



UNIVERSITY OF NAIROBI
SCHOOL OF COMPUTING AND INFORMATICS

AGENT BASED SYSTEM FOR
REAL TIME DATABASE AUDIT MONITORING

BY

BONIFACE AKUKU

P58/73079/2009

SUPERVISOR

MR. CHRISTOPHER MOTURI

August 2011

A research report submitted in partial fulfillment for the requirements of Master of
Science in Computer Science

University of NAIROBI Library



0439225 4

Table of Contents

Table of Contents	ii
Abstract	iii
Dedication	iv
Acknowledgement.....	v
Declaration	vi
Abbreviations	vii
List of Tables.....	viii
List of Figures	ix
CHAPTER 1-INTRODUCTION.....	1
1.1 Background	1
1.2 Problem Definition.....	2
1.3 Objectives.....	3
1.4 Research Questions	3
1.5 Proposed Solution	4
1.6 Agent Based System Algorithms	4
1.7 The scope of the study.....	4
1.8 Significance of the study	5
1.9 Conceptual Model of Agent Based System for Real-Time Database Audit Monitoring.....	6
CHAPTER 2 –LITERATURE REVIEW	9
2.1 Introduction	9
2.1.1 Inbuilt database audit log or audit trail system	9
2.1.2 Example Scenarios.....	10
2.1.3 Fingerprinting Scheme.....	11
2.1.4 Real-Time, Policy-Based Activity Monitoring.....	12
2.1.5 Agent Based Platform.....	12
2.1.6 Multi-Agent Concept and Approach.....	13
2.1.7 Reviewing and Evaluation of Available Database Auditing Tools and Solutions.....	13
CHAPTER 3 –RESEARCH METHODOLOGY	14
3.1 Data collection methods.....	14
3.1.1 Sources of data.....	14
3.1.2 Data collection tools	14
3.2 Data analysis method	15
3.3 Multi-Agent Methodology	15
3.4 Multi-Agent design	17
CHAPTER 4 –ANALYSIS AND DESIGN	18
4.1 Database Auditing Monitoring Requirements Analysis	18
4.2 System Specification.....	18

4.2.1	Functional Requirements	18
4.2.2	Scenarios	19
4.3	Architectural Design	25
4.3.1	Agents system overview	25
4.3.2	Agents Acquaintances using use case diagram.....	26
4.3.3	Agent messages communication.....	27
4.4	Agents Detailed Design.....	28
4.4.1	Agent Based System Overview Diagram	28
4.5	Agents Internal Process.....	28
4.5.1	Event Descriptors.....	28
4.5.2	Agents Plan Descriptors.....	29
4.6	Algorithms	29
4.7	Database Design	30
CHAPTER 5- SYSTEM IMPLEMENTATION AND RESULTS		31
5.1	Implementation of the System	31
5.2	System Testing.....	31
5.3	Discussion of Results	31
5.3.1	Challenges Facing Database Auditing.....	31
5.3.2	Evaluation of available database auditing tools and solutions.....	32
5.3.3	System Results.....	32
CHAPTER 6- CONCLUSION.....		37
6.1	Achievements.....	37
6.2	Research Contributions	38
6.3	Recommendation/ Future work.....	38
6.4	Assumptions and limitations	38

ABSTRACT

Database auditing is the examination of audit or transaction logs for the purpose of tracking changes with data or database structure. Existing database audit tools exerts performance overhead onto the database when logging audit activities. In proactive database auditing the agents track the database activities through database command execution and provide real time alert notification.

The research aims to develop a proactive database auditing system/prototype using multi-agent technology that provide real time audit reports and alerts to the auditors, independent of the database system, non-proprietary while overcoming performance overhead issues on the database system and enhances database audit logs availability. In this research we use Prometheus methodology which has been proven effective in the design and building of agent system, it is detailed and complete (start to end).

The Agent based database audit monitoring system utilizes agent properties and providing ability to monitor local or remote activities and events within the database with real time alerting and notification. The challenges facing existing database auditing are met by the agent system ability to enlist other agents including people to accomplish the alerting and notification tasks.

DEDICATION

I would like to dedicate this Master of Science Research Project to my wife Mrs. Christine Mwai Opiyo, my daughter Precious Haggie Okelo and my son David Omondi Okelo. There is no doubt in my mind that without their inspiration and counsel I could not have completed this process.

ACKNOWLEDGEMENT

We are extremely grateful to our supervisor Mr. Christopher Moturi for providing us an opportunity to get the in-depth knowledge of agent based system for real time database audit monitoring. We also acknowledge to Dr. Peter Waiganjo, Mr. Andrew Mwaura and Mr. Ogutu for their support and encouragement and given us an opportunity to work on this project.

We are thankful to Mr. Alfayo Adede for his intuitive ideas. Last but not the least we owe this achievement to the Almighty God for the strength and no to forget our family for their encouragement though being hundred miles away.

DECLARATION

The project presented in this report is my original work and has not been presented for any other university award.

Signature B Akuku

Date 18/08/2011

Boniface Akuku (P58/73079/2009)

This project has been submitted in partial fulfillment of the requirements of the Master of Science in Computer Science of the University of Nairobi with my approval as the University supervisor.

Signature CM

Date September 8, 2011

Mr. Christopher Moturi
Deputy Director
School of Computing and Informatics

Abbreviations

SQL- Standard Query Language

SYS.AUD\$ table - log table

Sys.fga_log\$ table --Fine grained log table

COBIT - Control Objectives for Information and related Technology

CD²FA - Content Addressed Delayed Input Deterministic Finite Automata

NTFS --NT files system

JCAP- JAVA packet capturing (JCAP) library

SYS -Systems

SMS --Short Message services

JADE - Java Agent Development Framework

I/O --Input output

GUI -- Graphical User Interface

E-book- Electronic book

DBMS- Database Management systems

OS- Operating systems

CPU- Central Processing Unit

DBA- Database Administrators

TCP/IP- Transport Control Protocol/ Internet protocol

List of Tables

Table 1: Agents and their functions

Table 2: Oracle Database Auditing Parameters that define specific conditions that must take place for the audit to occur

Table 3: Oracle Database 11.2.01 Standard Audit Trail with 50% CPU System Load

Table 4: Oracle Database 11.2.01 Fine Grained Audit Trail with 50% CPU System Load

Table 5: Summary analysis of existing database audit and monitoring tools and solutions

Table 6: Agents types, categories and responsibilities

Table 7: The Major Models of Prometheus

Table 8: Summary of existing database auditing tools and solutions

List of Figures

- Figure 1- Agent based system for real time database audit monitoring conceptual model
- Figure 2- Process flow diagram of agent based system for real time database audit monitoring
- Figure 3- Prometheus Methodology
- Figure 4- Database design
- Figure 5: Exceptional analysis agent receives notification and send the SMS and email to the auditor
- Figure 7: Formatted database audit report for all cases
- Figure 6: Formatted audit report on exceptional cases
- Figure 8: alert messages to the auditor
- Figure 9 and 10: User Interface of the system

CHAPTER 1

INTRODUCTION

1.1 Background

Database auditing is the examination of audit or transaction logs for the purpose of tracking changes with data or database structure. Databases can be set to capture alterations to data and metadata, along with modifications to the database system storing the data for auditing purpose. Most organizations business-critical data are stored in databases, therefore data confidentiality, availability or integrity of audit data is very important.

Database auditing involves monitoring and recording of selected user database actions and can be based on individual actions, such as the type of SQL statement executed, or on combinations of factors that can include user name, application, time, and so on. Most databases applications (DMBS) have inbuilt audit capabilities and stores auditing information (records/logs) when specified elements in databases are accessed or altered.

Currently in database auditing the recorded audit data is stored in either a data dictionary table, called the database audit trail, or in operating system files, called an operating system audit trail. Standard audit records can be written either to DBA_AUDIT_TRAIL (the sys.aud\$ table) or to the operating system. Fine-grained audit records are written to DBA_FGA_AUDIT_TRAIL (the sys.fga_log\$ table) and the DBA_COMMON_AUDIT_TRAIL view, which combines standard and fine-grained audit log records. The recording of audit data are inbuilt within the database system or operating system, leading to negative database performance issues such as response time, locking of log file system, database unavailability, crashing of file system especially NTFS, corruption of the log files and are non-real time lacking agents based capabilities, and logs extraction process overheads can be quite high.

According to Sushila (2008), best practices such as Control Objectives for Information and related Technology (COBIT) recognizes the need for database audit monitoring to use agent based solutions to overcome the challenges inherent with database auditing tools. Agent based system for database audit monitoring tracks the database activities through database command execution, provides a real time alert notification on exceptional cases at the execution of commands statements within the database. The emerging conclusion here is that agents role adds value to database auditing domain and database auditing needs to adopt multi-agent technology to overcome the present challenges it is faced with.

1.2 Problem Definition

Database management systems (DBMS) has become almost the main respiratory of organizations critical data, database auditing therefore needs to capture every single transaction taking place in the database. However the constant reading and writing of audit data results in substantial disk input/output operation slowing down the database performance (Herlands, 2007) and leads to slow database operation response time, locking of log file (sysfile), database unavailability due to crashing of file system especially NTFS, corruption of the log files on the database systems. Because of these performance issues database auditing and database performance are in constant conflict often resulting to database auditing being ignored.

It has also been argued that there are some limitations with database auditing because data access statements, commonly referred to as SELECT statements, are not collected attributed to the reasons above. Additionally, inbuilt database audits seldom capture the original query or variables passed by the user; rather they record a synthesized view of the event. The logs capture data values, both the before and after changes were made, according to Lane (2010) that makes audit trails much more useful for detecting what was changed, rather than what was accessed.

Several approaches and solutions have been developed to enhance database auditing however they are neither is real time nor proactive besides they do not address fully the database performance issues. These database auditing tools are susceptible to failure sometimes preventing auditing actions from completing in cases where the audit records cannot be stored due to database destination for audit data becoming full, therefore may be unable to accept new records, making recording auditing actions impossible.

This research project seek to develop an agent based system for real time database audit monitoring that both enhances and add value to database auditing domain by employing and utilizing multi-agent technology.

1.3 Objectives

The aim of this research project is develop a proactive real time agent based system/prototype that provides the auditor with overall visibility to the operations taking place within the database system and add value by enhancing database audit logging techniques to overcome the database performance problems mentioned above in the previous section. This will definitely contribute knowledge, value and technology option to database auditing domain in readiness for the future. The specific objectives of the agent based technology to make database auditing real time, proactive and platform independent are to:-

1. Develop database audit and monitoring agent based system using multi-agent technology that is proactive, real time, platform independent (non-proprietary), without performance overhead on the database and enhances audit logs availability.
2. Identify and apply a suitable algorithm in developing agent based database audit monitoring system via TCP/IP.
3. Develop a logical database structure for audit logs that meets the increasing demand for database security, compliance and regulatory requirements driving the database auditing.
4. Develop analysis agent that performs real-time notification.

1.4 Research Questions

Findings from other studies indicates that there is a relationship between the quality of service auditors provides for their organizations and the database auditing tools and solutions provided for the audit function, for this reason a number of third party solutions are in use today by auditors in trying address the problems and challenges with database auditing. Thus a better understanding of database auditing tools and solutions plays a critical role in the design and implementation of database audit monitoring solution. This research project attempt to answer several fundamental questions to database auditing such as:

1. Do the current database auditing tools have agent based audit and monitoring capabilities?
2. Is agent based real time system for database audit monitoring an appropriate proactive solution to overcome database performance overheads and audit log availability issues found in inbuilt database audit tools?
3. Are the database audit logs, file systems and log extraction process independent from the database system being audited?
4. What value additions does agent based audit monitoring system adds to database auditing and auditors?

1.5 Proposed Solution

The Agent based for real time database audit monitoring system utilizes agent properties and agent characteristics that is data driven execution, the ability to monitor local or remote data activities and events within the database, determining for itself the nature of database activities and SQL command statement being executed using production rules and regular expression parsing and communication protocols, the agent ability to enlist other agents including people to accomplish the alerting and notification tasks.

1.6 Agent Based System Algorithms

The agent based system for real time database audit monitoring uses using agent technology to capture network packets and will be built on JAVA packet capturing (JCAP) library to capture all packet to the database then passes control to the next level (Level one) agent for deep packet inspection through content (packet body) analysis. Several algorithms exists for deep packet inspection however in this project Advanced Algorithm for fast and scalable deep packet inspection and a fast multi-pattern matching algorithm are used by employing Content Addressed Delayed Input Deterministic Finite Automata (CD²FA) technique for parsing regular expression due to its suitability for low memory resource requirement.

In deep packet inspection both the header and the body of the packet are analyzed at the same time performing packet matching using fast multi-pattern matching algorithm due to their low memory cost ability (Jia Ni et al, 2007) . These algorithms offers a high speed performance and low memory cost besides they reduces false positive ratio by using hash function (Williams et al, 2008).

The research study uses SQL statement commands packets, SQL syntax algorithm in this case SQL parser is used to implement the execution of invalid SQL query statements and to join the command statements.

1.7 The scope of the Study

Database auditing monitoring was selected for the reason that database management system has become key to organizations operations hence a critical resource in the present world today. The need we employ agent based system in database auditing and monitoring is because database audit trails should include the individual SQL query, the database response, the timestamp, and, most importantly, the individual user or computer that accessed or changed database data. Multi-agent technology is considered a viable solution in achieving the research project objectives stated above in the following ways:

1. Database auditing need to move to the next level that supports real time/proactive auditing

2. The need to include audit monitoring of the SELECT, UPDATE, DELETE, and INSERT statements, SQL query command statements.
3. There is need for databases to offer high availability of audit logs
4. There need to employ a technology that overcomes performance overheads and other weaknesses of existing database auditing tools and solution.
5. The demand to have logical database audit data structure for database security, compliance and regulatory requirements.

