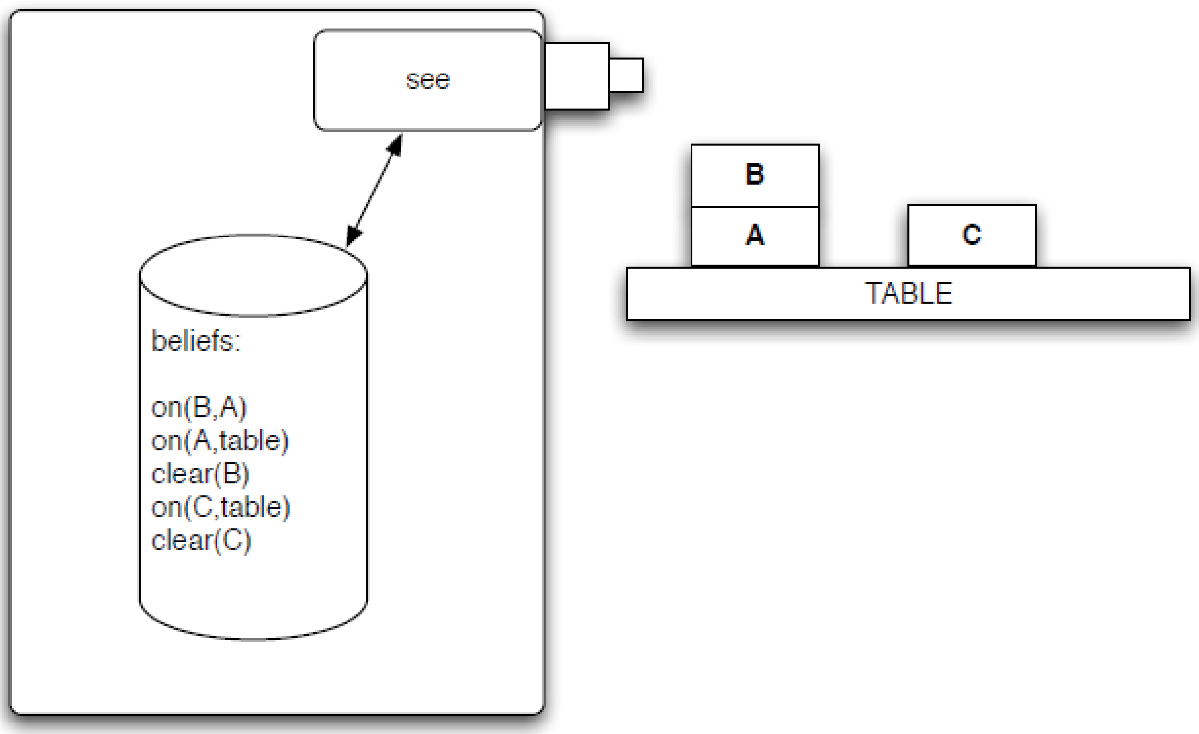


Figure 6: Representing the world symbolically –(Maes, 1991)

Challenges of Symbolic architecture

i. The transduction problem: that of translating the real world into an accurate, adequate symbolic description, in time for that description to be useful.
ii. The representation/reasoning problem: that of how to symbolically represent information about complex real-world entities and processes, and how to get agents to reason with this information in time for the results to be useful.

How to overcome these challenges

i. Weaken the logic
ii. Use symbolic, non logical representations
iii. Shift the emphasis of reasoning from runtime to design time

2.6.2 Reactive architecture

They gain their intelligence from interacting with their environment. They have task specific modules that initiate direct reactions in response to specific situations that occur in the environment. These architecture increase fault tolerance and robustness of an agent

It was researched in order to solve the problem of symbolic AI. Rodney Brooks criticizes symbolic AI by putting forward three theses:-

i. Intelligence behavior can be generated without explicit representation of the kind that symbolic AI proposes.
ii. Intelligence behavior can be generated without explicit abstract reasoning of the kind that symbolic AI proposes.
iii. Intelligence is an emergent property of certain complex system.

2.6.3 Hybrid architectures

It has been argued by many researchers that neither a completely deliberate nor completely reactive approach is suitable for building agents. They propose hybrid systems.

This builds agent based on two subsystems namely:- deliberate and reactive one. The combination of two leads to a layered architecture.

Horizontal layering – layers are each directly connected to the sensory input and action output.

Vertical layering – sensory input and action output are each dealt with by at most one layer each

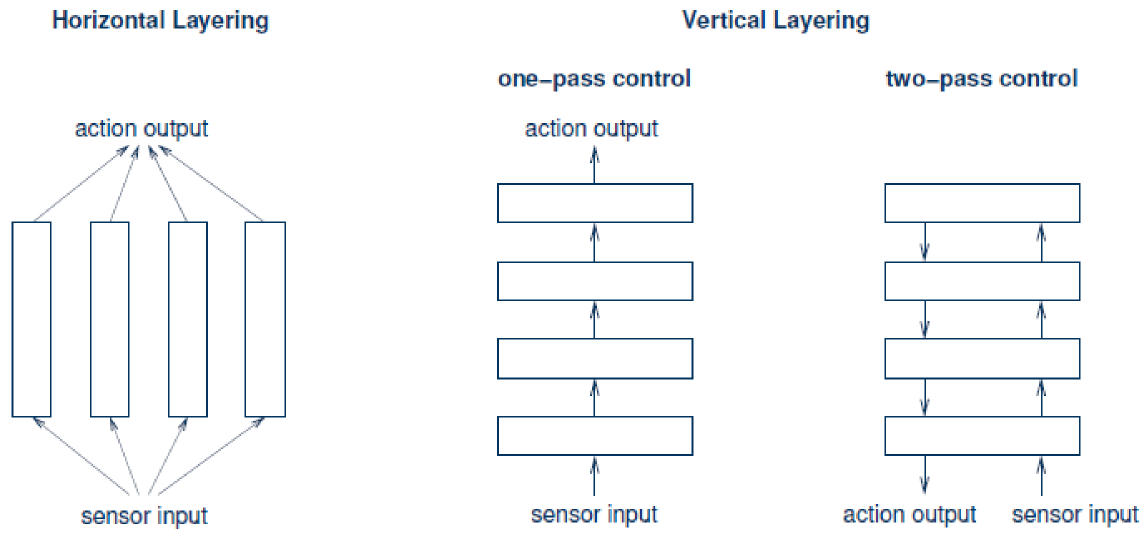


Figure 7: Agents architectures – Muller et al (1995)

2.7 Multi agent system (MAS)

(Wooldridge, 2002) defines multi agent system as a system of agents which interacts with one another through cooperation, competition, coordination or negotiation to accomplish some goals. (Sycara, 1998)describes MAS as a system of several agents. It is an organization of coordinated autonomous agents that interacts in order to achieve a common goal (Georgini et al, 2001)

This research proposal identifies Multi agent based technology as a tool for implementing a security monitoring system. This is because MAS are able to model very complex and inherently distributed systems.

Advantages of multi-agent systems

- ❖ *Speed and efficiency* – multiple agents can break a task into parallel tasks and perform them simultaneously
- ❖ *Robustness and reliability*- interconnected agents can perform their roles ensuring that there is no single point of failure
- ❖ *Scalability and flexibility*- a system made of agents can grow and increase in size without jeopardizing its functionality and control

- *Reusability/ cost* – single agent developed, roles assigned; replicable
- *Suitability for distributed* environment

2.8 Agent's methodology

According to (Giorgini, 2005)an agent methodology is a body of methods employed in a discipline. It is a procedure of attaining something. There are certain cases where applying agents will be appropriate which includes:-

i. Open, dynamic, uncertain or complex environments
ii. Organization with distributed functions, intelligence interfaces
iii. Data, control or expertise is distributed such as database systems with different autonomous ownership
iv. Legacy system where interfaces to old systems are important

2.9 Genealogy of agents

Agent oriented methodologies have several roots. Some are based on the artificial intelligence. Others based on existing OO methodologies. Other tries to merge the two approaches. The diagram below show these lineages and influence in what might be called a genealogy of the ten AO methodologies.

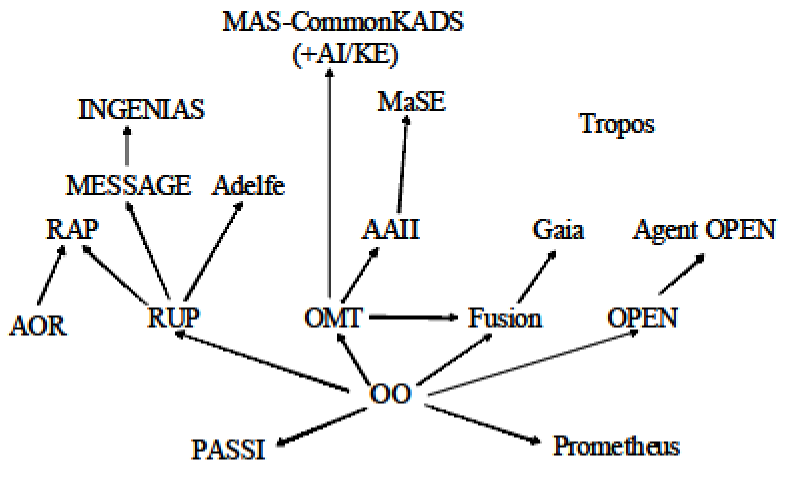


Figure 8: Direct and indirect influences of object oriented methodologies of an agent oriented methodology (**James Odell (2005)**)

Features of agent methodology (Brian Henderson- Sellers, Paolo Girogini (2005).

- ❖ Provide sufficient abstractions to fully model and support agents and MASs—arguably
- ❖ Should focus on an organized society of agents playing roles within an environment
- ❖ Support MAS, where agents interact according to protocols determined by the agents' roles
- ❖ Should be "agent-oriented" in that it is geared towards the creation of agent-based software

2.10 The Prometheus methodology (Australia)

This study identifies Prometheus methodology as a means of specifying and designing the agents. The Prometheus methodology consists of three phases which includes :-

i. *System Specification:* where the system is specified using goals and use case scenarios; the system's interface to its environment is described in terms of actions, percepts, and external data; and functionalities are defined.

ii. *Architectural design:* where agent types are identified; the system's overall structure is captured in a system overview diagram; and use case scenarios are developed into interaction protocols.

iii. *Detailed design:* where the details of each agent's internals are developed and defined in terms of capabilities, data, events, and plans; process diagrams are used as a stepping stone between interaction protocols and plans.

The diagram below describes the stages of Prometheus methodology.

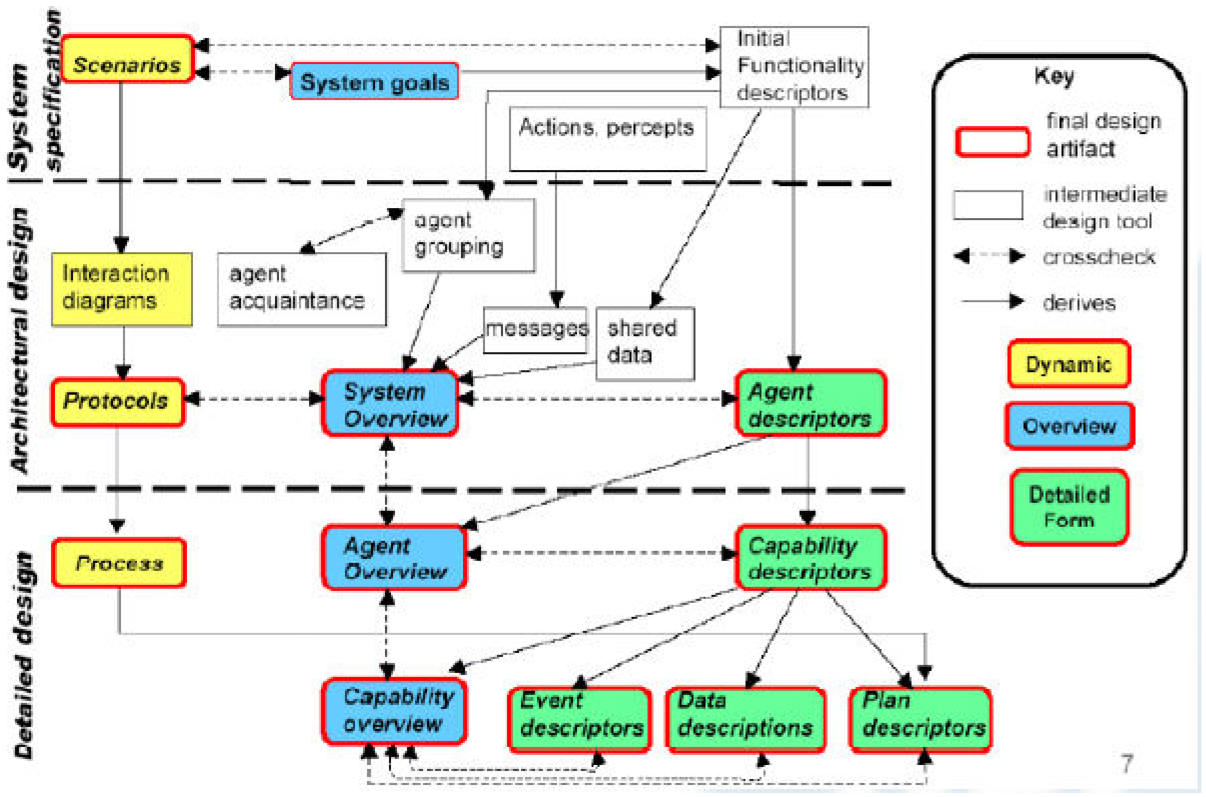


Figure 9: Stages of Prometheus methodology – Winikoff and padgham (2005)

2.11 RELATED WORK

RETSINA (Reusable Task structure based Intelligent Network Agent).

This is a multi-agent architecture that was developed by Carnegie Mellon University Robotic Institute. This architecture supports communities of heterogeneous agents. It implements distributed services that facilitates the interaction between agents, as opposed to managing them. This architecture has been applied to many disciplines. (K S Decker, 1997)

It consists of four main agents type:-

i. *Interface agents* - interact with users, receive user input, and display results.

ii. *Task agents* - help users perform tasks, formulate problem-solving plans and carry out these plans by coordinating and exchanging information with other software agents.

iii. *Information agents* - provide intelligent access to a heterogeneous collection of information sources.

iv. Middle agents - help match agents that request services with agents that provide services.

Each RETSINA agent has four reusable modules for communicating, planning, scheduling and monitoring execution of tasks and requests from other agents

- ❖ The *Communication and Coordination* module accepts and interprets messages and requests from other agents.
- ❖ The *Planning* module takes as input a set of goals and produces a plan that satisfies the goals.
- ❖ The *Scheduling* module uses the task structure created by the planning module to order the tasks.
- ❖ The *Execution* module monitors this process and ensures that actions are carried out in accordance with computational and other constraints.