1.8 Significance of the Study

For most organizations database management system has become the central main data and information repository. The financial systems and mission critical business operations have shifted from paper-based to digital based, database systems have from single systems to enterprise resource planning (ERP) systems however database auditing tools and solutions used still face a number of challenges mainly database performance problems, none real time, platform and database independency.

Software agent technology is an emerging and promising technology which attracts great interest in recent years, research and development is currently ongoing in the multi-agent domain, with anticipated benefits of additional new functionalities that agents brings to database auditing and monitoring including, intelligence, autonomy, scalability, logging of the information flows, real time alerts and notifications, non-proprietary, platform independence of audit records/data from the database itself and no database resource overhead.

1.9 Conceptual Model of Agent Based System for Real-Time Database Audit Monitoring

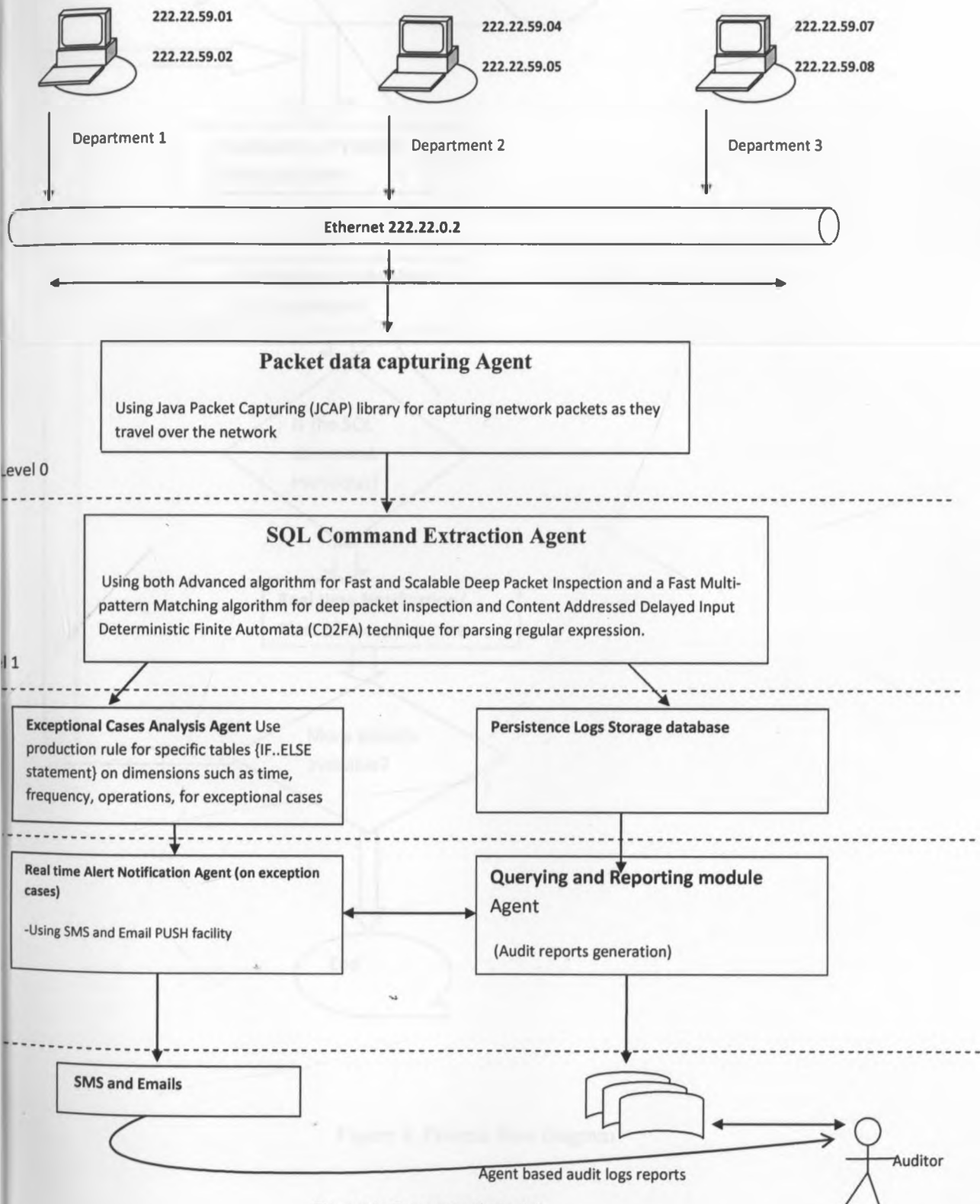


Figure 1: Conceptual model

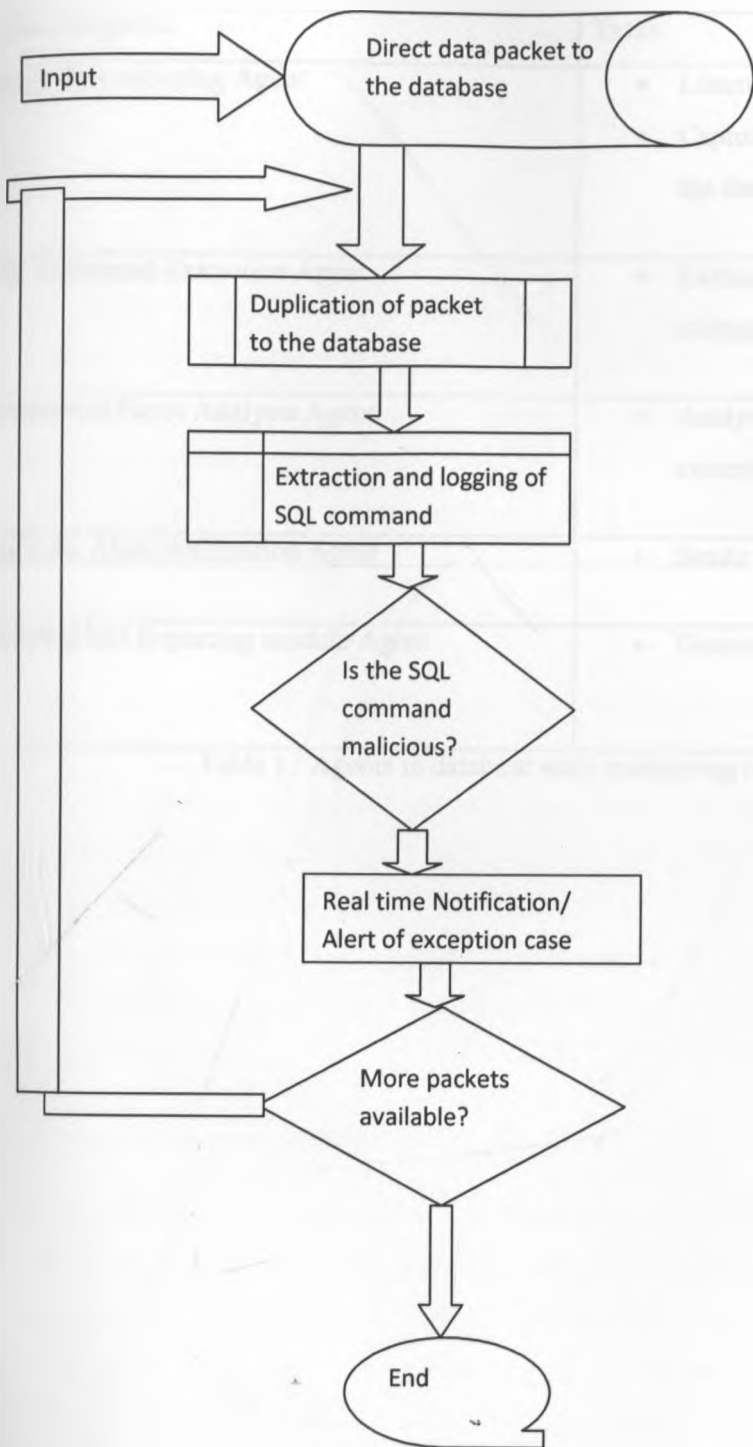


Figure 2: Process flow diagram

Agent categories	Tasks
Packet data capturing Agent	<ul style="list-style-type: none"> • Listen to all the open ports • Captures all packets from the network to the database
SQL Command Extraction Agent	<ul style="list-style-type: none"> • Extracts SQL commands for analysis and storage in a persistent database
Exceptional Cases Analysis Agent	<ul style="list-style-type: none"> • Analyses the SQL commands on exceptional cases
Real time Alert Notification Agent	<ul style="list-style-type: none"> • Sends alerts and notification
Querying and Reporting module Agent	<ul style="list-style-type: none"> • Generate audit reports

Table 1: Agents in database audit monitoring and their functions

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

From the inception of database age there has been significant development and implementation of database auditing and monitoring tools have been developed to assist both financial auditors and information system auditors including security risk managers. In attempting to analyze database audit data or information a number database tools and solutions are employed in different aspects of computing setup. Network monitoring tools for example, are used to identify problems to quantify expected performance. Most database monitoring tools are employed to monitor specific aspects of the database, for this reason several monitoring tools are employed in order to get full picture of the nature of activities taking place within the database.

It is noted that most database management systems (DMBS) have database auditing or “Logging” systems, including capability to log database transactions. This system appears to be adequate solution for capturing records of all the database activities in the sense that when audit log function is turned on, it can be set to capture a great deal of information, including, who connected to the database, who made changes to the database, what changes were made, and who accessed records. While this functionality is powerful, the main drawbacks are negative database performance and database and audit log availability should the database crash. As stated by (Herlands, 2007), it is also possible to turn off this functionality by privileged users.

2.1.1 Inbuilt Database Audit log

Inbuilt database audit log (also known as the audit system) function is built within the database, and is part and parcel of the database system though very effective and almost adequate but has adverse effect on database system’s performance. This is especially true when attempts are made to record every access to certain data; the constant reading and writing of audit data results in substantial disk input/output (I/O) on the database system, creating a bottleneck that significantly slows down database system performance. As argued by (Newman, 2009), that since auditing data is stored in the SYS.AUD\$ table, it ends up sharing disk space with user data, resulting into possible application downtime when log files get filled up.

2.1.2 Example Scenarios

Oracle is the leading database management system in use today across the globe, below are the configuration parameters of inbuilt auditing system in Oracle and summary of simulated performance analysis when audit trail is configured and turned on.

Parameter	Value	Description
audit_trail	DB	Write the standard audit content to sys.aud\$ table
	DB, EXTENDED	Write standard audit content to sys.aud\$ along with the SQL text and bind variable content that was executed for that SQL
	OS	Write the standard audit content to text files
	XML	Write the standard audit content and FGA audit content to an XML formatted file
	XML, EXTENDED	Write the standard audit content and FGA content to an XML formatted file along with SQL text and bind variable content.
audit_sys_operations	TRUE/FALSE	Audits all top-level SYSDBA and SYSOPER activity. These audit records are only written to OS files regardless of the audit_trail parameter setting.
audit_syslog_level	<FACILITY_CLAUSE. PRIORITY_CLAUSE>	Provides level information for the syslog.
audit_file_dest	<OS_DIRECTORY>	Specifies the OS directory location to write the OS and XML audit files

Table 2 – Oracle Database Auditing Parameters that define specific conditions that must take place for the audit to occur (Bednar, 2010).

Bednar (2010) argues from the analysis of throughput and additional CPU utilization after auditing function is turned on using Oracle database results tabulated in table 2 and 3 below from a real-world simulation scenario with 50% CPU System Load, that when audit is written to a text file it minimizes performance impact on the database however it still results in substantial performance effects.

This fact is supported by a summary of simulated results from a test created to generate approximately 250 audit records per second using the Oracle database standard database auditing command.

Audit Trail Setting	Additional Throughput Time <i>(Additional time used by the transaction after auditing was turned on)</i>	Additional CPU Usage <i>(Measured additional CPU after auditing was turned on)</i>
OS	1.39%	1.75%
XML	1.70%	3.51%
XML, Extended	3.70%	5.26%
DB	4.57%	8.77%
DB, Extended	14.09%	15.79%

Table 3: Oracle Database 11.2.01 Standard Audit Trail with 50% CPU System Load (Tammy, (2010).

A summary of test results for fine grained auditing, created to generate approximately 200 audit records per second is tabulated .

Audit Trail Setting	Additional Throughput Time	Additional CPU Usage
XML	3.66%	4.35%
XML, Extended	4.62%	9.09%
DB	6.60%	11.11%
DB, Extended	9.61%	20%

Table 4: Oracle Database 11.2.01 Fine Grained Audit Trail with 50% CPU System Load

From the results, it shows that writing audit records to Operating System file, has least impact to system resources (Tammy, 2010), however inbuilt auditing functions has performance overheads on the database, configuration nightmares, database platform dependent and lacking autonomy regardless of the audit trail file destination..

2.1.3 Fingerprinting Scheme

The main capability provided by the fingerprinting scheme (also called learning based or behavioral analysis) is that, under reasonable assumptions, it can embed and detect arbitrary bit-string marks in relational databases. This capability, which is not provided by prior techniques, permits this technique to be used as a fingerprinting scheme. The models demonstrate that fingerprints embedded by this scheme are detectable and robust against a wide variety of breaches including conspiracy attacks. Research efforts have been made to improve, among other things intelligent auditing monitoring solutions to rely on behavioral “fingerprinting” or “learning” and behavioral analysis. However these techniques often result in false positives and incomplete or inaccurate information. Once base lining of

database activity is completed, a task that often takes months of data collection, requires skilled and knowledgeable staff with privileged access or knowledgeable database users, in many cases must then invest a significant amount of time editing and refining the collected data to eliminate undesirable entries and false positives (Rakesh et al 2003) using this type of system in automatic mode often provides an organization with a false sense of security and results in an incorrect assumption that the system is smart enough to identify when a new user is performing an illegitimate activity in the database, without agent-based capabilities, this technique cannot provide a proactive solution.

2.1.4 Real-Time, Policy-Based Activity Monitoring

Arguably the most efficient and effective method of monitoring database activities is implementing a solution that relies on real-time activity analysis and policy based auditing and monitoring. Organizations that need to monitor database activities have shown a preference for deploying database monitoring solutions which are real time. While solution attempts to address the need for real time activity monitoring, policy-based solutions collect the data as defined by the monitoring policy, when the environment changes, the policy needs to be changed too, this ensures that all changes to the system are legitimate and planned for and that abnormal activity is detected (Herlands, 2007), practically it mean making frequent changes to the system as changes occur in the environment which is quite costly.

2.1.5 Agent Based Platform

The concept of agent based database audit monitoring involves agent software in which agents are developed and designed to achieve specific goals. Several agent development platforms exist including, Agent Builder (Acronymics, Inc., 2006), Aglet (IBM Research, 2002) and JADE (Telecom Italia Group, 2007), JACK (Agent Oriented Software Pty. Ltd., 2006) and more research is ongoing, in this research.

There are several agent development platforms that exist such as JADE for agent's development and FIPA for agent communication, JADE has been developed by the Telecom Italia Lab and the Agent and Object Technology Lab at the University of Parma and is selected based on two criteria. The research project study utilizes JADE because it is well-proven and scalable. It provides complete control to the agent framework (Bellifemine et al., 2007). Java Agent Development Environment (JADE) also simplifies the implementation of multi-agent systems through a middle-ware that complies with the Foundation for Intelligent Physical Agents (FIPA) specifications and through a set of graphical tools that supports the debugging and deployment phases

2.1.6 Multi-Agent Concept and Approach

The emergence of multi-agent technology is radically transforming software development, design and implementation of software solution. The Agent based system for real time database audit and monitoring is considered effective due to multi-agent capabilities. The desired optimal solution should be proactive, database platform independence and should include an up-to-date persistent log database. Our view of the demonstrated and implemented agent based system we believe that the system provide real time alert notification of database operations, independent audit logs from the database system, reduces performance overheads of critical resources of the database and offer availability of audit logs.

The audit data logs are analyzed, reported in real time through agent technique in a proactive manner showing exceptional cases .This provide proactive data of the “who, what, when, where, and how” of all database transactions.

2.1.7 Reviewing and Evaluation of Available Database Auditing Tools and Solutions

Inbuilt database audit log function	Fingerprinting/ Learning-Based scheme	Real-Time, Policy-based Activity Monitoring	Agent based system (Proposed)
The constant reading and writing of audit data results in substantial disk input/output operation slowing down the database (Herlands, 2007).	Although detects bit strings however it gives false positive (inaccurate and incomplete information) (Rakesh et al, 2003)	Collect the data as defined by the monitoring policy, when the environment changes, the policy need to be changed too (Bednar, 2010).	A proactive system, include an up-to-date persistent audit log database, with platform database independence (Natan, 2005)

Table 5: Summary analysis of available database audit and monitoring tools and solutions

CHAPTER 3

RESEARCH METHODOLOGY

The research project study was conducted in two phases, where phase one was for data collection through literature review and phase two involved system development. Research methods were critical in order to achieve the objectives outlined in section 1.3 above and include:

3.1 Data Collection Methods

3.1.1 Sources of Data

The Study utilized both secondary and primary data in generating additional facts on the subject. Literature review formed a major source of data; other methods involved included understanding and observation of the available database auditing tools and solutions. Other sources data has been from books, academic, journals, and internet.

3.1.2 Data Collection Tools

Three different methods were used to collect data and are outlined below; these methods were not used exclusive.

Interviews

Interviewing of the both conventional and information systems auditors was very necessary for gaining in depth understanding of database audit and monitoring domain. The thinking behind this tool was that interviews were used as a means of sampling the target users. Simple interviews questions were used with both conventional and system auditors and the data gathered provided insight into the development and implementation of the system in line with the objective.

Questionnaires

The questionnaires were used to supplement interviews and particularly to reach respondents that could not be available for face to face interviews, also for confidence and confidentiality of the information.

Experiments and observations

This method proved very useful, a number of tests and trials were done with existing database audit tools and the process of log extraction was observed.

3.2 Data Analysis Method

This research partially bases its findings through both quantitative and qualitative research methods for flexibility. During data gathering the choice and design of methods was constantly modified, based on progress of the system development. Qualitative research method helped to find and build on theories that explain the relationship between different outcomes and results from the system.

3.3 Multi-Agent Methodology

The system developed uses Prometheus methodology because it is detailed and complete in the sense of covering all activities required in developing intelligent agent systems. Three phases of Prometheus Methodology followed are outlined below.

i. *System specification* – Where the focuses is on identifying the basic functionalities of the system, along with inputs (percepts), outputs (actions) and any important shared data sources. Agent based system for real time audit monitoring involves 5 types of agents:

Type & level	Agent categories	Responsibilities
Level 0-Interface Agent	Packet data capturing Agent	<ol style="list-style-type: none"> 1. Listen to all the open ports 2. Captures all packets from the network to the database
Level 1- Extraction Agent	SQL Command Extraction Agent	Extracts SQL commands for analysis and storage in a persistent database
Level 2- Analysis Agent	Exceptional Cases Analysis Agent	Analyses the SQL commands on exceptional cases
Level 3 – Notification Agent	Real time Alert Notification Agent	Sends alerts and notification
Level 4- Reporting Agent	Querying and Reporting module Agent	Generate audit reports

Table 6: Agents types, categories and responsibilities

ii. *Architectural design* – Whereby the outputs from the previous phase is used by the next agent level in the system and show how they interact

iii. *Detailed design* – Involves looking at the internals of each agent and how it accomplishes its tasks within the overall system. This is designed by modeling the interaction between the different agents.

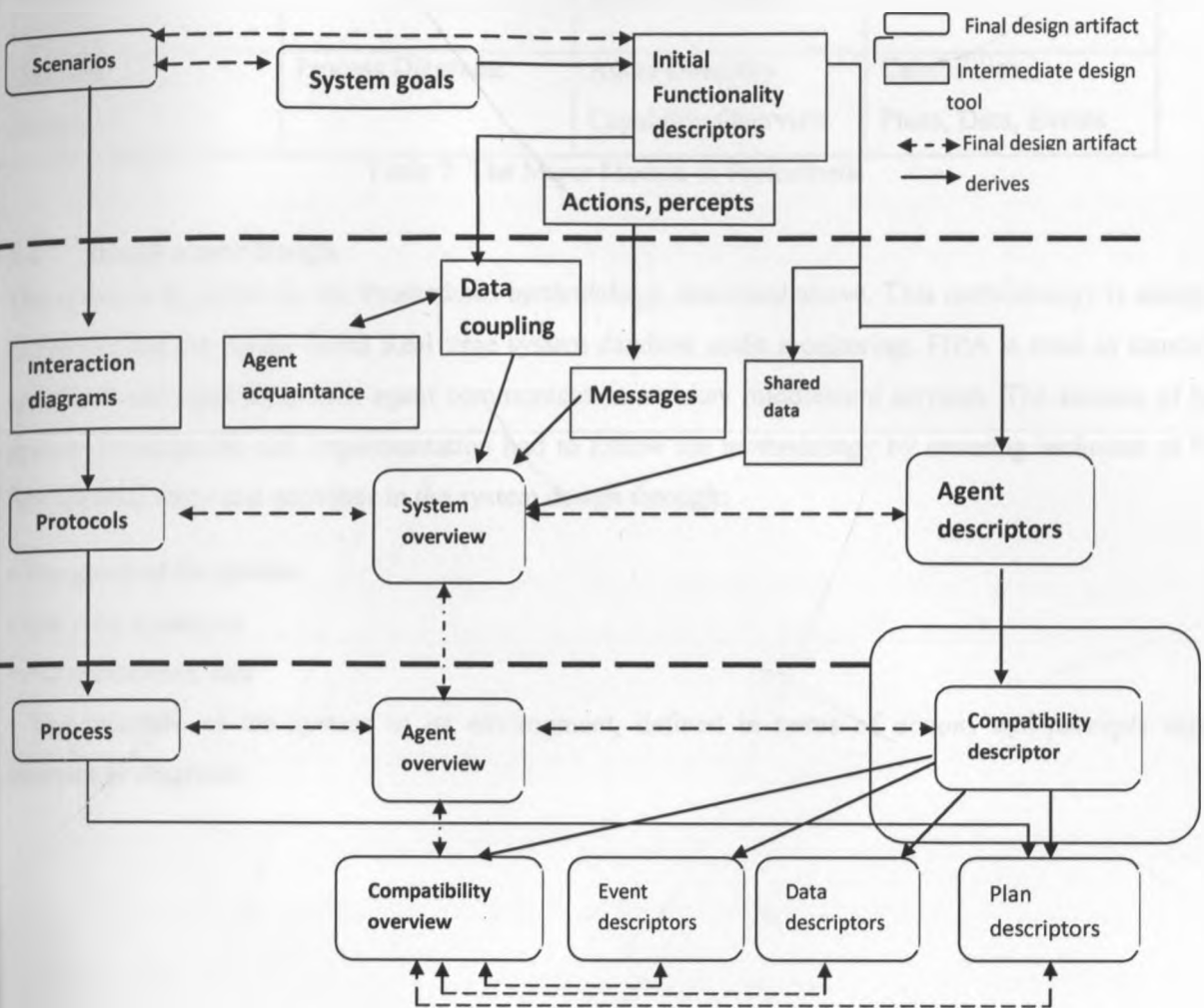


Figure 3: Prometheus Methodology

	<i>Dynamic Models</i>	<i>Structural Overview Models</i>	<i>Entity Descriptors</i>
<i>System Specification</i>	Scenarios	Goals	Functionalities actions & percepts

<i>Architectural Design</i>	(interaction diagrams) Interaction Protocols	(coupling diagram) (agent acquaintance) System Overview	Agents Messages
<i>Detailed Design</i>	Process Diagrams	Agent Overview Capability Overview	Capabilities Plans, Data, Events

Table 7: The Major Models of Prometheus

3.4 Multi-Agent Design

The research is guided by the Prometheus methodology, described above. This methodology is adopted in developing the Agent based Real time system database audit monitoring. FIPA is used as standard specifications supporting inter-agent communication and key middleware services. The success of the system development and implementation had to follow the methodology by ensuring inclusion of the fundamental steps and activities in the system design through:

- The *goals* of the system
- *Use case scenarios*
- *Functionalities*, and
- The interface of the system to its environment, defined in terms of *actions* and *percepts* using interaction diagrams

CHAPTER 4

ANALYSIS AND DESIGN

4.1 Database Auditing Monitoring Requirements Analysis

Database auditing monitoring task involves monitoring and recording of activities that occurs in the database. Most Database Management systems have been enhanced with inbuilt abilities to keep an audit log; these inbuilt databases auditing function provides a set of auditing capabilities and varies from one database management system to another. These capabilities are implemented as a system that writes activity to tables, log files, or even in the Event Viewer subsystem which records login attempts status. The audit logs contain a variety of data information including, username, terminal, timestamp, object owner, object name and action name.

4.2 System Specification

System specification begins with a rough idea of the system, which may be simply a few paragraphs of rough description, and proceeds to define the requirements of the system in terms of:

4.2.1 Functional Requirements

The desired system should be able to perform the following tasks:-

1. Capture data packets from the Local Area Networks
2. Extract the SQL command statements from the captured data packets.
3. Analyze exceptional cases on the SQL command using production rules of the IF...ELSE statements
4. Store the logs captured of SQL statements into a persistent independent database system
5. Provide real-time notification through short messaging system (SMS) and electronic email system using a PUSH facility on analyzed exceptional Cases.
6. Generate system audit reports

4.2.2 Scenarios

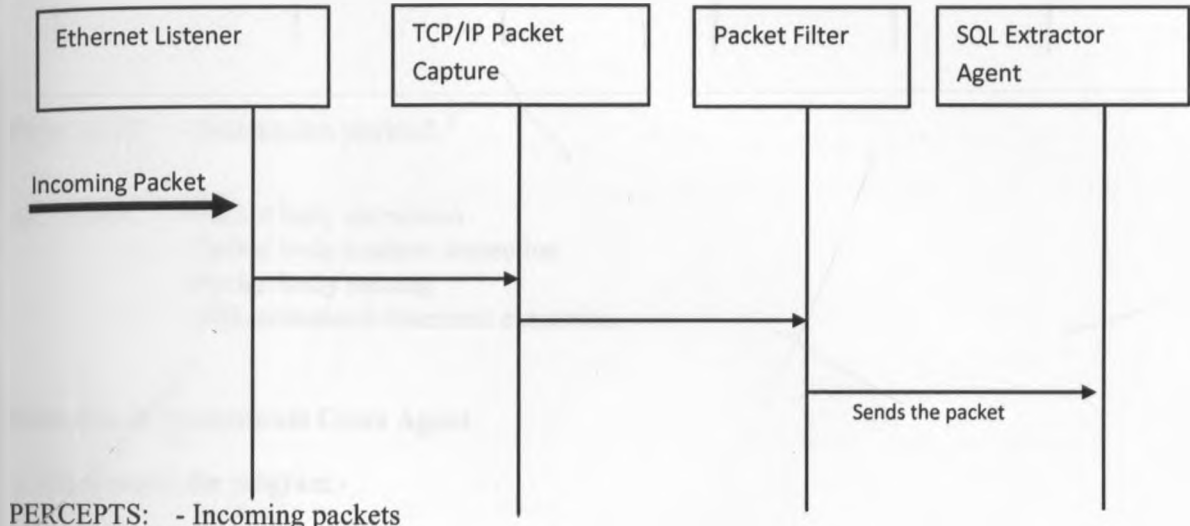
Scenarios 1: Packet Data Capturing Agent

The agent program will be listening on a specific Ethernet port connected on the local area network (LAN), it is an Ethernet Listener

The program will capture TCP/IP packets from the Ethernet port.