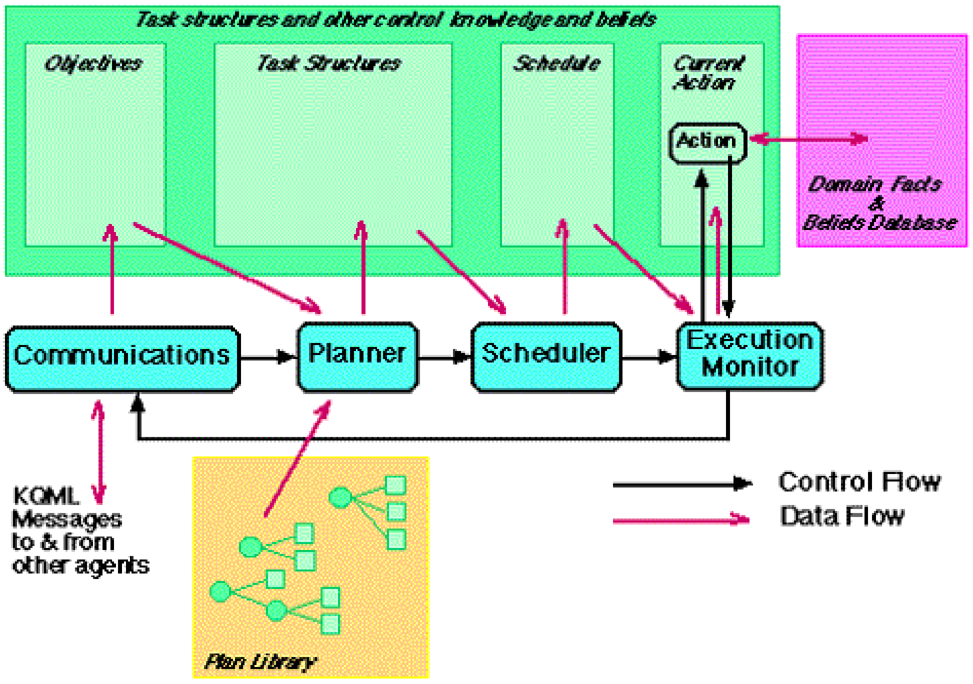


Fig10. A graphic representation of the RETSINA agent architecture – Decker (1997)

2.12 Agent development Technology

This research uses JADE platform to design multi agent. JADE is the abbreviation for the Java Agent Development Framework and has been developed by the Telecom Italia Lab (TILAB) in Italy, in compliance with the FIPA (Foundation for Intelligent Physical Agents) specifications. FIPA is a non-profit organization geared at producing standards for the interoperation of heterogeneous agents. Essentially, JADE is a middle-ware (written entirely in the Java language, using several Java technologies), which simplifies the implementation of multi-agent systems by providing a set of graphical tools that support the debugging and deployment phases. (Giovanni et al, 2006) says that the agent platform can be distributed across multiple machines, regardless of the underlying operating system, and the configuration controlled via a remote graphical user interface. By specifically focusing on the JADE platform in the design phase, the designer can move straight to implementation afterwards, without having to tediously adapt the results of the design phase to an agent platform of their choice. This will obviously result in significant time gains for the designer, in addition to providing them with a much clearer picture on how to progress in implementation.

2.13 Conceptual frameworks

The agent based security monitoring system provides critical information in a timely manner to users. It also provides a common data repository. The model has the following agents:-

i. Graphical User Agent
ii. User Agent
iii. Database Agent
iv. Communication Agent

The figure below shows the conceptual framework of the system. Each department has an agent namely Department 1 Agent, Department 2 Agent, Department 3 Agent and Department 4 Agent. These agents are responsible for filtering and categorizing information relevant to it. The database agent is responsible for validating and saving the categorized information into the common repository. It is also responsible for Saving user data and credentials into the database. Communication agent is responsible for receiving information from the common repository, package it and send it to all relevant departments and to the users email. The user agent is responsible for authenticating all the users of the system.

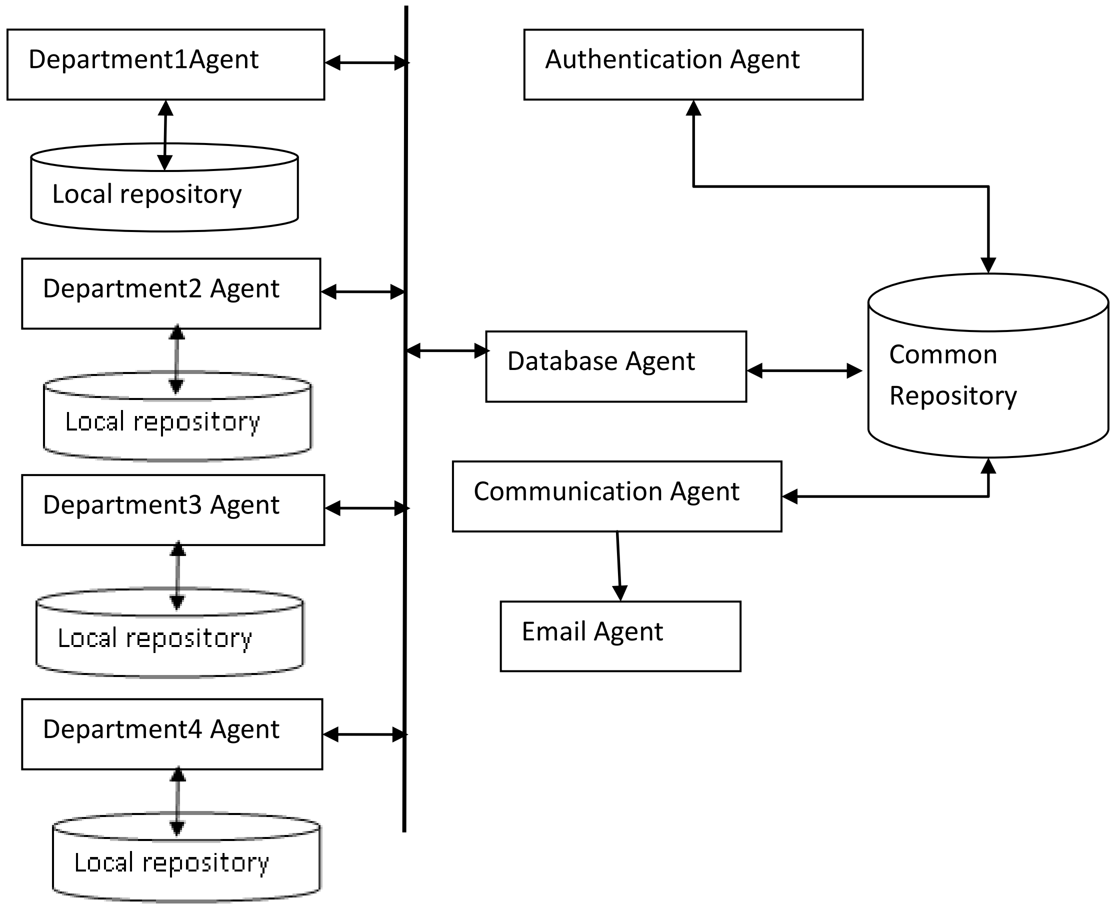


Figure 11: Conceptual framework

CHAPTER 3: METHODOLOGY

3.1 Introduction

In this study various steps were carried out in order to achieve the overall objectives set out in chapter one. The steps are as listed below:-

i. System specification
ii. Architectural design
iii. Detailed design

3.2 Multi agent system methodology

This study was guided by Prometheus agents methodology (Winikoff, 2005). It was adopted in development of security monitoring agents. It achieved this by specifying requirements, agent-oriented system design and implementation. It offers detailed guideline to specify, implement, test and debug agent based software systems. This methodology is complete and practical oriented. It provides support to and not limited to agents based on goals and plans to realize agents that are flexible and robust. The methodology was designed to facilitate tool support and has proved to be useful in industry and academia.

This methodology is composed of three phases i.e. system specification, architectural design and detailed design. Each of these phases includes models that focuses on the dynamics of the system, graphical models that focus on the structure of the system or its components and textual descriptor forms that provides details for individual entities. The following entails the three phases in details and activities involved.

3.2.1 System specification – This phase consists of the following activities:

- ❖ Identify system goals and sub-goals
- ❖ Develop use case scenarios
- ❖ Identify functionalities
- ❖ Identify data read and written by functionalities
- ❖ Identify the agent system's interface to the environment in terms of actions, percepts, and external data

- Prepare functionality schemas (name, description, actions, percepts, data used/produced, interaction (with other functionalities), and goals)

3.2.2 Architectural Design - This phase consists of the following activities:

- Group functionalities to determine agent types using data coupling and agent acquaintance diagrams to assess alternative groupings
- Define agent types (also define the number and life-cycle of the agent types) and develop agent descriptors
- Produce a system level overview diagram describing the overall structure of the system
- Develop interaction protocols from use case scenarios (via interaction diagrams)

3.2.3 Detailed Design – This phase consists of the following activities:

- Develop process diagrams
- Produce agent overview diagrams showing the internal workings of agents in terms of capabilities, events, data and plans
- Refine capability internals (add included capabilities and interactions)
- Introduce plans to handle events
- Define details of events (external, between agents, between capabilities and within agents)
- Define details of plans (relevance, context, sub goals)
- Define details of beliefs/data

3.3 Data sources

Data sources form one of the major limitations of this study. National security data is very confidential, protected and guarded at all costs. As such it is near to impossible to use real data. This study relied on dummy data. This data was supported by observation of activities conducted within the organization.

3.4 Data collection tools

Observations and interviews was used as primary data research instruments while secondary data tools was journals, books, newspaper, government publications etc.

3.5 Data analysis

This study employed qualitative data analysis. Microsoft Excel spreadsheet or statistical package SPSS was used as the tool for the data analysis.

3.6 System implementation

The implementation of the security monitoring system was done using JADE. The graphical User Interface was WSIG add-on for Jade framework.

CHAPTER FOUR: ANALYSIS DESIGN AND IMPLEMENTATION

The Prometheus methodology was adopted for System specification, System design and implementation.

The Prometheus methodology is a detailed process for specifying, designing, and implementing intelligent agent systems.

4.1 System specification

This is the first stage of Prometheus methodology for building Multi Agent Systems. it involved specifying system requirements using goals and scenarios. Various tools were used such as use case scenarios and system requirement identified such as the System Inputs, The processes and the System Output.

4.2 System Goals

What is the system is build for?

Overall Goal

The overall goal of the system is to record intelligence into the database as soon as it happens and disseminates the same intelligence to the people interested immediately.

Goal Diagram

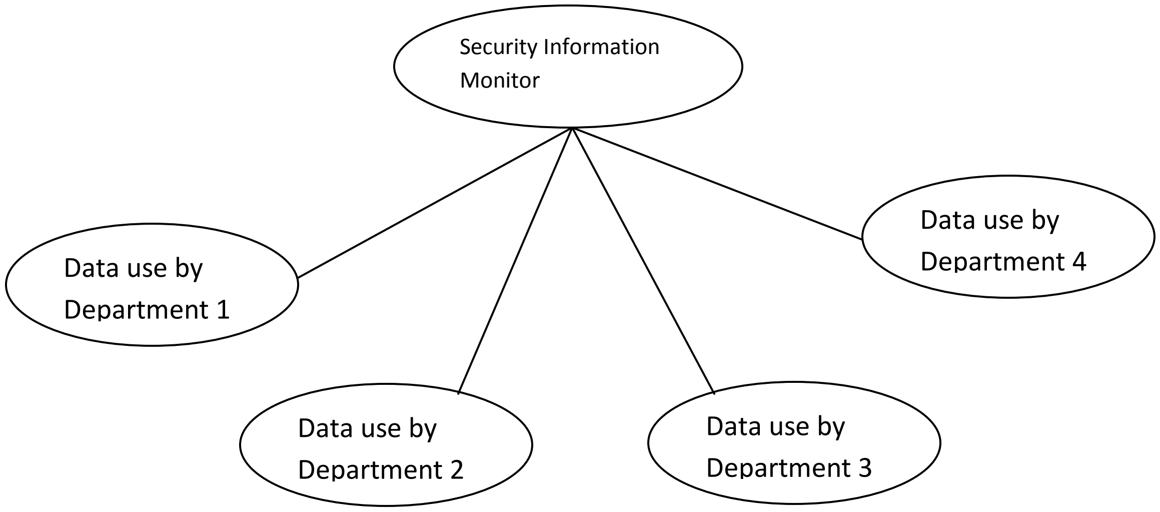
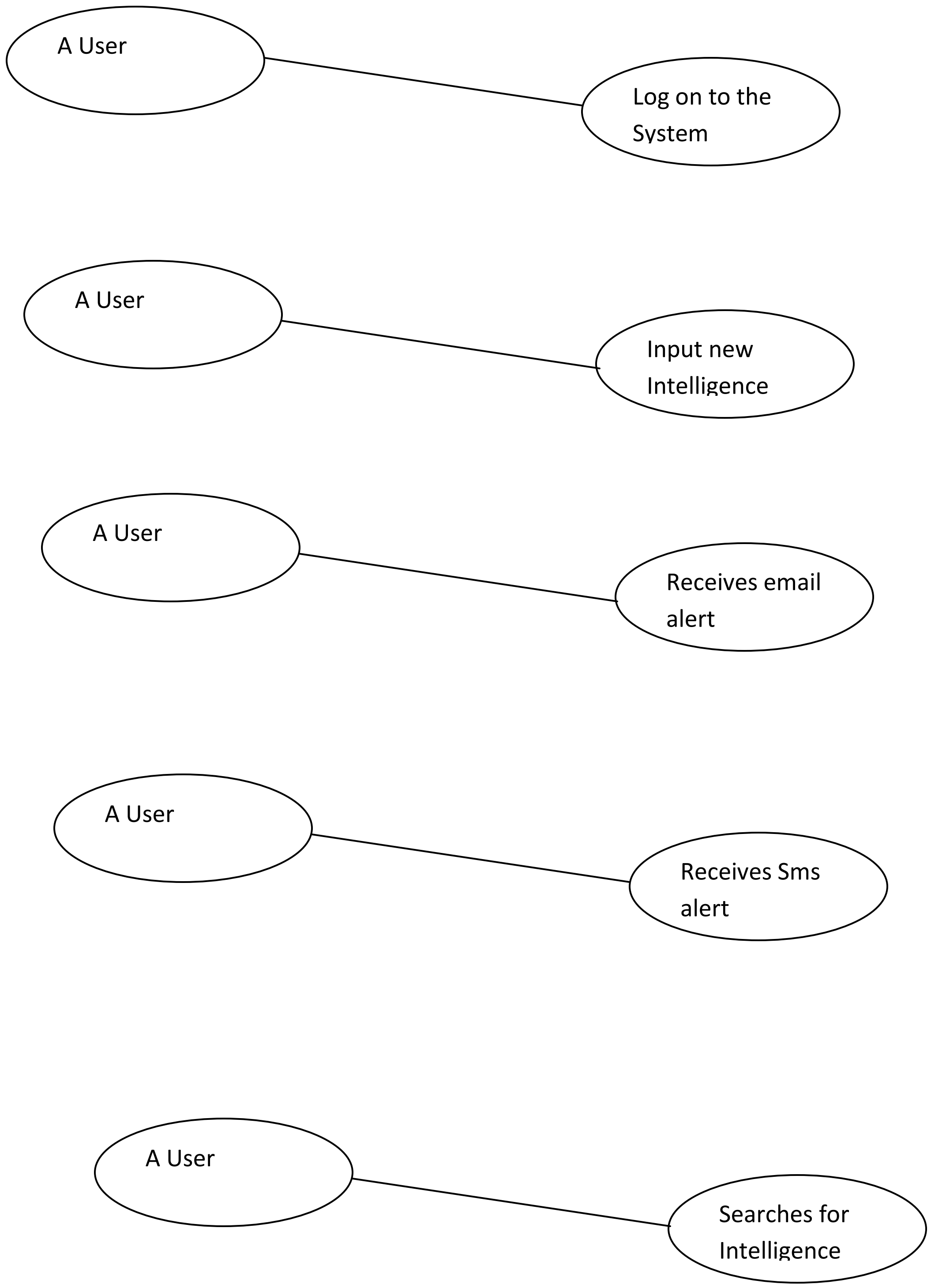


Figure 12: Goal diagram

Subsidiary goals

The overall goal was broken into smaller goals (Sub goals). A goal is a state of affair that is achieved by an actor.

Figure 13: Sub goals



4.2.1 Use case scenarios

Use case scenarios represents a particular instance of the system without branching (Michael Papasimeon and Clinton Heinze, 2000). Use case analysis was used for the system requirement specification that describes a view of the functionality of a system.

Use Case1: The Users or departments wishing to get updates on security intelligence information must subscribe to the system in order to receive alerts. They must feed the system with their Mobile numbers and Email Address.

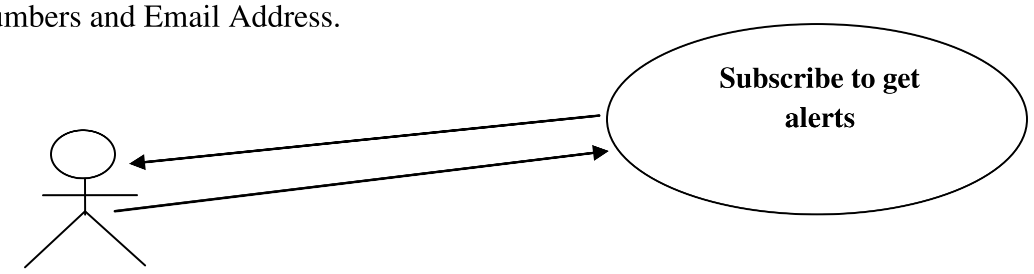


Figure 14: User subscribing

Use Case 2: The User of the system log on to the MAS using right credentials and input the collected security intelligence from various means.

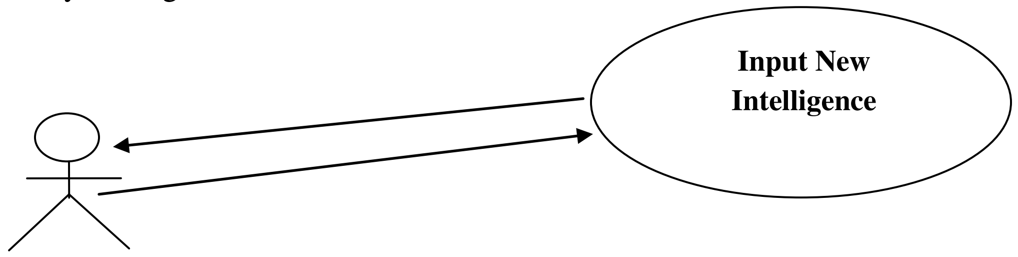


Figure 15: Intelligence Officer

Use Case 3: The user searches for Security intelligence using pre defined parameters

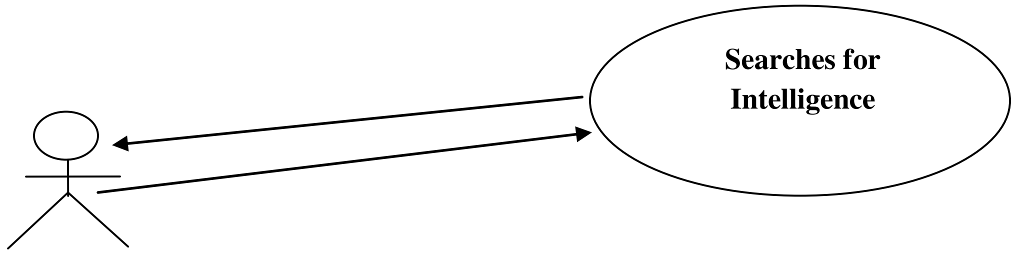


Figure 16. Intelligence Officer

4.2.2 Identification of the system interface to the environment

The system interface to the environment was defined in terms of actors, percepts and external data.

Actors

The main actors in the security intelligence system are:-

Intelligence Officer

The intelligence officer obtains intelligence using different means. The officer captures the intelligence into the system. The information capture includes:- intelligence name, report date, location, people involved, equipment involved, communication gadget, description and reporter.

Subscriber

The user of the system who has subscribed to receive alerts gets intelligence that is filtered to the specific intelligence officer's role.

Percepts

The percepts includes an individual security officer obtaining intelligence through different technical means, captures it into the multi agents system, other officers searching intelligence from the system and users subscribing and unsubscribing from the system.

Actions

Actions includes sending of emails alerts to subscribed users, sending of short messages alerts to subscribed users and processing subscription.

4.2.3 System functionalities

The system functionalities identified includes:- Registration of intelligence, registration of subscribers, sending of email and Sms alerts to subscribers and searching of Security Intelligence from MAS

a) Registration of Security Intelligence.

Name: Intelligence registration

Description: Register new Security Intelligence into the system

Percepts/ events / messages: Security Intelligence registered

Message send: Registration completed successfully

Action: Display customized message

Data used: Data supplied by Intelligence officer

Interactions: Information agent via dissemination agent

b) Subscription

Name: Subscription

Description: Register users so as to get alerts

Percepts/ events / messages: Successful subscription (Customized Message)

Message send: Subscription done

Action: Subscribe / Unsubscribe

Data used: Subscriber details

Interactions: Dissemination agent via subscription

c) Sending alerts

Name : Sending alerts to subscribers

Description: Alert subscribers that there is a new intelligence registration.

Percepts/events/messages: new Security Intelligence record in DB(customized message)

Message sent: new intelligence registration occurrence (customized message)

Actions: Display the record

Data used: Intelligence DB, Subscribers' details

Interactions: dissemination agent via sms alerts

d) Searching the Existing Security Intelligence Records

Name: Search record of registered intelligence

Description: perform search

Percepts/ events / messages: record found in DB (Customized message)

Message send: search results

Action: Display record

Data used: Intelligence DB, subscribers DB

Interactions: dissemination Agent via search

4.3 ARCHITECTURAL DESIGN

This is where agent types are identified; the system's overall structure is captured in a system overview diagram; and scenarios are developed into interaction protocols.

4.3.1 Determining agent type

Agent types are identified from the system functionalities based on consideration on coupling. These are explored using a coupling diagram and an agent acquaintance diagram. Once a grouping is chosen the resulting agents are described using agent descriptors.

The functionalities as mentioned earlier includes:- Registration of intelligence information, registration of users, sending of email and SMS alerts to subscribers, search of Intelligence. The agents required to achieve the above functionalities includes:- GUI Agent, Authentication agent, database Agent, communication Agent, Dissemination Agent

Data coupling Diagram

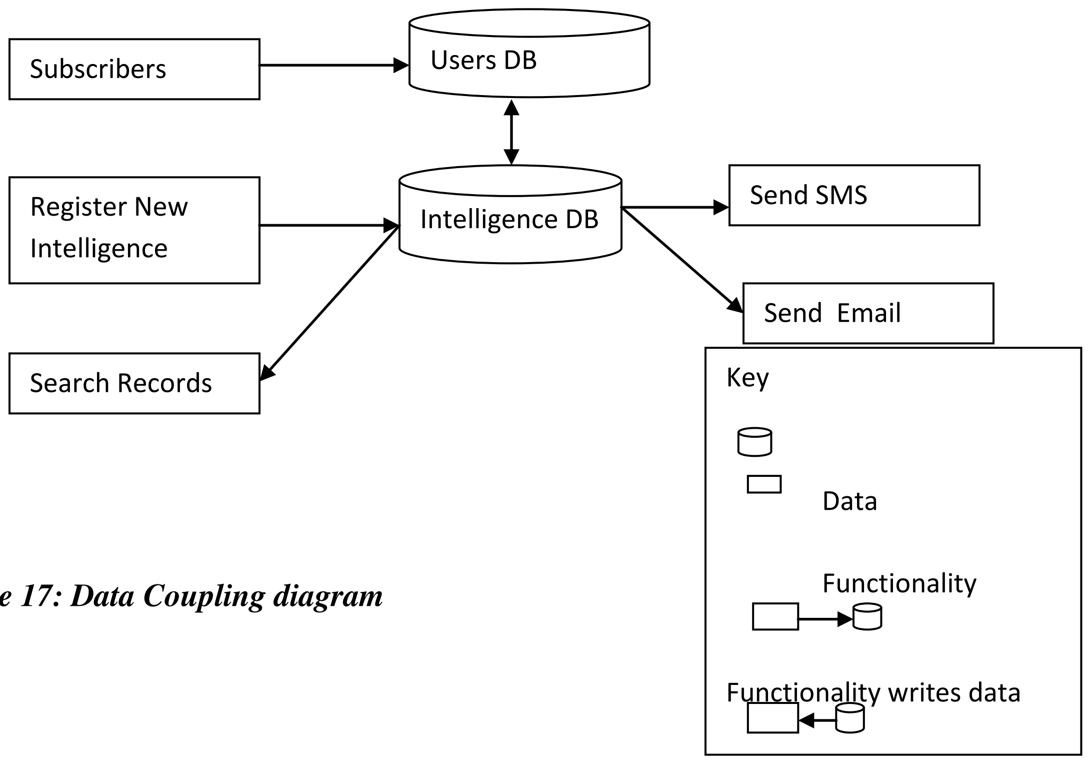


Figure 17: Data Coupling diagram

4.3.2 Agent Descriptors

The **agent descriptors** for the identified agents are described below.

Name: GUI Agent – Container:control

Description: receive all requests from users of the web interface and sends the responses back after agents have worked on them. Directs the requests to the appropriate agent(s)**.**

Lifetime : instantiated when system starts.

Initialization: Reads user input

Demise: Closes requests

Functionalities included: Registration, search, sms sending , email sending

Uses Data: Registration DB, SubscribersDB, usersDB

Goals: respond to web requests, direct requests to the appropriate agents, output results.

Events responded to: new registration record,search request, sms and email alerts.

Actions: display customized responses to search requests and successful entry of new records.

Name: Database Agent

Description: receive data from the WSIG control container

Lifetime : instantiated when a new registration is entered into the system

Initialization: receives data from WSIG container

Demise: Closes requests

Functionalities included: Registration,

Uses Data: registration data captured

Goals: receive message from WSIG , return response to WSIG,

Events responded to: new registration record

Actions:

Interacts with:WSIG, Data processing agent

Name : Communication Agent

Description: Responsible for general communication tasks.;SMS and email agents work under instruction from this agent.;Converts data objects into SMS and email objects.;Validates SMS email and data object data;Sends requests to sms and email agents

Lifetime: Instantiated on receipt of SMS or Email request. Demise when a user logs out

Initialisation: receives request, reads Intelligence DB

Demise: on close of DB connections

Functionalities included: sending sms, sending email

Usesdata: Intelligence DB,

Events responded to:

Actions: convert data into sms and email objects. Interacts with: sms sending, email sending , record searching

Name : Email agent

Description: Responsible for retrieving valid email address from the user table, send email

Lifetime: Instantiated on receipt of email object, demise when the application closes

Initialisation: receive email object

Demise: on close of DB connections

Functionalities included: sending email

Usesdata: user table, Intelligence DB

Events responded to: new email object

Actions: display email alert

Name : SMS agent

Description: Responsible for retrieving valid sms recipient from the user table, send sms

Lifetime: Instantiated on receipt of sms object, demise when the application closes

Initialisation: receive sms object

Demise: on close of DB connections

Functionalities included: sending sms

Usesdata: user table, Intelligence DB

Events responded to: new sms object

Actions: display sms alert

4.3.3 Interaction Diagram of the System

Figure 4.9i

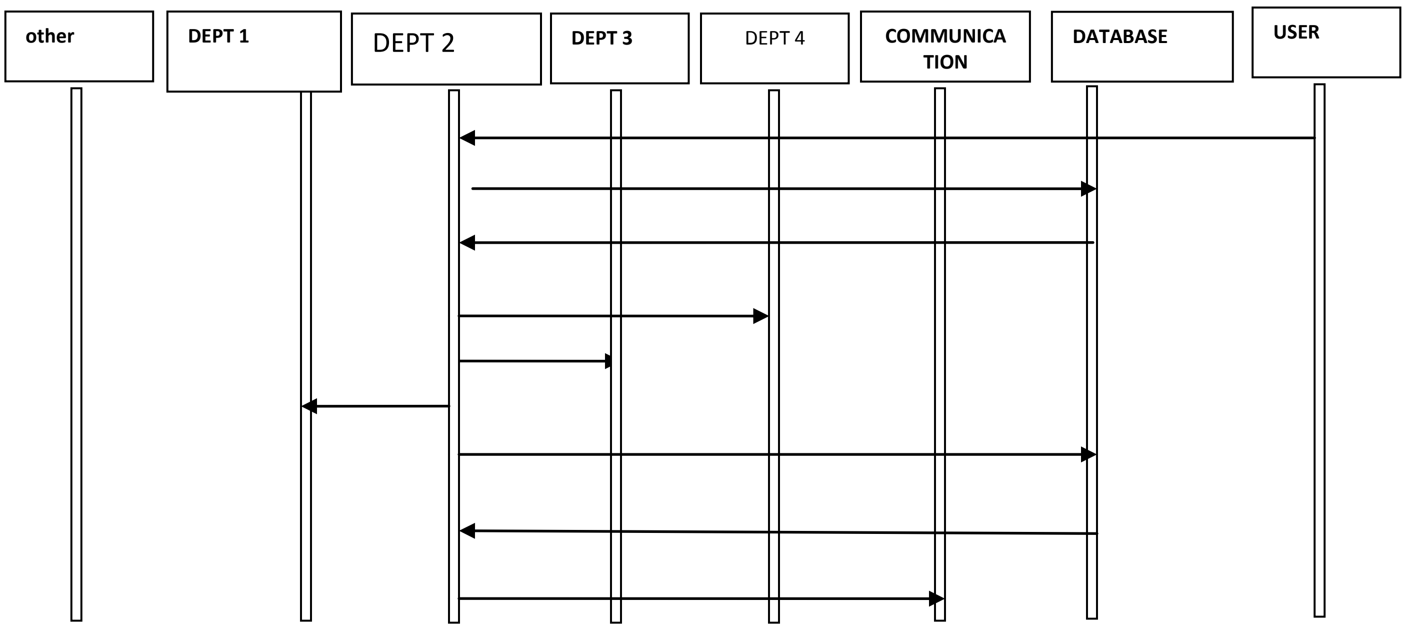


Figure 18: Interaction diagram

4.4 DETAILED DESIGN

the following were undertaken in this phase:-

agents internal including capabilities were developed, agents overview diagram and capability descriptors were used

details of capabilities in terms of other capabilities as well as events, plans and data were developed. this was done using capabilities overview diagrams and various descriptors. the main focus is to develop plan in order to achieve the set goals.