The program will filter all the packets captured whose destinations are to the specific internet protocol (IP) of the database server.

Interaction Diagram for Packet Data Capturing Agent



PERCEPTS: - Incoming packets

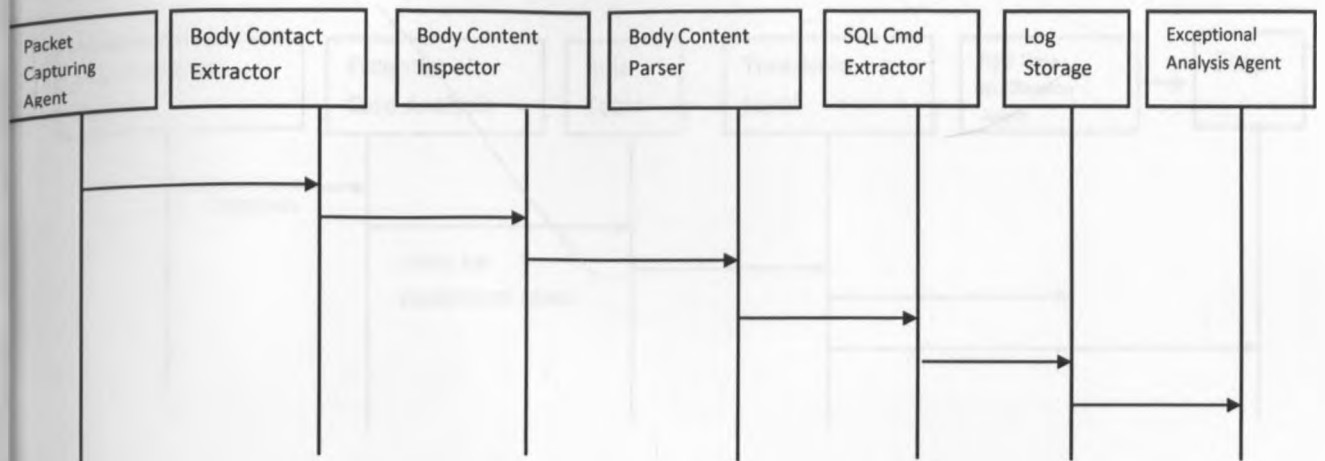
ACTIONS: - Packet capturing
- Packet filtering

Scenarios 2: SQL Command Extraction Agent

In this scenario the program:-

- Receives a packet from the packet capturing agent
- Extracts the body content of the packet
- Inspects the body content of these packets
- Parses the body content text
- Extracts the SQL statement in the packet body
- Sends the SQL statement packets to the persistent database
- Sends the SQL statement to the Exceptional cases analysis agent

Interaction Diagram for SQL Command Extraction Agent



PERCEPTS: - Data packet payload

ACTIONS: - Packet body extraction
- Packet body content inspection
- Packet body parsing
- SQL command statement extraction

Scenarios 3: Exceptional Cases Agent

In this scenario the program:-

Receives SQL Command from SQL extraction Agent

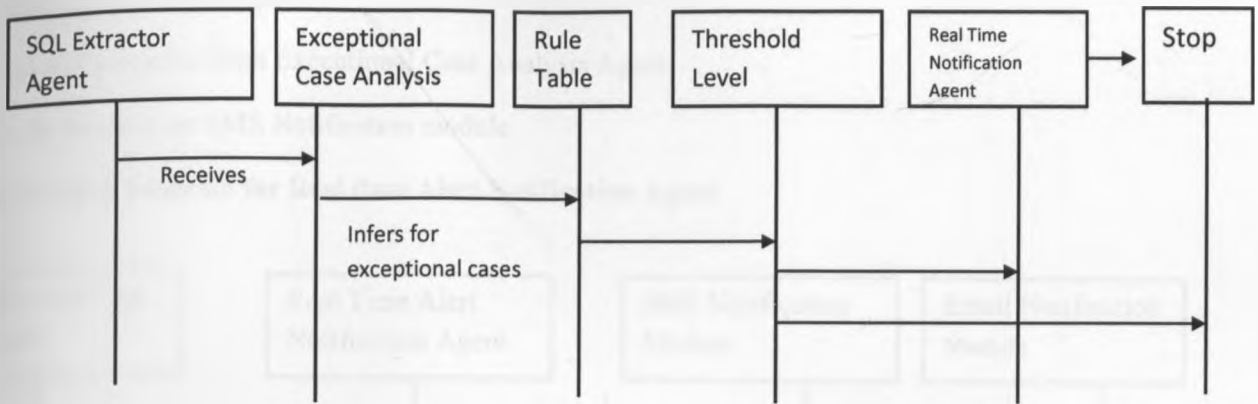
Infers from the production rule table, if the SQL Command rule meets the threshold anomaly, if it meets the threshold **THEN...**

Flag the result as anomaly

...**ELSE** ignore the case

If flagged as anomaly then send to Real time Notification agent

Interaction Diagram for Exceptional Cases Analysis Agent



PERCEPTS: - SQL Statement Extractions received

ACTIONS: - Inference

Scenarios 4: SQL Command Extraction Agent Database Persistent Module

In this scenario the program should:-

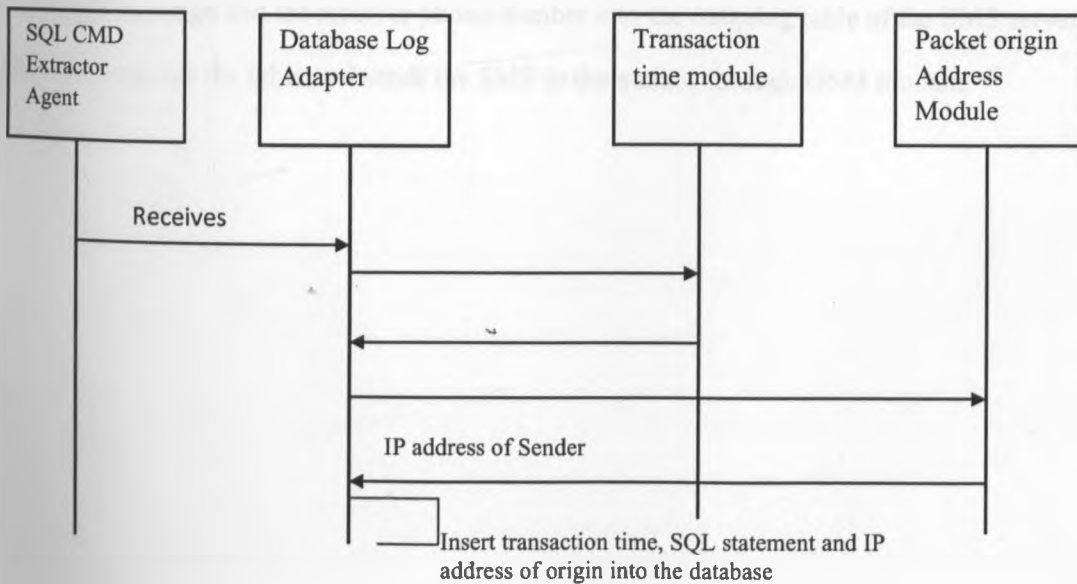
Receives SQL command statements from Extractor Agent

Captures the transaction time

Captures the source Internet Protocol (IP) address of the originating SQL command statement

Records the transaction time and the IP address origin in the log table

Interaction Diagram for SQL Command Extractor



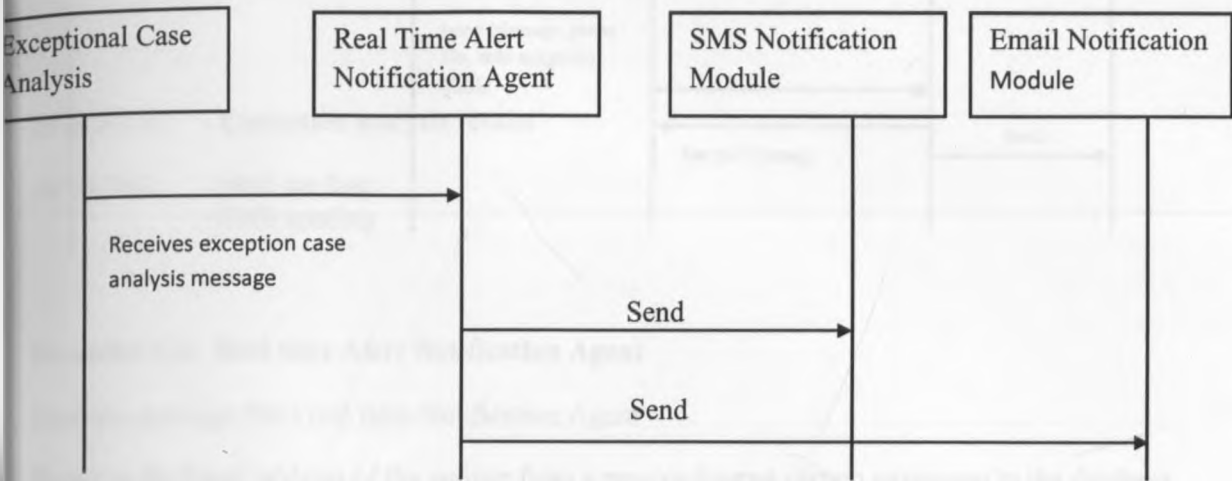
Scenarios 5: Real time Alert Notification Agent

In this scenario the program:-

Receives messages from Exceptional Case Analysis Agent

Sends the message SMS Notification module

Interaction Diagram for Real time Alert Notification Agent



Scenarios 5.1: Real time Alert Notification Agent

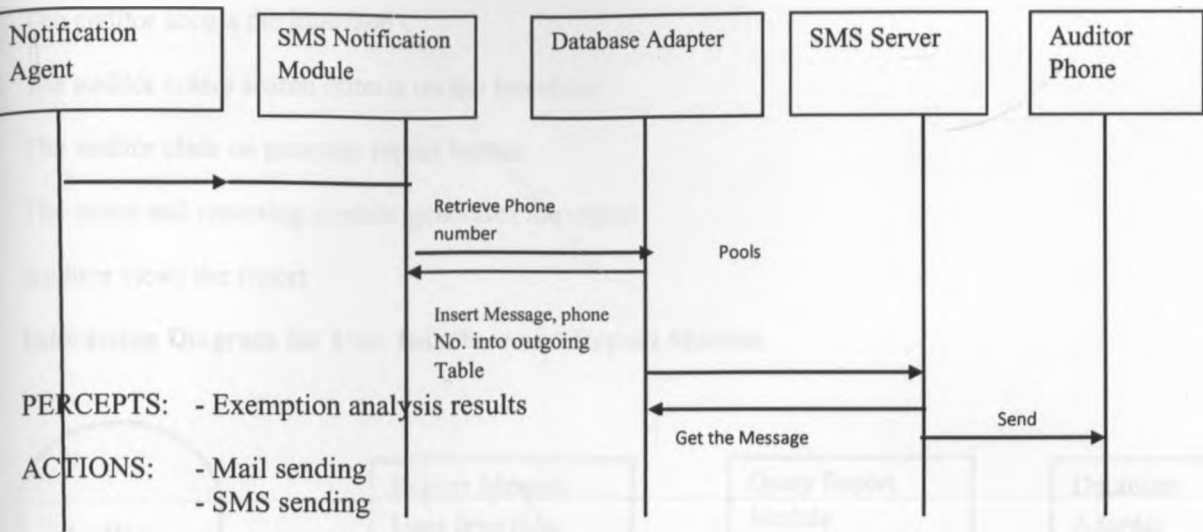
Receives the test message from Real time Notification Agent

Retrieves the Phone number of the Auditor from the preconfigured system parameters in the database

Inserts the message and the receiver phone number into the outgoing table of the SMS server