4.4.1 Agents Capabilities.

Each agent was analyzed and their capabilities were identified as follows:-

S/NO	**AGENT**	**CAPABILITIES**
1	Authentication Agent	Validates the authentic users as in the user table
2	Gui Agent	Interface between the web and other Agents
3	Communication Agent	Receives data and validates it into the database
4	Database Agent	Ensures that the data is updated and saved into the Database
5	Dissemination Agent	Used for communication purposes and sends request to sms and email agents
6	Email Agent	Email sending, validate email recipients from User Db
7	Sms Agent	Sending Sms, validates sms subscribers from user Db
8	Search Agent	Perform data search

4.4.2 Capability descriptor

The capability descriptor has been generated below:-

Name - search

External interface to the capability - user enter search string

Natural language description - produce search results

Interaction with the other capabilities - information agent capability

Data used / produced by capability - search result

Inclusion of other capability - none

Name - email sending

External interface to the capability - found new record

Natural language description - send email message to subscribed users

Interaction with the other capabilities - dissemination capability

Data used / produced by capability - email object

Inclusion of other capability - none

Name - email sms

External interface to the capability - found new record

Natural language description - send sms to subscribed users

Interaction with the other capabilities - dissemination capability

Data used / produced by capability - sms object

Inclusion of other capability - none

Name - information agent

External interface to the capability - data entry

Natural language description - enter data into the Intelligence Db

Interaction with the other capabilities - dissemination agent

Data used / produced by capability - Database record

Inclusion of other capability - none

Agent overview diagram

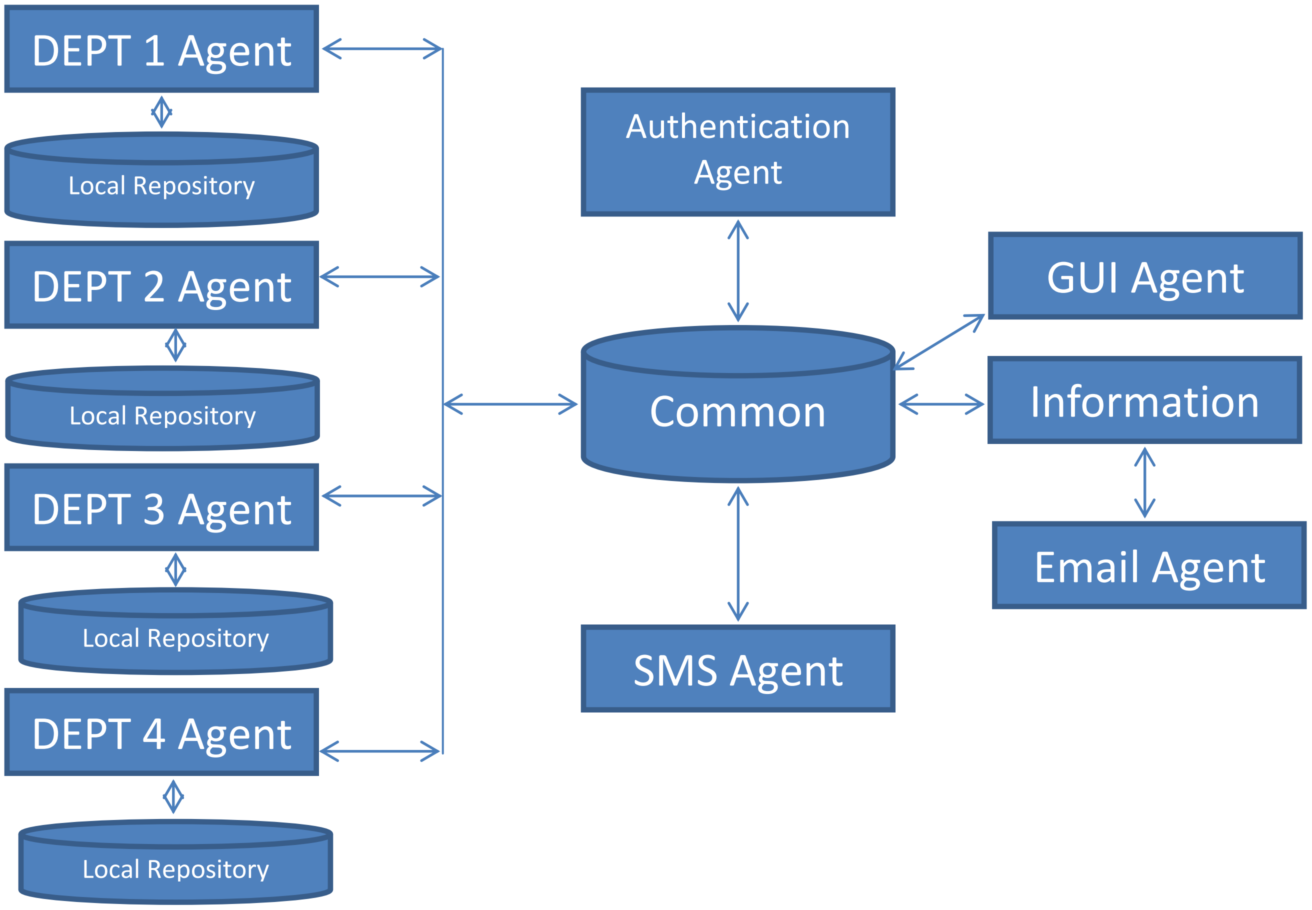


Figure 19: Agents overview diagram

4.5 Chapter Summary

The overall goal of the multi agent system is to record security intelligence into the database as soon as it happens and disseminates the same intelligence to the people interested immediately.

The overall goal was broken into several sub goals that were identified as logging and authentication into the system, input of new intelligence into the system, sending of email and sms alerts, search of new records.

The use case scenarios were identified as tools for identification of system functionalities. Identification of the system interface to the environment was done using actors, percepts and external data from the system functionalities.

Several agent types were identified which includes Gui Agent, Repository Agent, Dissemination Agent, Email Agent, Sms Agent, Authentication Agent.

Data coupling diagram was developed using functionalities defined and data. Interaction diagram was developed describing how agents interact with each other.

CHAPTER FIVE: SYSTEM TESTING AND IMPLEMENTATION

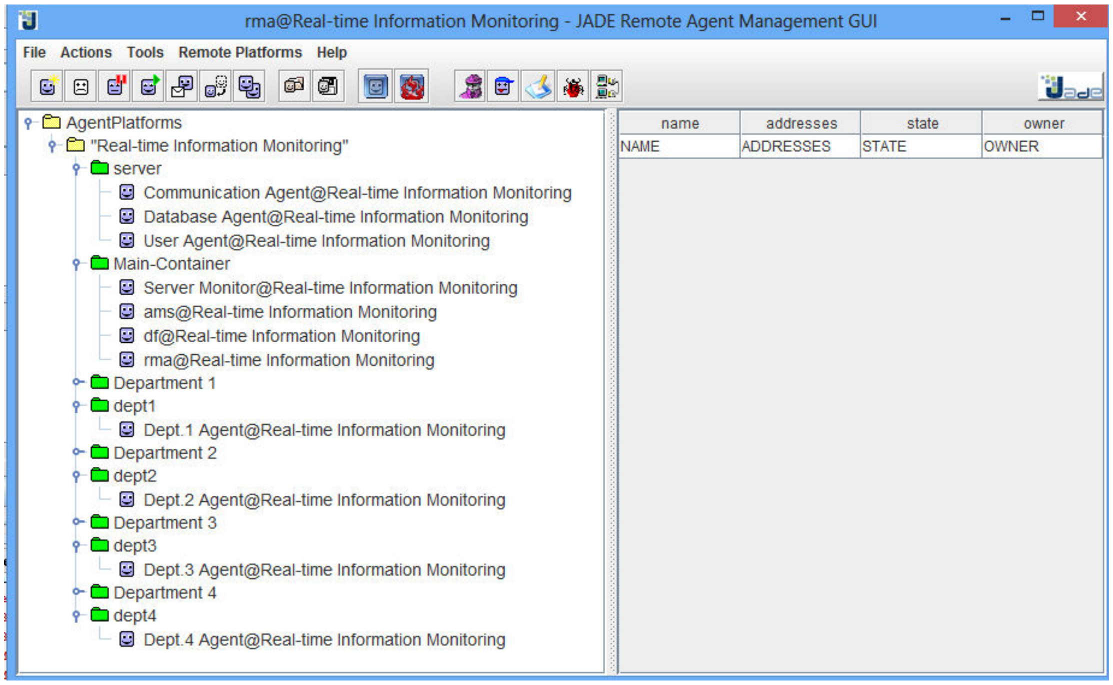
5.1 Overview

The main purpose of the system is providing an information monitoring framework for collaboration, sharing and exchange of information between security agencies.

This information exchange platform implements the multi agents solutions such as proactivenes, reactiveness, experimentation, interaction modeling, simplification of complex environment and thus facilitates speedy decision making process.

The main agents implemented includes Authentication agent which validated the authentic users as in the user table, Information Agent that receives data and validates it into the database, Database Agent that ensures that data is updated and saved into the common repository, Communication Agent used for communication purpose and send request to email agent, Email Agent for sending email and search Agent for conducting search over the common repository.

When the model is executed the Agent Gui runs the graphical interface. The main container automatically


rma@Real-time Information Monitoring - JADE Remote Agent Management GUI
File
Actions
Tools
Remote Platforms
Help

```
AgentPlatforms
  "Real-time Information Monitoring"
    server
      Communication Agent@Real-time Information Monitoring
      Database Agent@Real-time Information Monitoring
      User Agent@Real-time Information Monitoring
    Main-Container
      Server Monitor@Real-time Information Monitoring
      ams@Real-time Information Monitoring
      df@Real-time Information Monitoring
      rma@Real-time Information Monitoring
    Department 1
    dept1
      Dept.1 Agent@Real-time Information Monitoring
    Department 2
    dept2
      Dept.2 Agent@Real-time Information Monitoring
    Department 3
    dept3
      Dept.3 Agent@Real-time Information Monitoring
    Department 4
    dept4
      Dept.4 Agent@Real-time Information Monitoring
```

name	addresses	state	owner
NAME	ADDRESSES	STATE	OWNER

Figure 20: Agents container

5.2 Agents Interactions

The figure below demonstrates the interaction of all the agents involved in this system. It begins with users authentication, upload of data, sending of information to various agents and sending of email alert to all other users of the system.

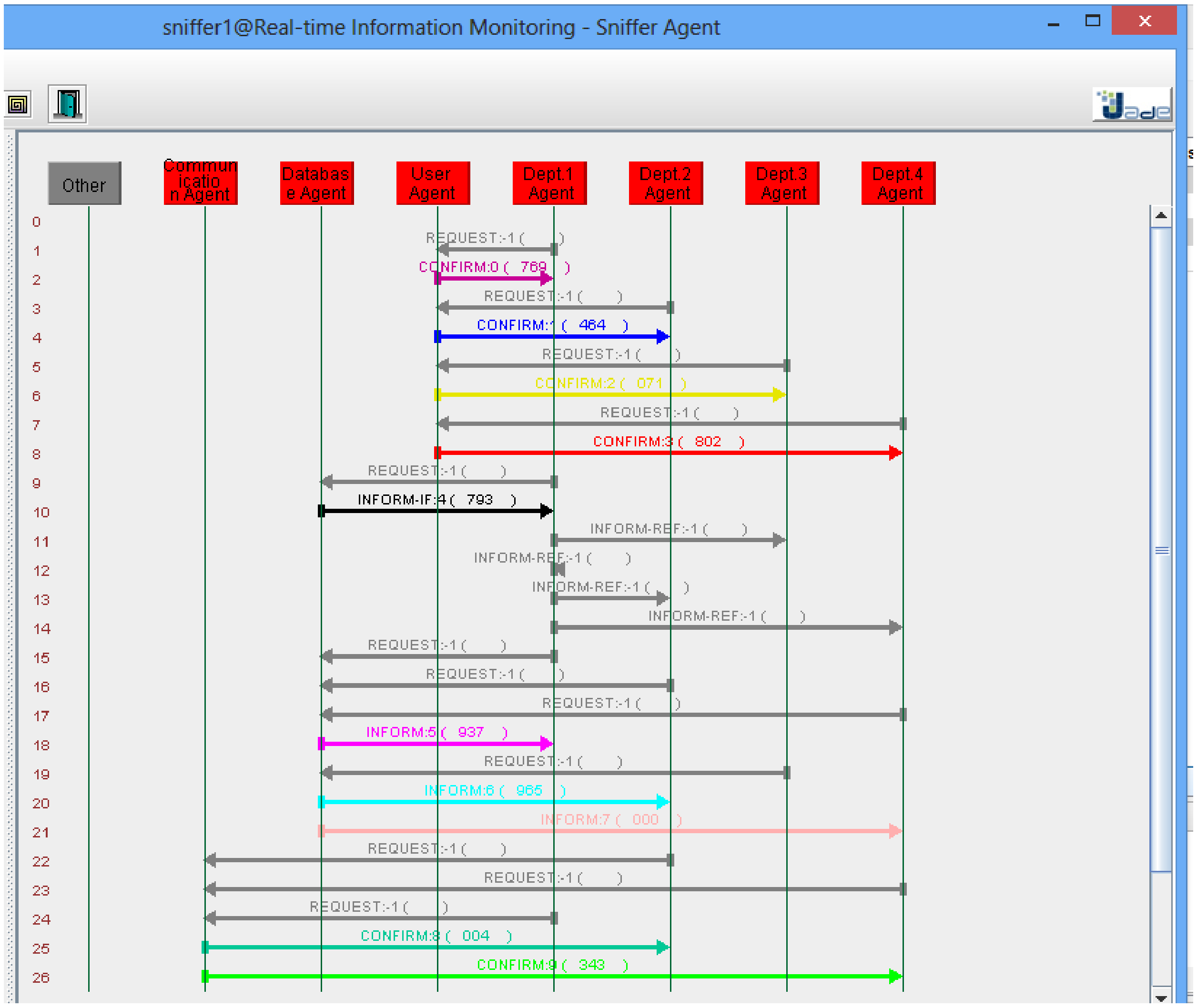


Figure 21: Agents Simulation

5.3 Implementation Tools

- Java (JDK (Java Development Kit) / JRE (Java Runtime Environment), Java

- Development Kit includes the Net Beans IDE, which is a powerful integrated development environment for developing applications on the Java platform.

- Jade Framework

- WSIG add-on for Jade framework

- Databases – mySQL, JavaDB

- Php, Python for External system Agents

- Application software for documentation

- Laptop

5.4 Tests for the developed system

TEST NUMBER	**TEST NAME**	**EXPECTED OUTCOME**	**ACTUAL OUTCOME**	**STATUS**
T-001	Create New Users	Successful creation of New users by the Administrator	All New users were created by the Administrator and displayed	Functional
T-002	User Authentication	For all Authentic Users the Log in session is successful	All Authentic Users successfully logged into the system while Unauthentic Users login failed	Functional

T-003	Report Upload	Report should be parsed based on the predefined departments	All the Reports in the correct format were successfully parsed into the various department	Functional
T-004	Report Submission	Submission of information to different department based on particular specific needs	All department received information that is specific to their needs	Functional
T-005	Send Email	For all information submitted an Email alert is sent to the correct users	Users received an Email alert for all information submitted specific to their department	Functional
T-006	Save Data	For all information submitted the data is saved in a common repository	All information submitted is committed into the common repository	Functional
T-007	Search Information	The system returns the information from common repository	All information searched from the common repository using predefined criteria were displayed	

5.5 Results

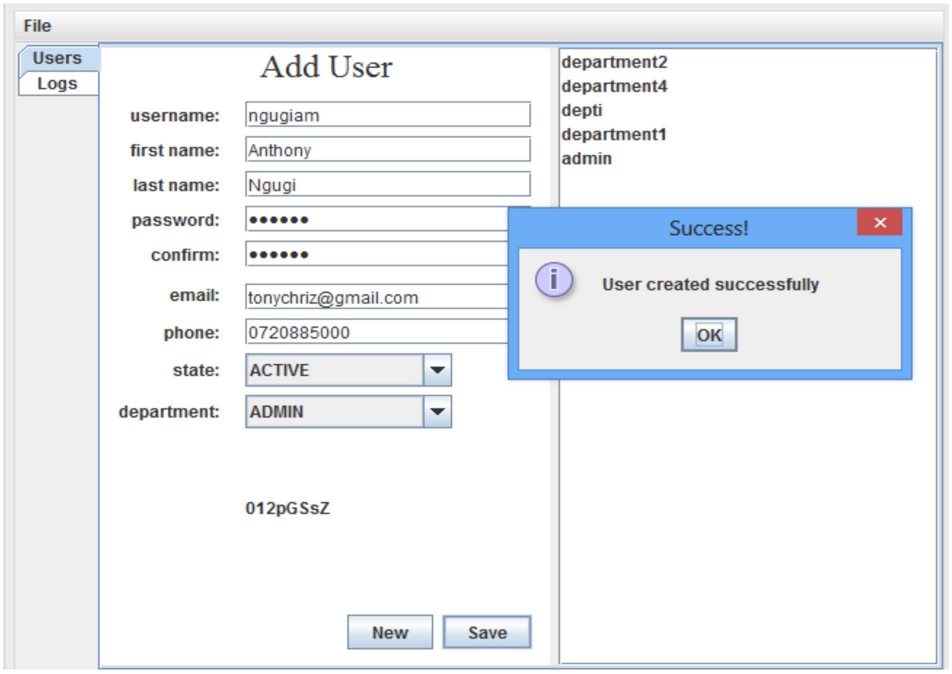
The user interface designed using Java Development Environment – Net beans has been provided for registration of new users. When a user logs into the system, the first step is to authenticate the user in relation to the users in the database. Authentic users are allowed into the system.

The home interface is where the information report is captured by first uploading it using acceptable report format. The report is parsed by the system algorithm partitioning it into four categories as defined by the relevant departments.

When the report is submitted, the following things takes place: The partitioned information is committed into the common repository, the information is sent to all the users interface for the users to see and finally an email alert is sent to all registered users.

The figure below shows the contents of the test obtained from the system and the results obtained

Fig:22 Successful creation of new users and the display of all authentic users of the system



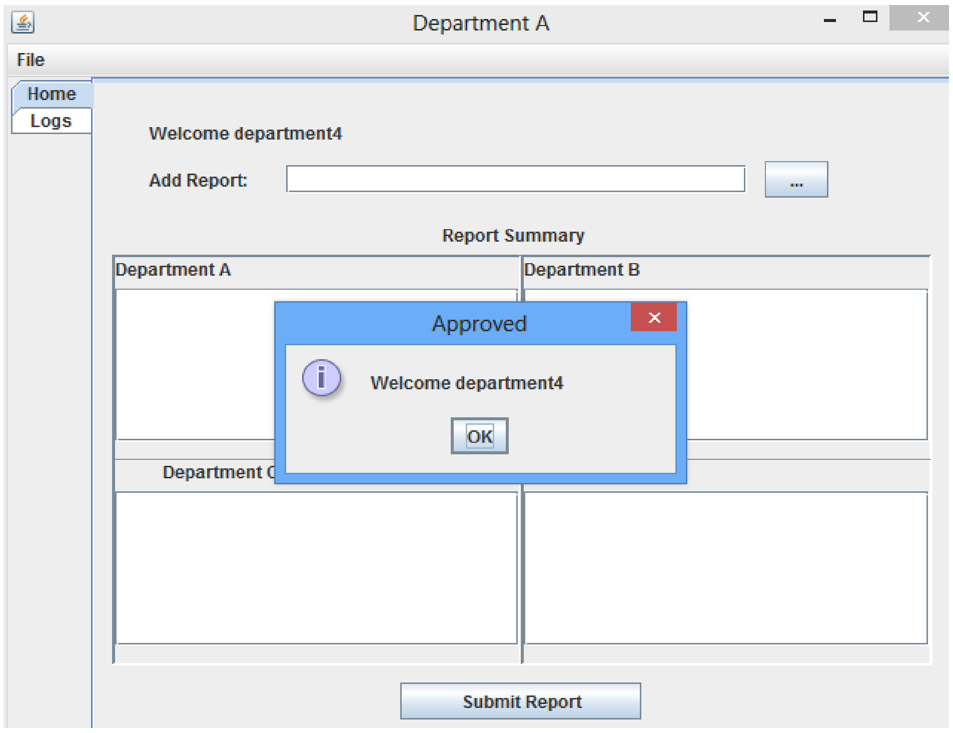
username: ngugiam
first name: Anthony
last name: Ngugi
password: ●●●●●●
confirm: ●●●●●●
email: tonychriz@gmail.com
phone: 0720885000
state: ACTIVE
department: ADMIN

012pGSsZ

department2
department4
depti
department1
admin

User created successfully

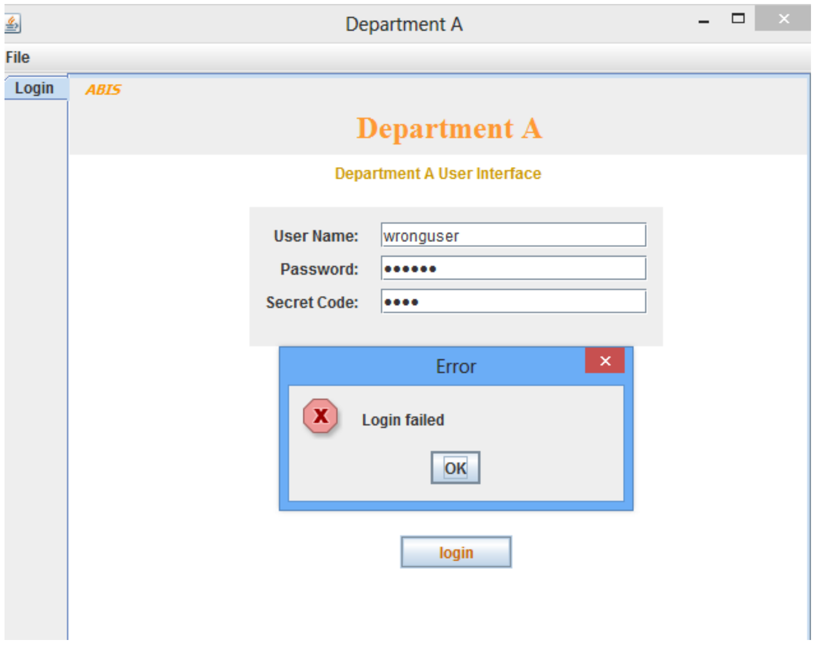
Figure 23: Shows the Log in session that was successful



Welcome department4

Welcome department4

Fig 24. Shows a log in session that had wrong credentials



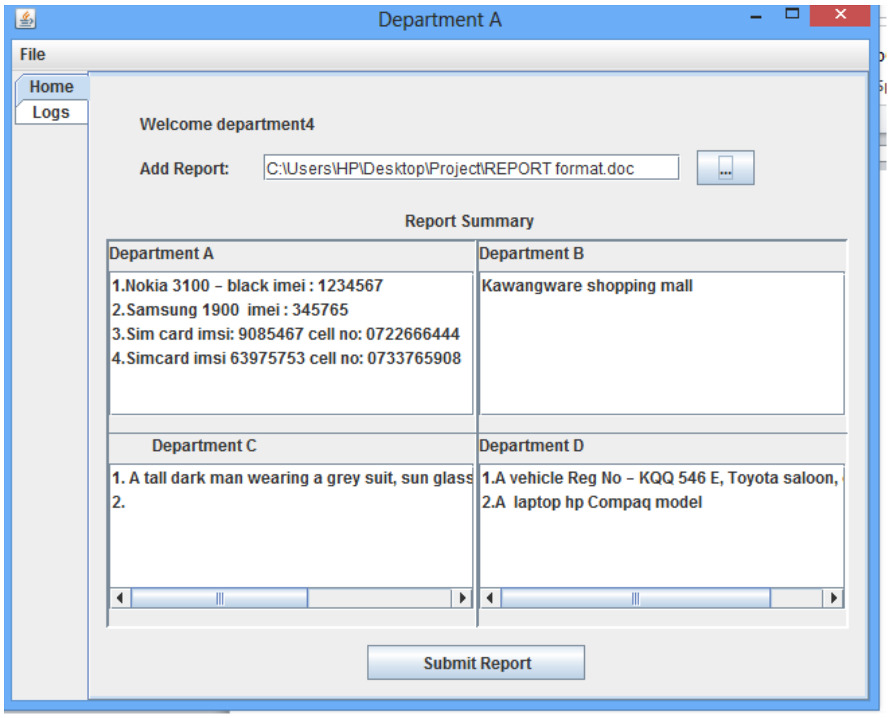
User Name: wronguser

Password: ••••••

Secret Code: ••••

Login failed

Fig 25. Shows the successful uploading of report presented in the standardized format



Welcome department4

Add Report: C:\Users\HP\Desktop\Project\REPORT format.doc

Report Summary

Department A
1.Nokia 3100 – black imei : 1234567
2.Samsung 1900 imei : 345765
3.Sim card imsi: 9085467 cell no: 0722666444
4.Simcard imsi 63975753 cell no: 0733765908

Department B
Kawangware shopping mall

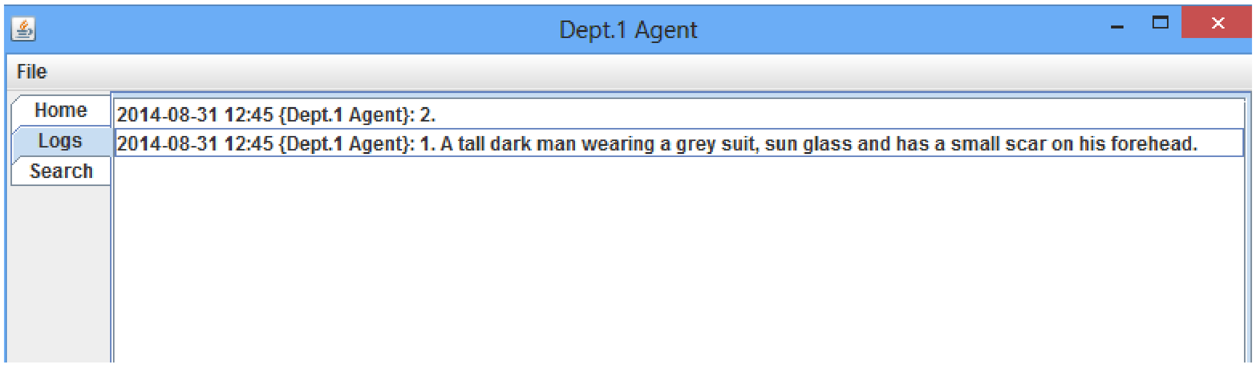
Department C
1. A tall dark man wearing a grey suit, sun glass
2.

Department D
1.A vehicle Reg No – KQQ 546 E, Toyota saloon,
2.A laptop hp Compaq model

Fig 25. Information Upload

The following figures demonstrate information sent to various departments from Department 1. Each department receives information relevant to them on a need to know bases.