SMS server pools the table and sends the SMS to the auditor through GSM modem

Interaction diagram for Real time Alert Notification Agent



Scenarios 5.2: Real time Alert Notification Agent

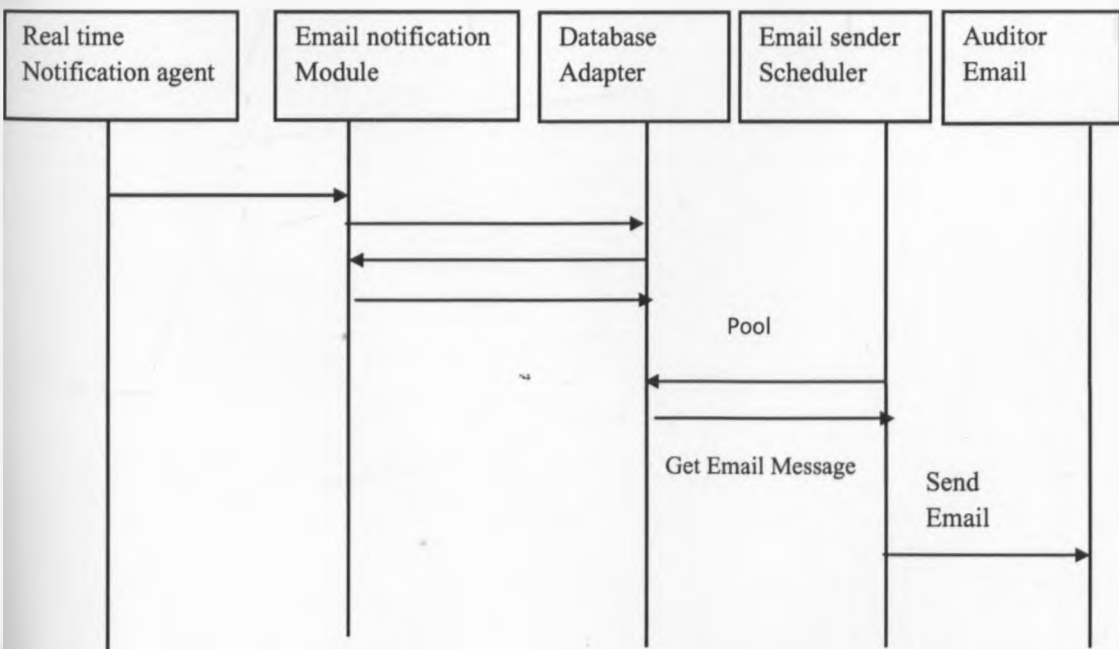
Receives message from real time Notification Agent

Receives the Email address of the auditor from a pre-configured system parameter to the database

Insert the message and receiver's email address into the outgoing email table

Mail sender scheduler will pools the outgoing email table and send the email

Interaction diagram for Real time Alert Notification Agent



Scenarios 6: User Interface and Report Module

The auditor access the interface

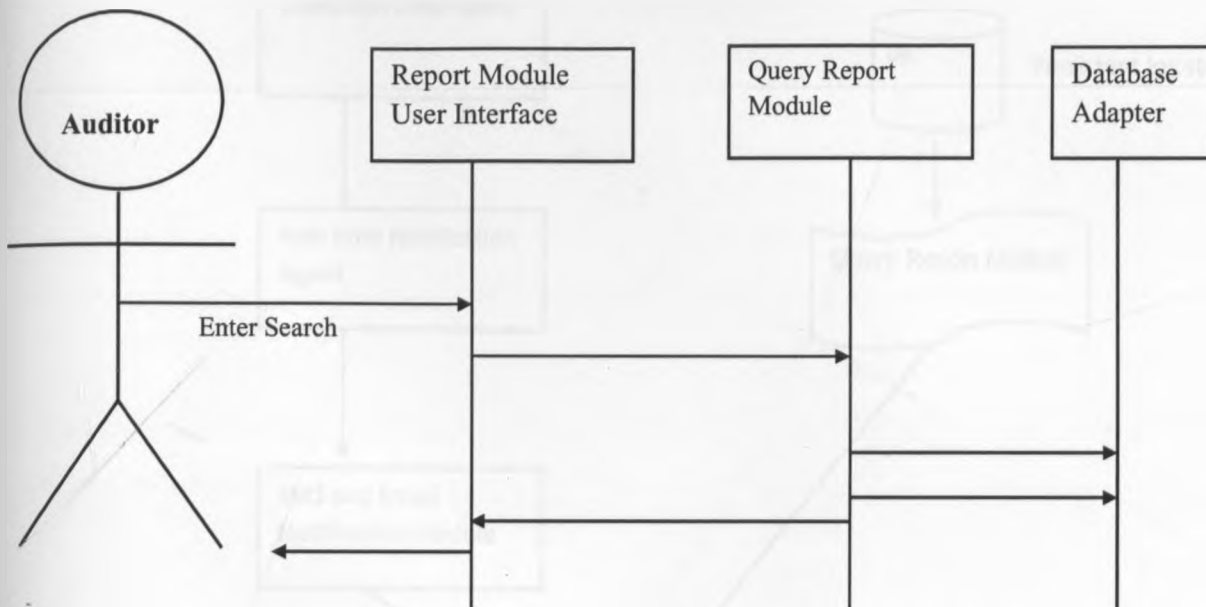
The auditor enters search criteria on the interface

The auditor click on generate report button

The query and reporting module generates the report

Auditor views the report

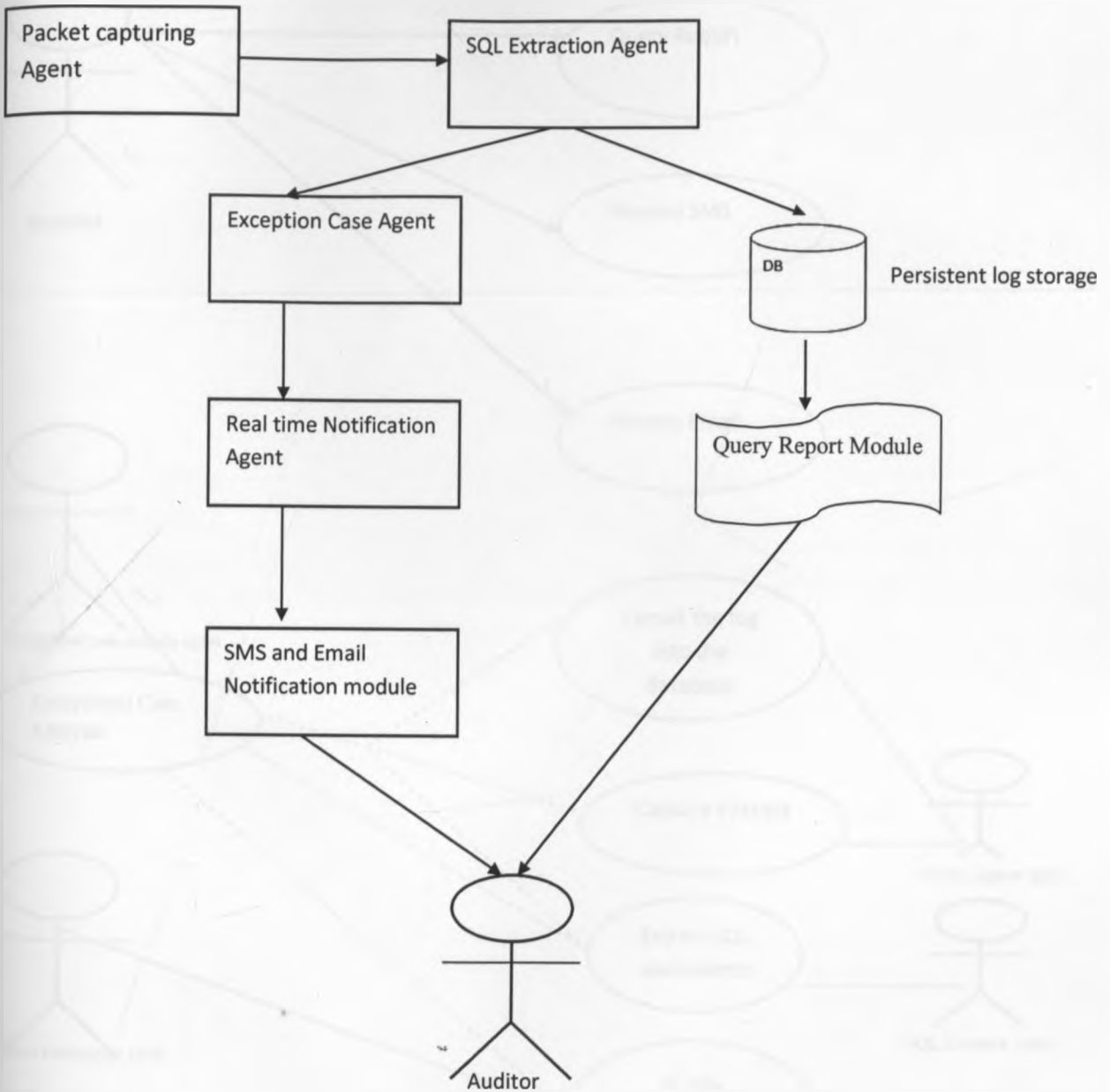
Interaction Diagram for User Interface and Report Module



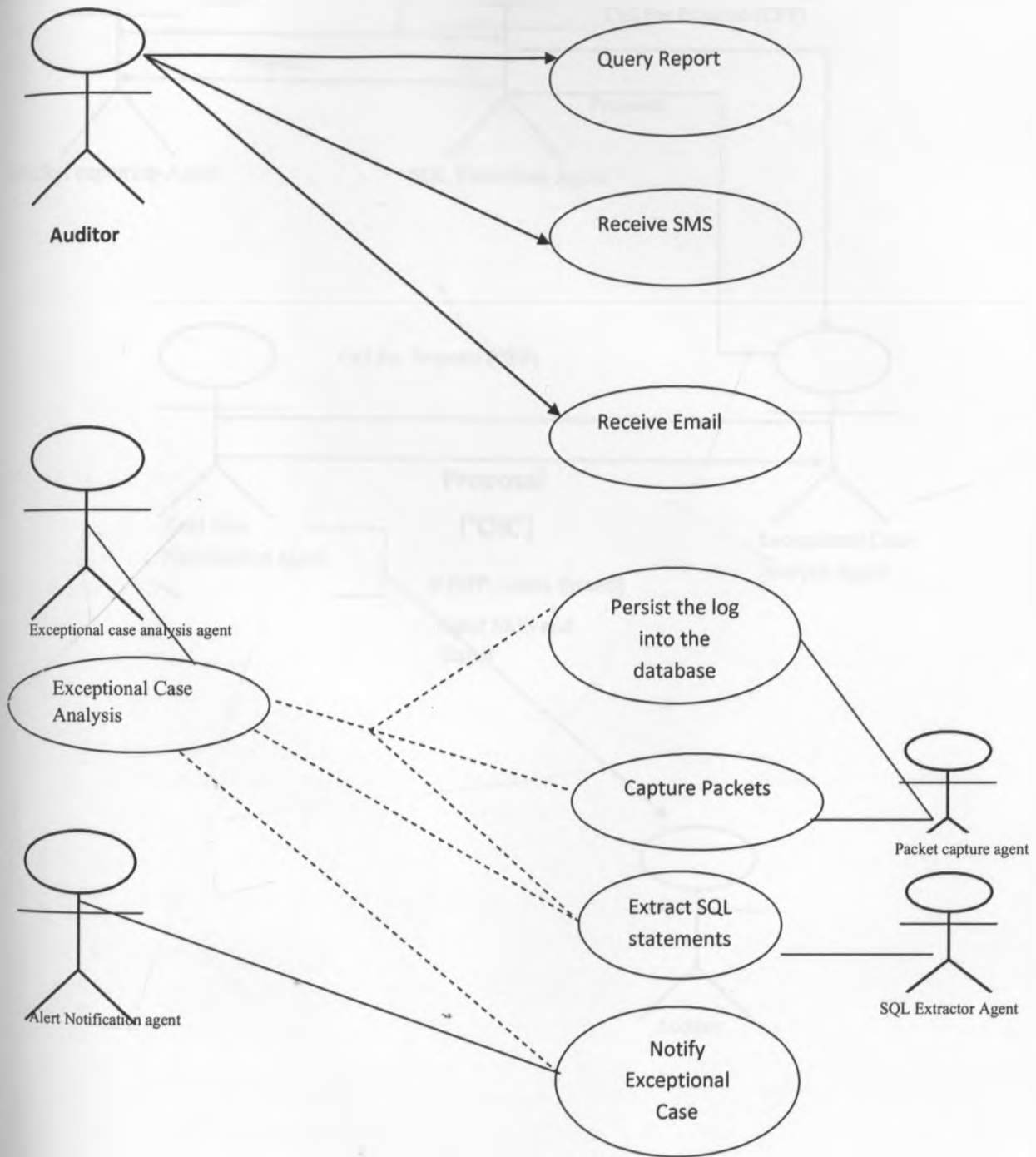
4.3 Architectural Design

The architectural design describes overall system view, communication protocols and the relationships between the agents through agent descriptors.