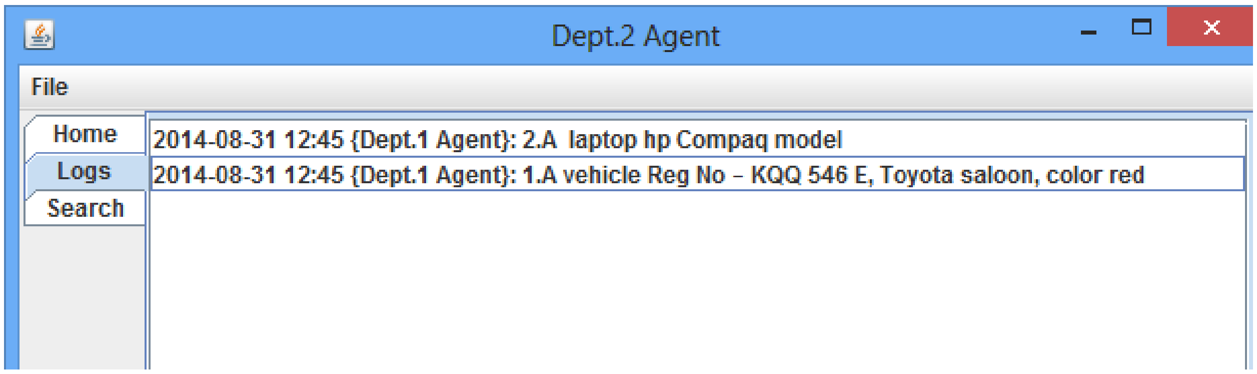
The figure below how shows a message sent to Department1 describing person involved



```
2014-08-31 12:45 {Dept.1 Agent}: 2.
2014-08-31 12:45 {Dept.1 Agent}: 1. A tall dark man wearing a grey suit, sun glass and has a small scar on his forehead.
```

Fig 26. Department 1 Message

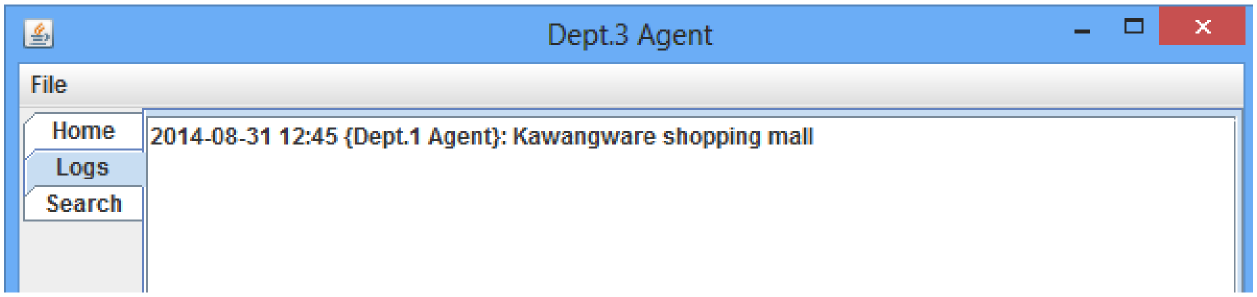
The figure below shows a message sent to Department 2 describing equipment involved



```
2014-08-31 12:45 {Dept.1 Agent}: 2.A  laptop hp Compaq model
2014-08-31 12:45 {Dept.1 Agent}: 1.A vehicle Reg No – KQQ 546 E, Toyota saloon, color red
```

Fig 27. Department 2 message

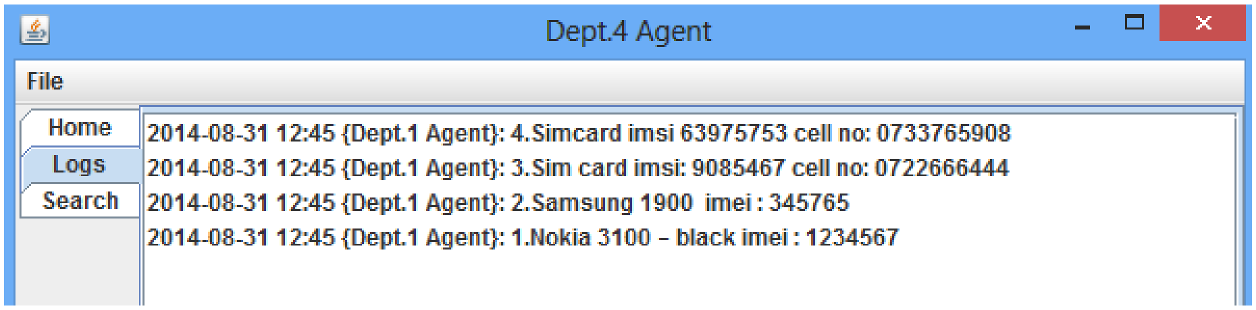
The figure below shows a message sent to Department 3 about location of incidence.



```
2014-08-31 12:45 {Dept.1 Agent}: Kawangware shopping mall
```

Fig 28. Department 3 message

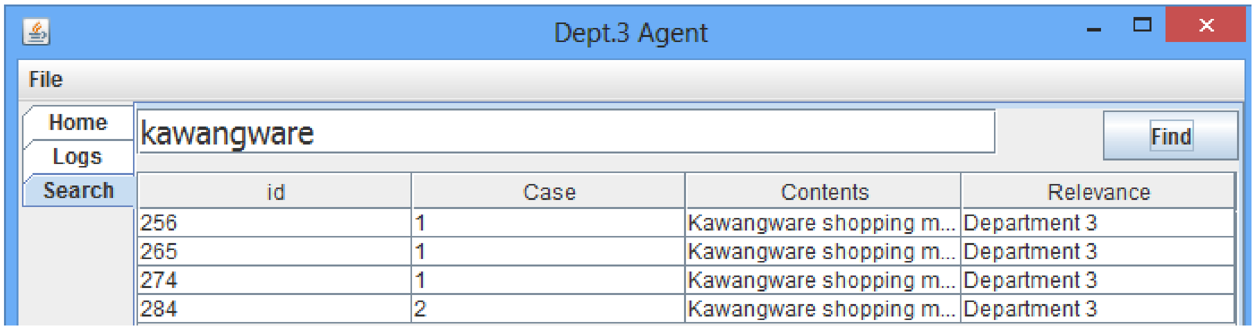
The figure below shows a message sent to Department 4 describing the communication gadgets.



```
2014-08-31 12:45 {Dept.1 Agent}: 4.Simcard imsi 63975753 cell no: 0733765908
2014-08-31 12:45 {Dept.1 Agent}: 3.Sim card imsi: 9085467 cell no: 0722666444
2014-08-31 12:45 {Dept.1 Agent}: 2.Samsung 1900  imei : 345765
2014-08-31 12:45 {Dept.1 Agent}: 1.Nokia 3100 – black imei : 1234567
```

Fig 29. Department 4 message

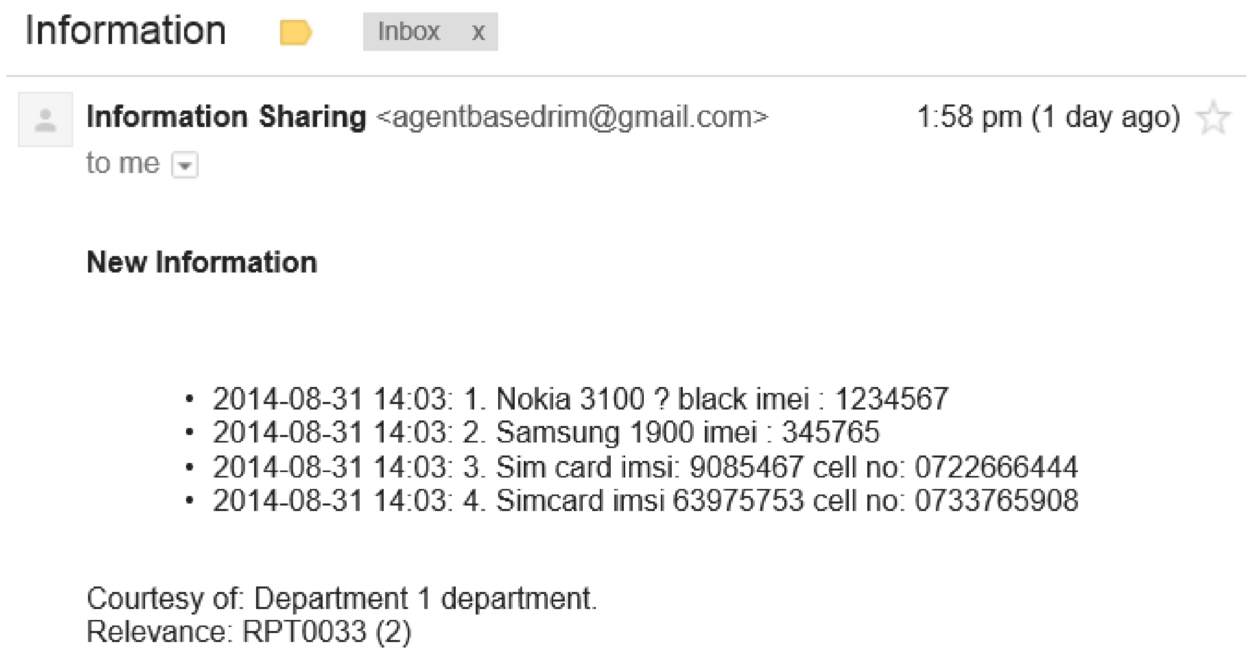
The figure below demonstrates how the system searches all the information from the common repository containing any particular string entered



id	Case	Contents	Relevance
256	1	Kawangware shopping m...	Department 3
265	1	Kawangware shopping m...	Department 3
274	1	Kawangware shopping m...	Department 3
284	2	Kawangware shopping m...	Department 3

Fig 30. Agents search

The figure below shows an email alert sent to department 1 describing the communication equipment used in crime


Information Inbox x

Information Sharing <agentbasedrim@gmail.com> 1:58 pm (1 day ago)
to me

New Information

- 2014-08-31 14:03: 1. Nokia 3100 ? black imei : 1234567
- 2014-08-31 14:03: 2. Samsung 1900 imei : 345765
- 2014-08-31 14:03: 3. Sim card imsi: 9085467 cell no: 0722666444
- 2014-08-31 14:03: 4. Simcard imsi 63975753 cell no: 0733765908

Courtesy of: Department 1 department.
Relevance: RPT0033 (2)

Fig 31. Email Alert

5.6 Evaluation

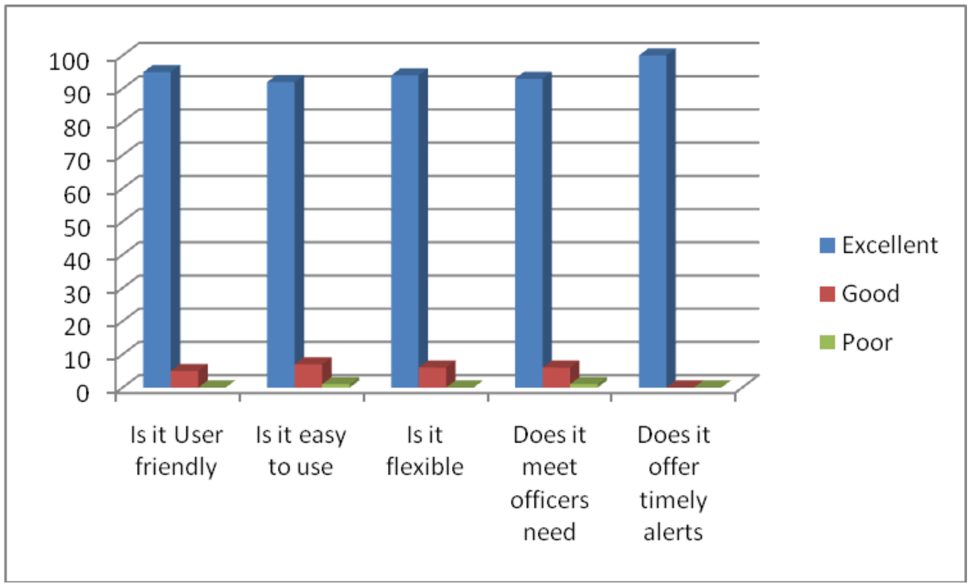
Evaluation of the multi agent system for monitoring security intelligence system was done by drawing respondents from a security department in a State Corporation. This was as a result of the high challenges faced in an attempt to perform evaluation in the main stream security organs due to sensitivity and secrecy of real data. Nevertheless the result achieved were a good indication and can be applied to a wider scope. Evaluation involved collection of data and analysis. It was done to ensure user requirement and system specifications were achieved. The evaluation was as stipulated in the table below.

S/No	Functionality	Excellent	Good	Poor	Mean
		3	2	1	
1	Is it User friendly	95	5	0	**2.95**
2	Is it easy to use	92	7	1	**2.91**

3	Is it flexible	94	6	0	**2.92**
4	Does it meet officers need	93	6	1	**2.92**
5	Does it offer timely alerts	100	0	0	**3**
TOTAL MEAN		94.8	4.8	0.4	**2.94**

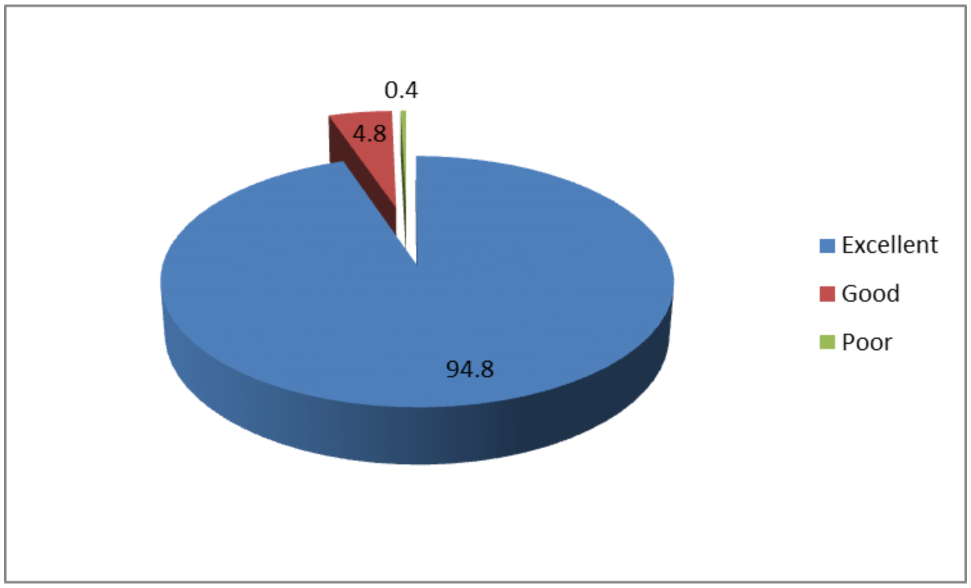
The table above reveals that the evaluation of the overall prototype based on the five main areas namely:- user friendliness, ease of use, flexibility of use, meeting user requirements and timely alert. Based on the evaluation results the overall prototype had a very high performance of a mean of 2.94 which means excellence performance rating.

The graph below reveals the performance of each item of evaluation in percentage. They ranged from an average from 2.91 - 3.0 meaning that all item had an excellent performance.



The pie chart below indicates that 94.8% registered that the overall prototype is excellent while only 4.8% registered prototype as good and only 0.4% described it as poor. The five questions performed as below:- 98.3% described it as user friendly, 97% described it as easy to use, 97.3%

described it as flexible, 97.3% registered that it met their needs while all of the respondent registered their satisfaction with regard to email alerts.