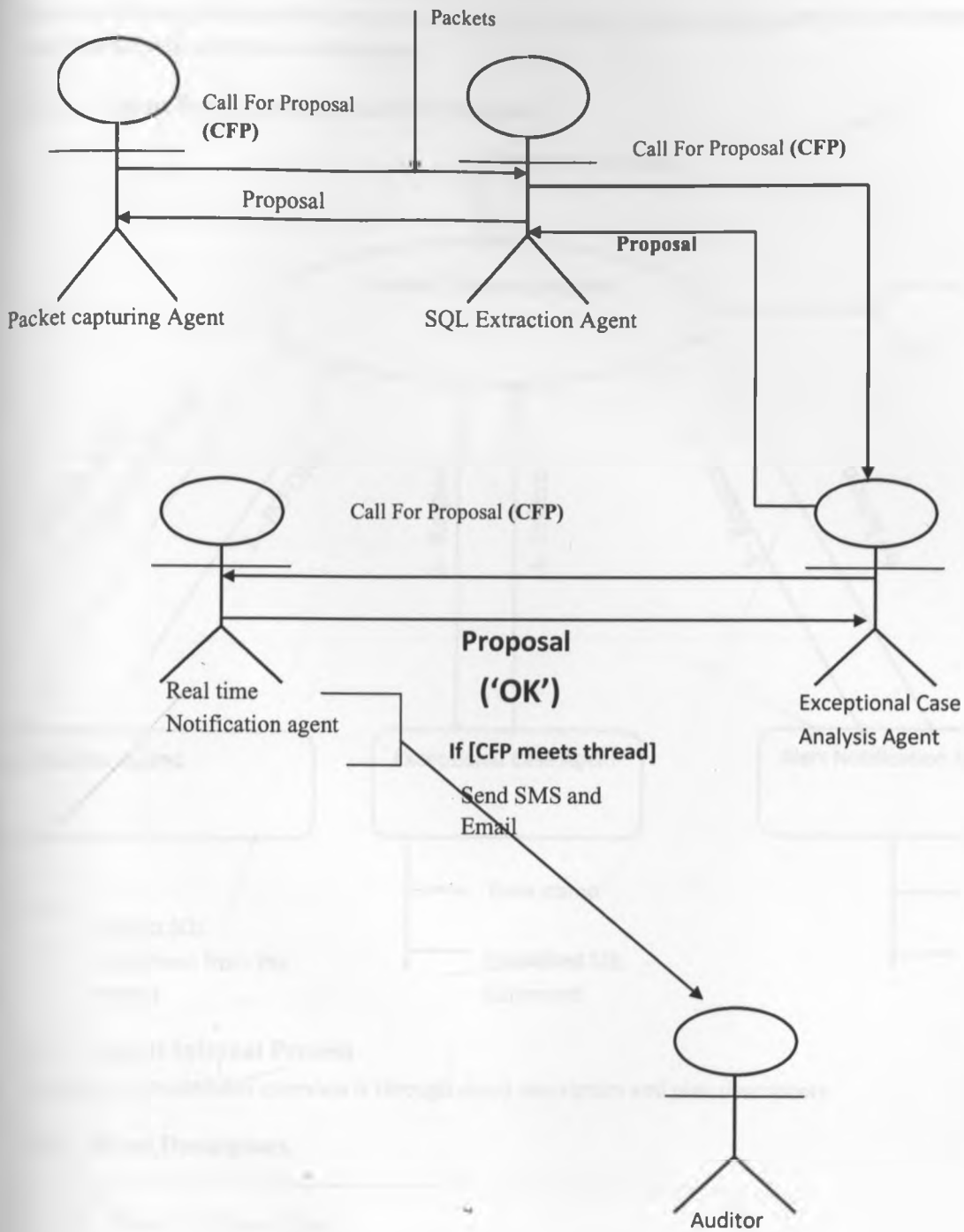
4.3.1 Agents System Overview



4.3.2 Agents Acquaintances using Use Case Diagram



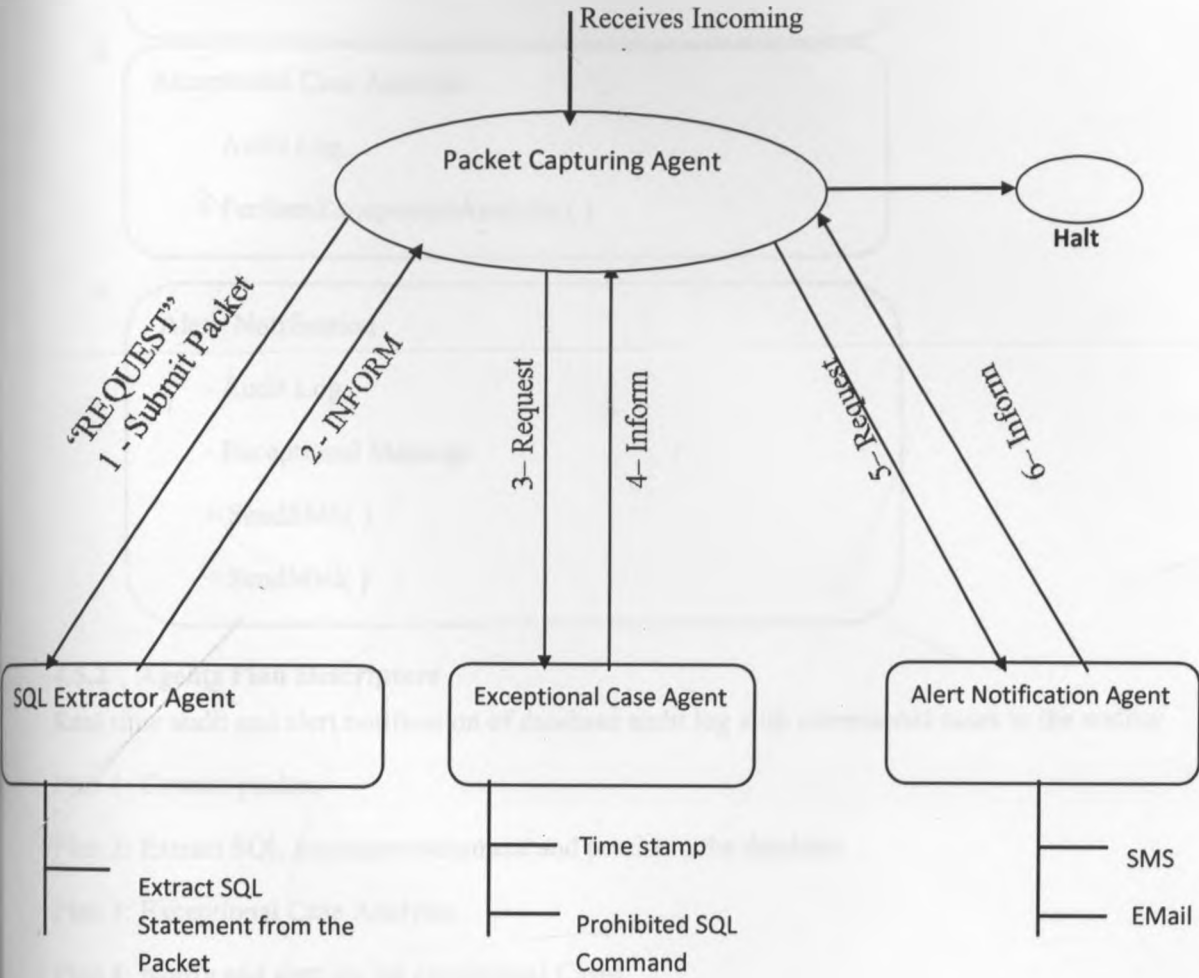
4.3.3 Agent Messages Communication



4.4 Agents Detailed Design

The detailed design describes the process, agent overview, event descriptors and plan descriptors showing their relationship with the overall system.

4.4.1 Agent Based System Overview Diagram

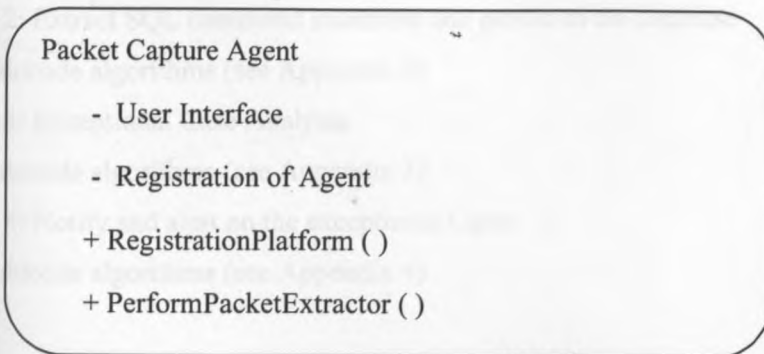


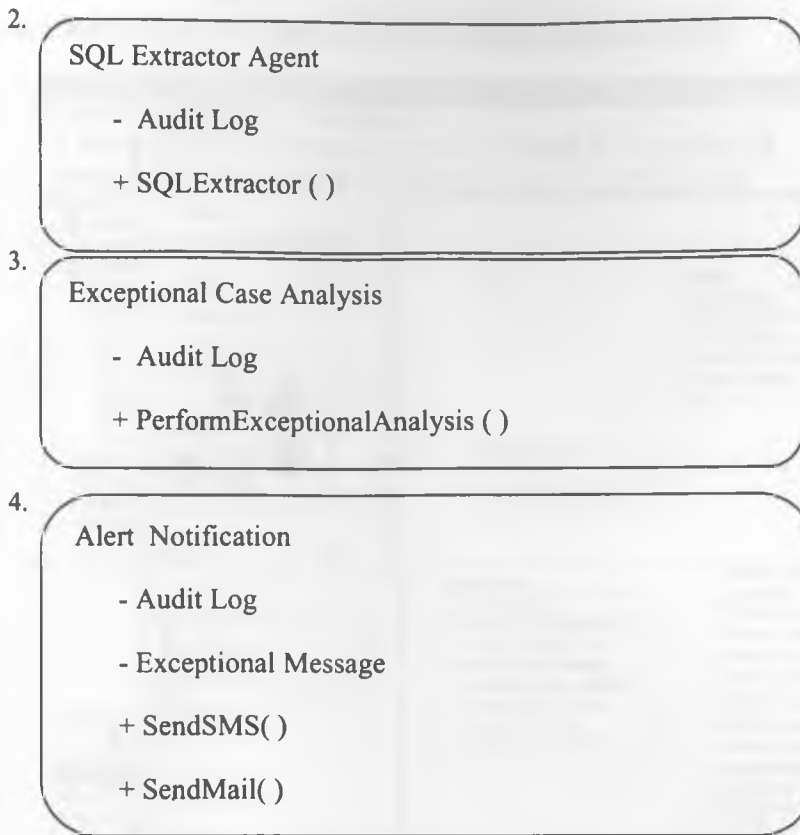
4.5 Agents Internal Process

The agents' compatibility overview is through event descriptors and plan descriptors.

4.5.1 Event Descriptors

1.





4.5.2 Agents Plan Descriptors

Real time audit and alert notification of database audit log with exceptional cases to the auditor

Plan 1: Capture packet

Plan 2: Extract SQL command statement and persist to the database

Plan 3: Exceptional Case Analysis

Plan 4: Notify and alert on the exceptional Cases

4.6 Algorithms

Step1: Capture packet

Pseudocode algorithms (see Appendix 1)

Step 2: Extract SQL command statement and persist to the database

Pseudocode algorithms (see Appendix 2)

Step 3: Exceptional Case Analysis

Pseudocode algorithms (see Appendix 3)

Step 4: Notify and alert on the exceptional Cases

Pseudocode algorithms (see Appendix 4)

4.7 Database Design

This database contains the Agents description fields and agents details.

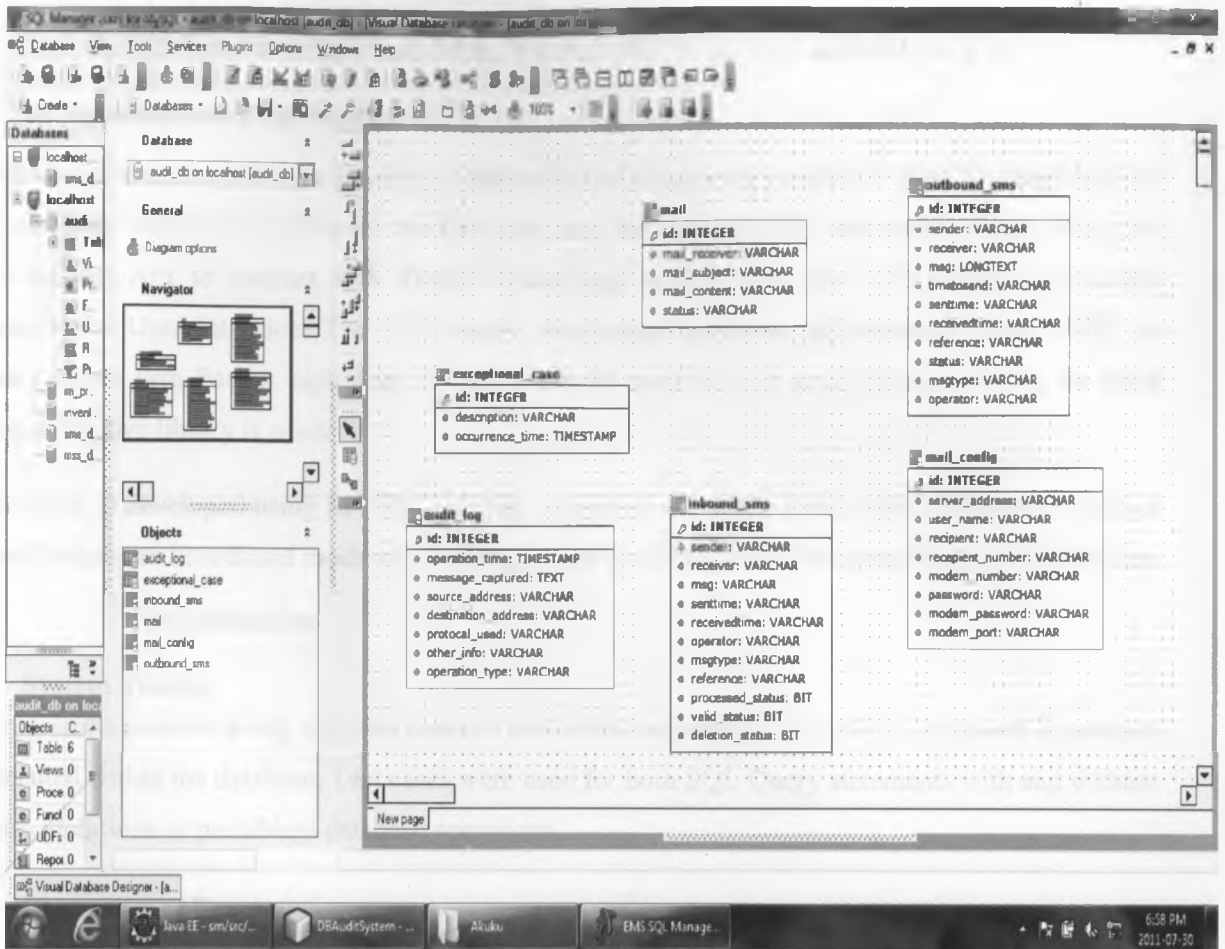


Figure 4: Database design

CHAPTER 5

SYSTEM IMPLEMENTATION AND RESULTS

5.1 Implementation of the System

The system has been implemented using a combination of frameworks under the Java Standard Edition (JSE) platform, Jasper for reports for the front end and the back end has been implemented using the java persistent API to interact with DBMS underlying eclipse data link API, Swing Application framework for User Interface. The Multi-agent component has been implemented using JADE, to capture packets Java Packet capturing (JCAP) library is used and for email java mail API, for SMS sending SMS java library is used.