5.7 Discussion of the results

As described above, the multi agent system has demonstrated agent's cooperation in the sense that all the information received by one agent is exchanged with other agents on the basis of what is relevant to them. The system is designed in such a way that each agent need each other in order to accomplish its task.

Agent coordination is also mainfested by using communication and database agents as the center through which all other agents pass through to reach each other.

Each department agent has a unique task to accomplish. It can be deployed in its unique platform independently. This demonstrates autonomy of agents.

The MAS system proved that the interaction of agents can be modeled and thus build system by aid of experimentation.

From the test conducted by random sample of users, we established that security information monitoring can be done effectively by the use of multi agent system.

CHAPTER SIX: CONCLUSSION AND FUTURE WORK

6.1 Chapter overview

This chapter presents projects achievements, project challenges, project limitations, recommendations and further work

6.2 Project achievements

This study has proved that multi agent system is a solution to facilitate quick information processing and a platform for data exchange to enable faster decision making. This is further necessitated by the ability of multi agent system to work with virtually all data formats and multi platforms. This system can be applied by any authorized security agency for processing critical intelligence and thereby detecting and preventing crimes before they happen.

6.3 Project challenges and limitations

Collection of data was hampered by the nature of confidentiality and sensitivity of security information. It was further compounded by respondent who were not willing to discuss on any matter pertaining security.

Evaluation and testing of the Multi Agent System was also a big issue since we had to rely on simulated and anonymous data. This data was validated using a business organization setting that was willing to cooperate.

The tools that were used to design and develop the multi agent system were difficult to learn within a very strict time line. The selection of the most appropriate tool, technology and methodology was also a big problem.

6.4 Recommendations and future work

Based on the results achieved, it is evident that multi agent system is a solution to any security intelligence monitoring and exchange.

This study was only limited to security organizations. Further study should extend to business organizations that rely on intelligence for competitiveness and survival to employ multi agents.

Since the system stores its data in a common repository, it is recommended that in future better decisions can be achieved by applying data mining tools and techniques to pull information from this rich repository.

APPENDIX I: References

1. ARONSON, SL. (2013) Kenya and Global War on Terror. *Africa Journal of Criminology and Justice Studies.* Vol 7
2. BLOCK, L.(2005), European Counter Terrorism Culture and Methodology, *Terrorism Monitor*, Volume 3, issue 8, 21-Apr-05.
3. BRENNER, ET AL. (1998). Intelligent Sofwatre Agents. Foundations and Applications. Heidelberg, Springer.
4. BROWN, M. (2001). System, methods and computer program products for automatically monitoring police records for crime profile
5. BRÜCK, T . KARAISL, M. SCHNEIDER, F. (2008) A Survey of the Economics of Security. *Economics of Security Working Paper Series.*
6. FREY, BS. LUECHINGER, S. STUTZER, D. (2004) Calculating Tragedy: Assessing the Costs of Terrorism. *Institute for Empirical Research in Economics.* University of Zürich. Working Paper Series ISSN: 1424-0459.
7. GIORGINI, P. (2001) Agent oriented software engineering. *Journal of Autonomous Agents and Multi-Agent Systems.* Kluwer Academic Publishers, Volume 6, Number 2, pp.115-143, March 2003
8. GIORGINI, P. HENDERSON-SELLERS, B.(2005) Agent-Oriented Methodologies*, Idea Group Inc*
9. GIOVANNI, C ET AL.(2006) *Developing multi-agents systems with JADE*. Wiley series in Agent technology.
10. JACKSON, BA. (2006) Information Sharing and Emergency Responder Safety Management, *testimony presented before the House Government Reform Committee on March 30th, 2006.* RAND Corporation Testimony Series, 30-Mar-06. p. 5.
11. KASSEL, A. (2001) *Internet Monitoring Clipping*. Marketing and competitive Intelligence
12. KEITH, SD. & KATIA,S. (1997) Intelligence Adaptive Information Agents. *Journal of intelligence information system.* Vol 9
13. MAES, P. (1991). The agent network architecture (ANA). *SIGART Bulletin*, 2(4):115-120.

14. MAKKAI, T. (1999) Linking Drugs & criminal activity: Developing an integrated monitoring system. *Trends & issues in crime and criminal justice*. No 109. (April)
15. MCGARELL, EF. ET AL. (2007) Intelligence-Led Policing As a Framework for Responding to Terrorism. *Journal of Contemporary Criminal Justice*. Vol. 23, p. 149.
16. MULLER, ER. ET AL. (1999) Trends in Terrorism. *COT Institute for Crisis management*. p. 172.
17. ODELL, J (2005) The Genesis of a Pattern Language for Agent-based Enterprise Systems. QSIC 2005: 395-400
18. OVEREINDER, BJ. BRAZIER, FM. (2004). Scalable middleware environment for agent-based internet applications. p 675–679
19. SERBAN, A. LUAN, M. JING. (2002). Corporate Strategy Model. *Scenario Planning*, pg 5.
20. SRIVASTAVA, A. (2012) Transaction Processing In Replicated Data in the DDBMS. *International Journal of Modern Engineering Research (IJMER).*Vol2. Issue.4, July-Aug. 2012 pp-2409-2416
21. SYCARA, K. (1998) Multiagent System. *AI Journals*. Vol 19 (2)
22. TAKÁČ, L. ZÁBOVSKÝ, M. MATIAŠKO, K. (2011) MNSight – A new Automatic on-line Media Monitoring System
23. VRIES, G. (2005) The European Union's Role in the Fight against Terrorism. *Irish Studies in International Affairs*. Vol. 16, (2005), p. 3.
24. WINIKOFF, M. PADGHAM, L. (2005) Prometheus: A practical Agent-oriented Methodology
25. WOOLDRIDGE, JENNINGS, ET AL.(1999), A methodology for agent-oriented analysis and Design

APPENDIX II: EVALUATION FORM

OFFICERS NAME:				
DATE OF EVALUATION:				
OVERALL PERFOMANCE:				
S/NO	**QUESTION**			
1	It it user friendly?	3	2	1
2	It it easy to use?	3	2	1
3	Is it flexible?	3	2	1
4	Does it meet Officers needs?	3	2	1
5	Does it offer timely information?	3	2	1
(Maximum Total of 15)				
Ratings **3= Excellence 2= Good 1= Poor**				

APPENDIX III: AGENTS ALGORITHM

MAIN CLASS

```
package agent.base.info.sharing;

import agent.base.info.sharing.agents.SystemRunnerAgent;
import agent.base.info.sharing.database.DEPARTMENTS;
import agent.base.info.sharing.database.DatabaseManager;
import agent.base.info.sharing.database.EntityState;
import agent.base.info.sharing.database.classes.User;
import agent.base.info.sharing.resources.Methods;
import jade.core.Profile;
import jade.core.ProfileImpl;
import jade.wrapper.AgentContainer;
import java.util.logging.Level;
import javax.swing.JOptionPane;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class MainClass {

    private final static Logger logger = LoggerFactory.getLogger(MainClass.class);
    private static RunMode runMode;

    public static RunMode getRunMode() {
        return runMode;
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        logger.info(" main -- starting platform ");
        MainClass main = new MainClass();
        boolean start = main.start(args);
        if (start) {
            logger.info(" main -- platform started successfully ");
        } else {
            logger.error(" main -- platform startup failed ");
            JOptionPane.showMessageDialog(null, "Platform startup failed", "Fatal error!",
JOptionPane.ERROR_MESSAGE);
            System.exit(0);
        }
    }

    private boolean start(String[] args) {
        runMode = new RunMode(args);
        boolean started = false;
        createDefaultSystemUsers();
        try {
            Profile p;
            if (runMode.isMain()) {
                p = new ProfileImpl("localhost", 1099, runMode.getPlatformName(), true);
                p.setParameter(jade.core.Profile.MAIN_HOST, "localhost");
                p.setParameter(jade.core.Profile.MAIN_PORT, "1099");
                p.setParameter(jade.core.Profile.LOCAL_HOST, "localhost");
                p.setParameter(jade.core.Profile.LOCAL_PORT, "1099");
                logger.info(" start -- starting main container ");
            } else {
                p = new ProfileImpl("localhost", 1099, runMode.getPlatformName(), false);
                p.setParameter(Profile.MAIN_HOST, "localhost");
                p.setParameter(Profile.MAIN_PORT, "1099");
                p.setParameter(Profile.CONTAINER_NAME, runMode.getContainerName());
//                p.setParameter(Profile.LOCAL_PORT, "10099");
                logger.info(" start -- starting secondary container ");
            }
//            p.setParameter(Profile.PLATFORM_ID, runMode.getPlatformName());
            p.setParameter(Profile.MAIN, runMode.isMain() ? "true" : "false");
            p.setParameter("gui", "true");
            AgentContainer container;
            if (runMode.isMain()) {
                container = jade.core.Runtime.instance().createMainContainer(p);
            } else {
                container = jade.core.Runtime.instance().createAgentContainer(p);
            }
            container.start();
            container.createNewAgent(runMode.getMonitorAgentName(),
SystemRunnerAgent.class.getName(),
                    null).start();
            started = true;
        } catch (Exception ex) {
            logger.error(" start -- error starting container ", ex);
        }
        return started;
    }

    private void createDefaultSystemUsers() {
        DatabaseManager dm = new DatabaseManager();
        User u = new User();
        u.setDepartment(DEPARTMENTS.ADMIN);
        u.setEmail("admin@dummy.com");
        u.setEntityState(EntityState.ACTIVE);
        u.setFirstName("Admini");
        u.setLastName("Default");
        Methods methods = new Methods();
        try {
            u.setPassword(methods.getHash("password"));
        } catch (Exception ex) {
            java.util.logging.Logger.getLogger(MainClass.class.getName()).log(Level.SEVERE, null, ex);
        }
        u.setPhone("0700000000");
        u.setSecret("1234");
        u.setUsername("admin");
        try {
            dm.save(u);
        } catch (Exception e) {
        }
    }

    public class RunMode {

        private final String[] args;

        public RunMode(String[] args) {
            this.args = args;
        }

        public boolean isMain() {
            if (isEmptyArgs()) {
                return true;
            }
            return args[0].toLowerCase().contains("main");
        }

        public boolean isDEPT2() {
            if (isEmptyArgs()) {
                return false;
            }
            return args[0].toLowerCase().contains("DEPT2");
        }

        public boolean isDEPT1() {
            if (isEmptyArgs()) {
                return false;
            }
            return args[0].toLowerCase().contains("DEPT1");
        }

        public boolean isDEPT3() {
            if (isEmptyArgs()) {
                return false;
            }
            return args[0].toLowerCase().contains("DEPT3");
        }

        public boolean isDEPT4() {
            if (isEmptyArgs()) {
                return false;
            }
            return args[0].toLowerCase().contains("DEPT4");
        }
```

```
        private boolean isEmptyArgs() {
            return args == null || args.length == 0;
        }

        public String getPlatformName() {
            return "Real-time Information Monitoring";
        }

        private String getMonitorAgentName() {
            if (isMain()) {
                return "Server Monitor";
            } else if (isDEPT1()) {
                return "Dept1 Monitor";
            } else if (isDEPT3()) {
                return "Dept3 Monitor";
            } else if (isDEPT4()) {
                return "Dept4 Monitor";
            } else if (isDEPT2()) {
                return "Dept2 Monitor";
            } else {
                return "Unknown";
            }
        }

        private String getContainerName() {
            if (isMain()) {
                return "Server";
            } else if (isDEPT1()) {
                return "Department 1";
            } else if (isDEPT3()) {
                return "Department 3";
            } else if (isDEPT4()) {
                return "Department 4";
            } else if (isDEPT2()) {
                return "Department 2";
            } else {
                return "Unknown";
            }
        }

    }
}
```

CUSTOM AGENT TEMPLATE

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package agent.base.info.sharing;
```

```
import agent.base.info.sharing.database.classes.Report;
import agent.base.info.sharing.database.classes.User;
import agent.base.info.sharing.gui.AgentGUI;
import agent.base.info.sharing.onto.EmailRequestWrapper;
import jade.core.AID;
import jade.core.behaviours.CyclicBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.SearchConstraints;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAException;
import jade.gui.GuiAgent;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import java.io.IOException;
import java.io.Serializable;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.slf4j.LoggerFactory;

public abstract class CustomAgentTemplate extends GuiAgent {

    protected final org.slf4j.Logger log = LoggerFactory.getLogger(getClass());

    @Override
    protected void takeDown() {
        log.debug(" takeDown -- closing agent");
        try {
            DFService.deregister(this);
        } catch (FIPAException ex) {
            log.error(" takeDown -- error ", ex);
        }
        try {
            super.takeDown(); //To change body of generated methods, choose Tools | Templates.
        } catch (Exception e) {
        }
    }

    @Override
    protected void setup() {
        super.setup(); //To change body of generated methods, choose Tools | Templates.
        log.debug(" setup -- initializing agent {} ", getLocalName());
        register();
        handleLoginRequests();
    }

    protected final void register() {
        log.debug(" register -- registering agent {} with DF ", getLocalName());
        DFAgentDescription dfd = new DFAgentDescription();
        dfd.setName(getAID());
        ServiceDescription sd = new ServiceDescription();
        sd.setType(getClass().getSimpleName());
        sd.setName(getLocalName());
        dfd.addServices(sd);
        try {
            DFService.register(this, dfd);
        } catch (FIPAException fe) {
            log.error(" register -- error ", fe);
        }
    }

    protected final AID getService(String service) {
        log.debug(" getServices -- searching DF for one {} ", service);
        DFAgentDescription dfd = new DFAgentDescription();
        ServiceDescription sd = new ServiceDescription();
        sd.setType(service);
        dfd.addServices(sd);
        try {
            DFAgentDescription[] result = DFService.search(this, dfd);
            if (result.length > 0) {
                return result[0].getName();
            }
        } catch (FIPAException fe) {
            log.error(" getService -- error ", fe);
        }
        return null;
    }

    protected final AID[] getServices(String service) {
        log.debug(" getServices -- searching DF for {} ", service);
        DFAgentDescription dfd = new DFAgentDescription();
        ServiceDescription sd = new ServiceDescription();
        sd.setType(service);
        dfd.addServices(sd);

        SearchConstraints ALL = new SearchConstraints();
        ALL.setMaxResults(new Long(-1));

        try {
            DFAgentDescription[] result = DFService.search(this, dfd, ALL);
            AID[] agents = new AID[result.length];
            for (int i = 0; i < result.length; i++) {
                agents[i] = result[i].getName();
            }
            return agents;

        } catch (FIPAException fe) {
            log.error(" searchDF -- error ", fe);
        }

        return null;
```

```
}

protected AgentGUI getGUI() {
   return null;
}

private void handleLoginRequests() {
   if (getGUI() == null) {
      return;
   }
   log.info(" handleLoginRequest -- {} can Login", getLocalName());
   class LoginResponseHandler extends CyclicBehaviour {

      private final MessageTemplate loginResults
            = MessageTemplate.MatchSender(new AID(USER_AGENT, AID.ISLOCALNAME));

      @Override
      public void action() {
         ACLMessage receive = receive(loginResults);
         if (receive != null) {
            try {
               Serializable contentObject = receive.getContentObject();
               if (contentObject == null) {
                  //login failed
                  getGUI().applyLoginResults((User) contentObject);
               } else {
                  getGUI().applyLoginResults((User) contentObject);
               }
            } catch (Exception ex) {
               log.error(" handleLoginRequests -- error ", ex);
            }
         }
         block();
      }

   };
   addBehaviour(new LoginResponseHandler());
}

public static final String USER_AGENT = "User Agent";
public static final String EMAIL_AGENT = "Communication Agent";
public static final String DATABASE_AGENT = "Database Agent";
public static final String DEPT1_AGENT = "Dept.1 Agent";
public static final String DEPT2_AGENT = "Dept.2 Agent";
public static final String DEPT3_AGENT = "Dept.3 Agent";
public static final String DEPT4_AGENT = "Dept.4 Agent";

public void handleReport(final String re, final String caseNo) {
   new Thread(new Runnable() {

      @Override
```

```
            public void run() {
                Report r = new Report();
                r.setFile(re);
                r.setUploader(getGUI().getActiveUser());
                r.setDepartment(getGUI().getActiveUser().getDepartment());
                r.setCaseSerial(caseNo);
                ACLMessage acl = new ACLMessage(ACLMessage.REQUEST);
                try {
                    acl.setContentObject(r);
                    acl.addReceiver(new AID(DATABASE_AGENT, AID.ISLOCALNAME));
                    send(acl);
                } catch (IOException ex) {
                    Logger.getLogger(CustomAgentTemplate.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        }).start();
    }

    public void sendEmail(EmailRequestWrapper ew) {
        ACLMessage a = new ACLMessage(ACLMessage.REQUEST);
        a.addReceiver(new AID(EMAIL_AGENT, AID.ISLOCALNAME));
        try {
            a.setContentObject(ew);
            send(a);
        } catch (IOException ex) {
            Logger.getLogger(CustomAgentTemplate.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

AUTHENTICATION AGENT

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package agent.base.info.sharing.agents;

import agent.base.info.sharing.CustomAgentTemplate;
import agent.base.info.sharing.database.DatabaseManager;
import agent.base.info.sharing.database.EntityTemplate;
import agent.base.info.sharing.database.classes.User;
import agent.base.info.sharing.onto.LoginRequestWrapper;
import jade.core.behaviours.CyclicBehaviour;
import jade.gui.GuiEvent;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.lang.acl.UnreadableException;
import java.io.IOException;
import java.io.Serializable;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author Ngugi 2014
 */
public class AuthenticationAgent extends CustomAgentTemplate {

    private DatabaseManager dManager;

    public AuthenticationAgent() {
        dManager = new DatabaseManager();
    }

    @Override
    protected void setup() {
        super.setup(); //To change body of generated methods, choose Tools | Templates.
        addBehaviour(new AuthenticateUsers());
    }

    private class AuthenticateUsers extends CyclicBehaviour {

        MessageTemplate m;

        private AuthenticateUsers() {
            this.m = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
        }

        @Override
        public void action() {
            ACLMessage r = receive(m);
            if (r != null) {
                try {
                    Serializable contentObject = r.getContentObject();
                    LoginRequestWrapper lw = (LoginRequestWrapper) contentObject;
                    String q = "SELECT u FROM User u WHERE u.username = :username AND u.password =
:password AND u.secret = :code ORDER BY u.id ASC";
                    Map<String, Object> map = new HashMap<>();
                    map.put("username", lw.getUsername());
                    map.put("code", lw.getCode());
                    map.put("password", lw.getPassword());
                    List<User> findEntities = dManager.findEntities(q, map, 0, 1);
                    ACLMessage createReply = r.createReply();
                    if (findEntities.isEmpty()) {
                        createReply.setContentObject(null);
                        createReply.setPerformative(ACLMessage.REFUSE);
                    } else {
```

```
                    User get = findEntities.get(0);
                    createReply.setContentObject(get);
                    createReply.setPerformative(ACLMessage.CONFIRM);
                }
                send(createReply);
            } catch (UnreadableException ex) {
                Logger.getLogger(AuthenticationAgent.class.getName()).log(Level.SEVERE, null, ex);
            } catch (IOException ex) {
                Logger.getLogger(AuthenticationAgent.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
        block();
    }
}

@Override
protected void onGuiEvent(GuiEvent ge) {
}

}
```

COMMUNICATION AGENT

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package agent.base.info.sharing.agents;

import agent.base.info.sharing.CustomAgentTemplate;
import agent.base.info.sharing.database.DEPARTMENTS;
import agent.base.info.sharing.database.DatabaseManager;
import agent.base.info.sharing.database.EntityTemplate;
import agent.base.info.sharing.database.classes.Report;
import agent.base.info.sharing.database.classes.ReportContents;
import agent.base.info.sharing.onto.DatabaseActionWrapper;
import agent.base.info.sharing.onto.EmailRequestWrapper;
import agent.base.info.sharing.resources.GMAIL;
import agent.base.info.sharing.resources.ReportParser;
import jade.core.behaviours.CyclicBehaviour;
import jade.gui.GuiEvent;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.lang.acl.UnreadableException;
import java.io.IOException;
import java.io.Serializable;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
```

```
 *
 * @author Ngugi 2014
 */
public class CommunicationAgent extends CustomAgentTemplate {

    public CommunicationAgent() {
    }

    @Override
    protected void setup() {
        super.setup(); //To change body of generated methods, choose Tools | Templates.
        addBehaviour(new HandleEmailRequests());
    }

    private class HandleEmailRequests extends CyclicBehaviour {

        MessageTemplate m;

        private HandleEmailRequests() {
            this.m = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
        }

        @Override
        public void action() {
            final ACLMessage receive = receive(m);
            if (receive != null) {
                new Thread(new Runnable() {

                    @Override
                    public void run() {
                        try {
                            EmailRequestWrapper ew = (EmailRequestWrapper) receive.getContentObject();
                            GMAIL.getInstance().sendEmail(ew.getTo(), ew.getTitle(), ew.getHtml());
                            ACLMessage createReply = receive.createReply();
                            createReply.setContent("OK");
                            createReply.setPerformative(ACLMessage.CONFIRM);
                            send(createReply);
                        } catch (UnreadableException ex) {
                            Logger.getLogger(CommunicationAgent.class.getName()).log(Level.SEVERE, null,
ex);
                        } catch (Exception ex) {
                            Logger.getLogger(CommunicationAgent.class.getName()).log(Level.SEVERE, null,
ex);
                        }
                    }
                }).start();
            }
            block();
        }
    }
    @Override
    protected void onGuiEvent(GuiEvent ge) {
    }

}
```

DATABASE AGENT

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package agent.base.info.sharing.agents;

import agent.base.info.sharing.CustomAgentTemplate;
import agent.base.info.sharing.database.DEPARTMENTS;
import agent.base.info.sharing.database.DatabaseManager;
import agent.base.info.sharing.database.classes.Report;
import agent.base.info.sharing.database.classes.ReportContents;
import agent.base.info.sharing.database.classes.User;
import agent.base.info.sharing.onto.DatabaseActionWrapper;
import agent.base.info.sharing.resources.ReportParser;
import jade.core.behaviours.CyclicBehaviour;
import jade.gui.GuiEvent;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.lang.acl.UnreadableException;
import java.io.IOException;
import java.io.Serializable;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author Ngugi 2014
 */
public class DatabaseAgent extends CustomAgentTemplate {

    private DatabaseManager dManager;

    public DatabaseAgent() {
        dManager = new DatabaseManager();
    }

    @Override
    protected void setup() {
        super.setup(); //To change body of generated methods, choose Tools | Templates.
        addBehaviour(new HandleDatabaseRequests());
    }

    private class HandleDatabaseRequests extends CyclicBehaviour {

        MessageTemplate m;

        private HandleDatabaseRequests() {
            this.m = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
        }

        @Override
        public void action() {
            ACLMessage r = receive(m);
            if (r != null) {
                try {
                    Serializable contentObject = r.getContentObject();
                    if (contentObject instanceof Report) {
                        Report report = (Report) contentObject;
                        dManager.save(report);
                        ReportParser rp = new ReportParser(report.getFile());
                        DatabaseActionWrapper dw = new
DatabaseActionWrapper(DatabaseActionWrapper.CREATE);
                        for (String string : rp.getHumans()) {
                            ReportContents rc = new ReportContents();
                            rc.setContents(string);
                            rc.setRelevance(DEPARTMENTS.DEPT1);
                            rc.setReport(report);
                            dManager.save(rc);
                            dw.addEntity(rc);
                        }
                        for (String string : rp.getSignals()) {
                            ReportContents rc = new ReportContents();
                            rc.setContents(string);
                            rc.setRelevance(DEPARTMENTS.DEPT4);
                            rc.setReport(report);
                            dManager.save(rc);
                            dw.addEntity(rc);
                        }
                        for (String string : rp.getLocations()) {
                            ReportContents rc = new ReportContents();
                            rc.setContents(string);
                            rc.setRelevance(DEPARTMENTS.DEPT3);
                            rc.setReport(report);
                            dManager.save(rc);
                            dw.addEntity(rc);
                        }
                        for (String string : rp.getEquipment()) {
                            ReportContents rc = new ReportContents();
                            rc.setContents(string);
                            rc.setRelevance(DEPARTMENTS.DEPT2);
                            rc.setReport(report);
                            dManager.save(rc);
                            dw.addEntity(rc);
                        }
                        ACLMessage createReply = r.createReply();
                        createReply.setContentObject(dw);
                        createReply.setPerformative(ACLMessage.INFORM_IF);
                        send(createReply);
                    } else if (contentObject instanceof DatabaseActionWrapper) {
                        DatabaseActionWrapper lw = (DatabaseActionWrapper) contentObject;
                        if (lw.getAction() == DatabaseActionWrapper.REQUEST_USERS) {
                            String q = "SELECT u FROM User u WHERE u.department = :dept";
                            Map<String, Object> map = new HashMap<>();
                            map.put("dept", lw.getDepartment());
                            List<User> findEntities = dManager.findEntities(q, map);
                            for (User user : findEntities) {
                                lw.addUser(user);
                            }
                        }
                        ACLMessage createReply = r.createReply();
                        createReply.setContentObject(lw);
                        createReply.setPerformative(ACLMessage.INFORM);
                        send(createReply);
                    }
                } catch (UnreadableException ex) {
                    Logger.getLogger(DatabaseAgent.class.getName()).log(Level.SEVERE, null, ex);
                } catch (IOException ex) {
                    Logger.getLogger(DatabaseAgent.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
            block();
        }
    }

    @Override
    protected void onGuiEvent(GuiEvent ge) {
    }

    }
```

REPORT

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package agent.base.info.sharing.database.classes;

import agent.base.info.sharing.database.DEPARTMENTS;
import agent.base.info.sharing.database.EntityTemplate;
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.EnumType;
import javax.persistence.Enumerated;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Lob;
import javax.persistence.ManyToOne;

/**
 *
 * @author Ngugi 2014
 */
@Entity
public class Report extends EntityTemplate {

   private static final long serialVersionUID = 1L;
   @ManyToOne
   private User uploader;
   @Lob
   private String file;
   private String caseSerial;
   @Enumerated(EnumType.STRING)
   private DEPARTMENTS department;

   public String getCaseSerial() {
      return caseSerial;
   }

   public void setCaseSerial(String caseSerial) {
      this.caseSerial = caseSerial;
   }

   public User getUploader() {
      return uploader;
   }

   public void setUploader(User uploader) {
      this.uploader = uploader;
   }

   public String getFile() {
      return file;
   }

   public void setFile(String file) {
      this.file = file;
   }

   public DEPARTMENTS getDepartment() {
        return department;
    }

    public void setDepartment(DEPARTMENTS department) {
        this.department = department;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id fields are not set
        if (!(object instanceof Report)) {
            return false;
        }
        Report other = (Report) object;
        if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "agent.base.info.sharing.database.classes.Report[ id=" + id + " ]";
    }

}
```

EMAIL AGENT

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package agent.base.info.sharing.resources;

import java.util.Properties;
import javax.mail.Message;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
```

```
/**
 *
 * @author Ngugi 2014
 */
public class GMAIL {

    private static GMAIL instance;

    public static GMAIL getInstance() {
        if (instance == null) {
            instance = new GMAIL();
        }
        return instance;
    }
    private final String usernames;
    private final String username;
    private final String password;
    private Session session;

    private GMAIL() {
        usernames = "Information Sharing";
        username = "agentbasedrim@gmail.com";
        password = "abrim123";

        init();
    }

    public synchronized void sendEmail(String to, String title, String html) throws Exception {
        Message message = new MimeMessage(session);
        message.setRecipients(Message.RecipientType.TO,
                InternetAddress.parse(to));
        message.setSubject(title);
        message.setFrom(new InternetAddress(username, usernames));
        message.setContent(html, "text/html");

        Transport.send(message);
    }

    private void init() {

        Properties props = new Properties();
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.starttls.enable", "true");
        props.put("mail.smtp.host", "smtp.gmail.com");
        props.put("mail.smtp.port", "587");

        session = Session.getInstance(props,
                new javax.mail.Authenticator() {
                    @Override
                    protected PasswordAuthentication getPasswordAuthentication() {
                        return new PasswordAuthentication(username, password);
                }
            });
    }
}
```

REPORT PARSER

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package agent.base.info.sharing.resources;

/**
 *
 * @author Ngugi 2014
 */
public class ReportParser {

    private final String report;

    public ReportParser(String report) {
        this.report = report;
    }

    public String[] getSignals() {
        try {
            return report.split("COMMUNICATION GADGET:",
2)[1].split("DESCRIPTION")[0].trim().split(System.getProperty("line.separator"));
        } catch (Exception e) {
        }
        return null;
    }
    public String[] getHumans() {
        try {
            return report.split("PEOPLE INVOLVED:", 2)[1].split("EQUIPMENT
INVOLVED:")[0].trim().split(System.getProperty("line.separator"));
        } catch (Exception e) {
        }
        return null;
    }
    public String[] getLocations() {
        try {
            return report.split("LOCATION:", 2)[1].split("PEOPLE
INVOLVED:")[0].trim().split(System.getProperty("line.separator"));
        } catch (Exception e) {
        }
        return null;
    }
    public String[] getEquipment() {
        try {
            return report.split("EQUIPMENT INVOLVED:", 2)[1].split("COMMUNICATION
GADGET:")[0].trim().split(System.getProperty("line.separator"));
        } catch (Exception e) {
        }
        return null;
    }

}
```