The database is developed using MYSQL database version 5-4-3 while Email and SMS communication protocol Safaricom broadband modem and Google mail for Small Mail Transport Protocol (SMTP) are used.

5.2 System Testing

The program was tested using trial test cases of the actual happenings when SQL command statements are executed within the database. Test cases were used for both SQL Query statements with and without database violations or prohibited database operations.

5.3 Discussion of Results

5.3.1 Challenges Facing Database Auditing

Database auditing involves monitoring and recording of database user actions and can be based on individual actions, such as the type of SQL statement executed. Most databases applications (DMBS) have inbuilt audit capabilities and stores auditing information (records/logs) when specified elements in databases are accessed or altered. However these inbuilt database audit solutions have negative database performance such as response time, locking of log file system, database unavailability, crashing of file system, corruption of the log files and lack real time or proactive capabilities. Existing audit trails are much more useful for detecting what was changed, rather than what was accessed (Adrian Lane, 2010). In view of database audit requirement for security, compliance and regulatory, software agent system is the solution to database auditing and monitoring making it the preferred approach.

5.3.2 Evaluation of Available Database Auditing Tools and Solutions

Database auditing tools and solutions have developed over time as summarized below.

Inbuilt database audit log function	Fingerprinting/ Learning-Based scheme	Real-Time, Policy-based Activity Monitoring
The audit data is written to a table within the database itself usually results in substantial disk input/output operation slowing down the database.	Detects bit strings but do give false positive (inaccurate and incomplete information).	Collect the data as defined by the monitoring policy, however when the environment changes, the policy need to be changed too.

Table 8: Summary of evaluation of available database auditing tools and solutions

5.3.3 System Results

The Agent based system for real time database audit monitoring is able to monitor and perform real time alert notifications of all exceptional database operation cases as they happen. The diagram below shows how the agents captures packets, extracts SQL command statements, analyses exceptional database operation cases and performs alert notification where the operations are of violation or prohibitory nature.

Alert and Notification monitoring Agent Communication

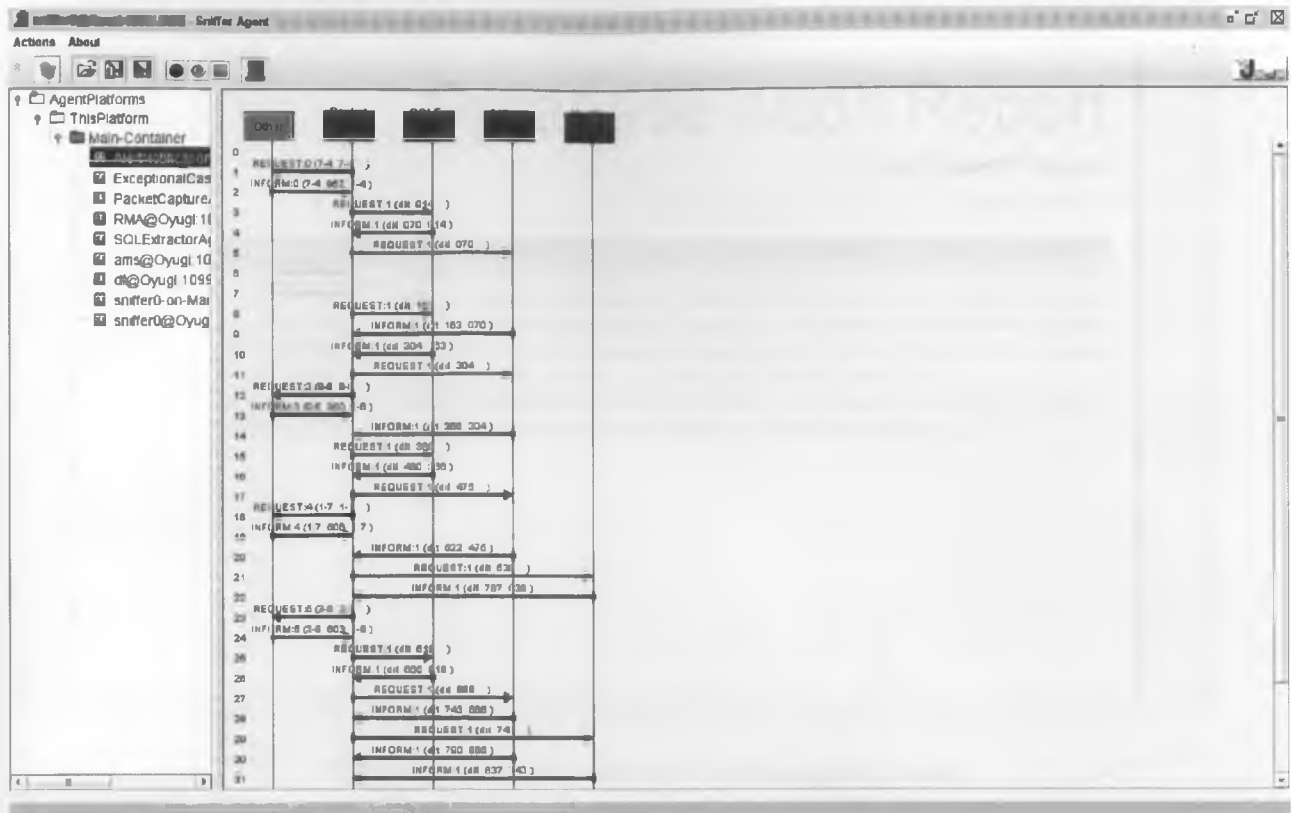
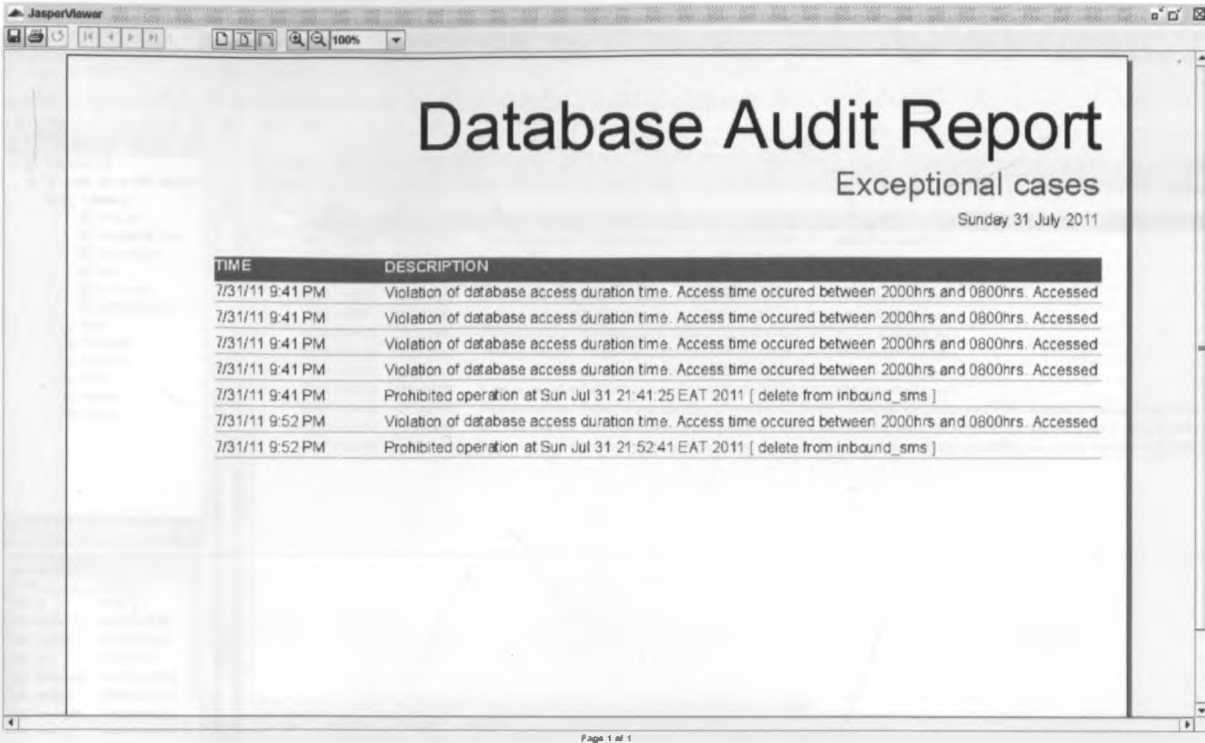


Figure 5: Exceptional analysis agent receives notification and send the SMS and email to the auditor

Database Audit Report

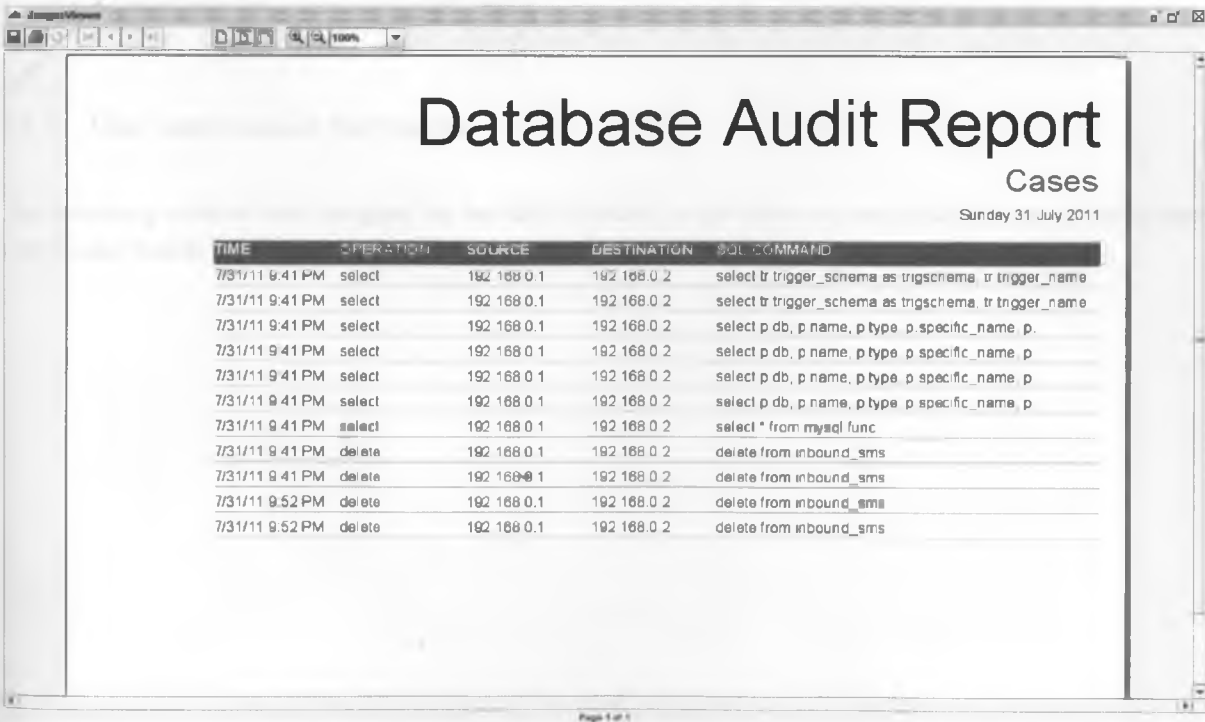
Formatted audit report on exceptional cases



TIME	DESCRIPTION
7/31/11 9:41 PM	Violation of database access duration time. Access time occurred between 2000hrs and 0800hrs. Accessed
7/31/11 9:41 PM	Violation of database access duration time. Access time occurred between 2000hrs and 0800hrs. Accessed
7/31/11 9:41 PM	Violation of database access duration time. Access time occurred between 2000hrs and 0800hrs. Accessed
7/31/11 9:41 PM	Violation of database access duration time. Access time occurred between 2000hrs and 0800hrs. Accessed
7/31/11 9:41 PM	Prohibited operation at Sun Jul 31 21:41:25 EAT 2011 [delete from inbound_sms]
7/31/11 9:52 PM	Violation of database access duration time. Access time occurred between 2000hrs and 0800hrs. Accessed
7/31/11 9:52 PM	Prohibited operation at Sun Jul 31 21:52:41 EAT 2011 [delete from inbound_sms]

Figure 6: Formatted audit report on exceptional cases

Formatted database audit report for all cases



TIME	OPERATION	SOURCE	DESTINATION	SQL COMMAND
7/31/11 9:41 PM	select	192.168.0.1	192.168.0.2	select tr trigger_schema as trigschema, tr trigger_name
7/31/11 9:41 PM	select	192.168.0.1	192.168.0.2	select tr trigger_schema as trigschema, tr trigger_name
7/31/11 9:41 PM	select	192.168.0.1	192.168.0.2	select p db, p name, p type p specific_name p
7/31/11 9:41 PM	select	192.168.0.1	192.168.0.2	select p db, p name, p type p specific_name p
7/31/11 9:41 PM	select	192.168.0.1	192.168.0.2	select p db, p name, p type p specific_name p
7/31/11 9:41 PM	select	192.168.0.1	192.168.0.2	select * from mysql func
7/31/11 9:41 PM	delete	192.168.0.1	192.168.0.2	delete from inbound_sms
7/31/11 9:41 PM	delete	192.168.0.1	192.168.0.2	delete from inbound_sms
7/31/11 9:52 PM	delete	192.168.0.1	192.168.0.2	delete from inbound_sms
7/31/11 9:52 PM	delete	192.168.0.1	192.168.0.2	delete from inbound_sms

Figure 7: Formatted database audit report for all cases

Alert Messages to the Auditor

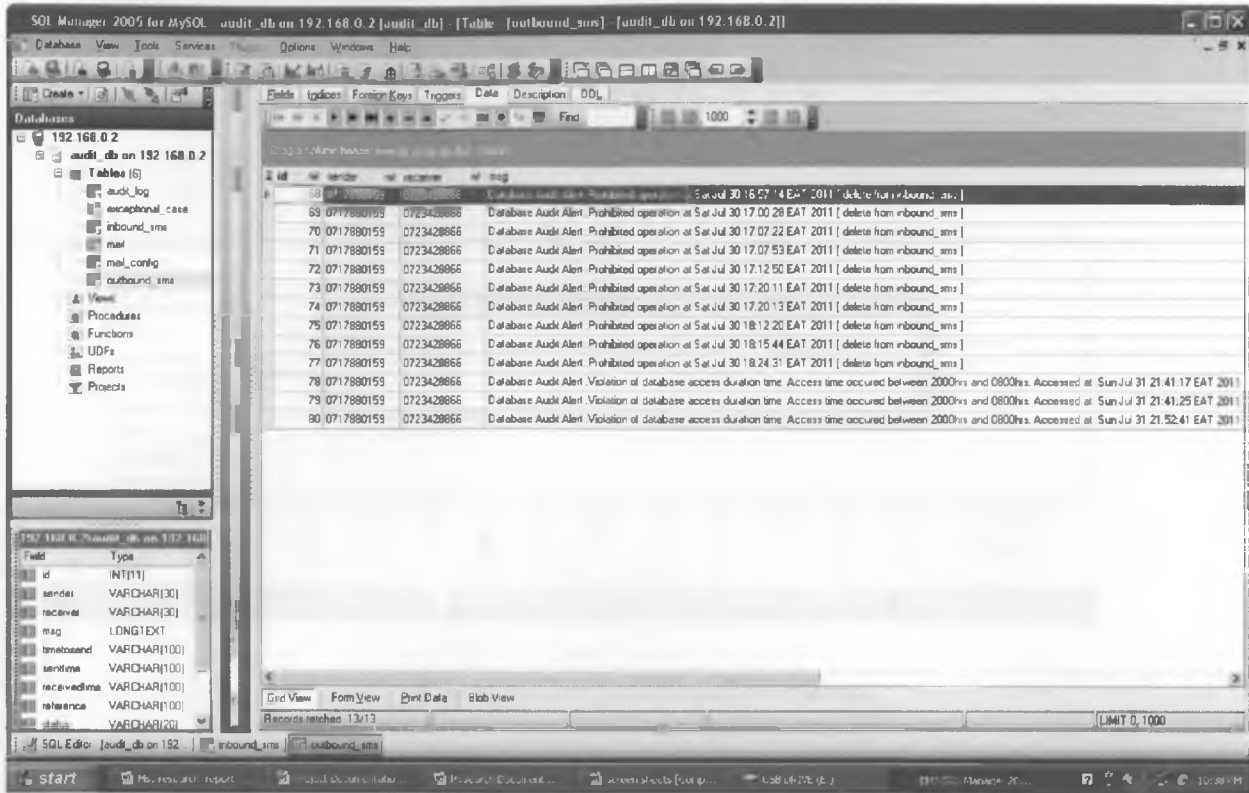


Figure 8: alert messages to the auditor

5.3.4 User Interfaces of the System

The following screens were designed for the user interface, in the below screen the auditor generates a report for a particular period for exceptional cases.

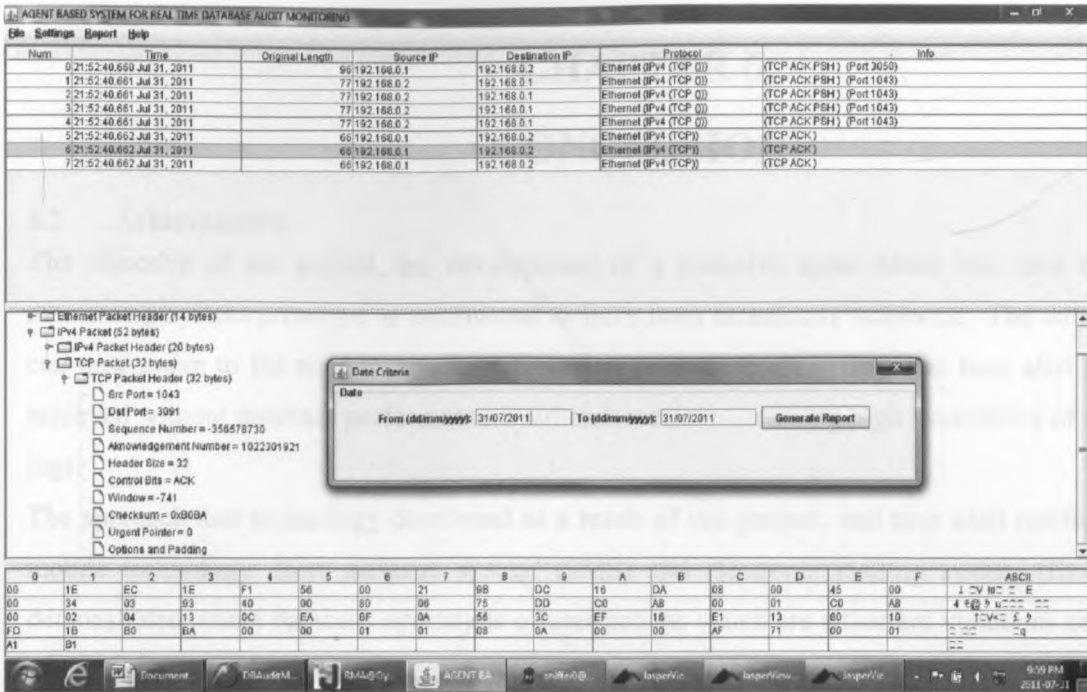


Figure 9: User Interface of the system

The interface below provides the auditor with the opportunity to generate different reports based on different criteria of database operations.

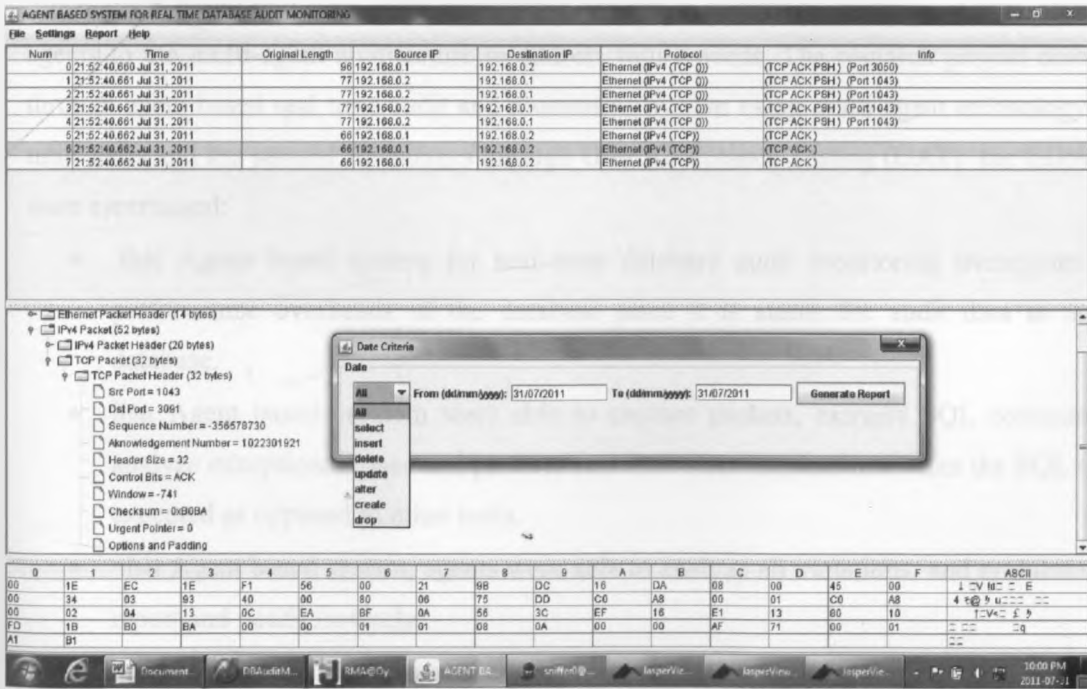


Figure 10: User Interface of the system

CHAPTER 6

CONCLUSION

6.1 Achievements

The objective of the project, the development of a proactive agent based real time database audit monitoring system/prototype is considered to have been technically achieved. The database auditing can now move to the next level where proactive database auditing and real time alert notification is achieved without database performance overheads while maintaining high availability of database audit logs.

The approach and technology developed as a result of this project, real time alert notification through mobile technology short message system (SMS) and electronic mailing system (Email) has been demonstrated where database operations of user actions which are either are violations or prohibited in nature are extracted, analyzed and the auditor receives real time alerts and notification.

Agent based database audit monitoring implementation enabled the faster analysis of database operations and exceptional cases as they happen. Auditors are able have visibility into the database transaction without performance overheads to the database and does not require cumbersome audit log extraction characterized by inbuilt database audit tools.

Since agent based real time audit monitoring system is inherently modular, it was easier to add new agents to the multi-agent architecture to address future needs. The research project demonstrated that through agent based real time audit and monitoring system using multi-agent technology provides real time alerts and has several advantages through User Acceptance Testing (UAT) the following feedback were ascertained:

- that Agent based system for real time database audit monitoring overcomes the database performance overheads of the database since it is stores the audit data in an independent database.
- that Agent based system were able to capture packets, extracts SQL commands statement, analyze exceptional cases and perform real time alert notifications when the SQL commands are executed as opposed to other tools.
- that Agent based system, agents were able to analyze all violations and prohibitory operations, report and notify instantly.
- that the Auditor's work was enhanced since there is no need for audit extraction exercise.

6.2 Research Contributions

The research has focused on adding value to database auditing, this research project demonstrates the agent based technology development approach in database auditing with agent capabilities and characteristics notably data driven execution. The agent technology enables agents' communication and coordination considerations that proved instrumental in the success of this system.

6.3 Recommendation/ Future work

The following improvements/functionality can be added to agent based database auditing:

1. The system should incorporate new databases model that do not use SQL query such as graph-based databases and geospatial databases.
2. A plug-in application programming interface with SQL commands to be added as additional functionality within the system.
3. The deployment configuration access includes other OSI layers for profiling.

6.4 Assumptions and limitations

This research project, the system is built with certain assumptions and limitations, the communication of the agent application and the database is purely based on SQL command query only, in case of new database model that do not use SQL query such as graph-based databases, geospatial databases the system may not apply, however the SQL commands can be applied as a plug-in. The deployment configuration access is through the network, Packets and SQL command statement extraction is only through the network connection and any Database that is not connected to the network cannot be audited.

References

1. Adrian Lane, 2010, Learn about database security auditing tools: Information Security Magazine, October , 2010
2. Sushila Nair, 2008, Journal Online: The Art of Database Monitoring, Information Systems Control Journal, ISACA, 2008
3. Jia Ni , Chuang Lin, Zhen Chen, Peter Ungsunan, 2007, A Fast Multi-pattern Matching Algorithm for Deep Packet Inspection on a Network Processor: Department of Computer Science, Research Institute of Information Technology Tsinghua University, Beijing ,International Conference on parallel processing (ICPP2007), IEEE Computer Society, 2007 page 1
4. John Williams, Sailesh Kumar, Jonathan Turner, 2008, Advanced Algorithms for Fast and Scalable Deep Packet Inspection: Cisco Systems and Washington University journal paper
5. Aaron C. Newman, CTO & Founder, 2009, Intrusion Detection and Security Auditing In Oracle White Paper [12], application security inc, Steve's Pick of the Week 2009
6. Andrew C. Herlands, 2007 Arrest the Threat: Monitoring Privileged Database Users, application security inc., 2007
7. Acronymics, Inc., (2006). Agent Builder – An Integrated Toolkit for Constructing Intelligent Software Agents, <http://www.agentbuilder.com> (Visited 2011 4-6-2011)
8. IBM Research, 2002. Aglets, <http://www.trl.ibm.com/aglets/> (Visited 2011 1-4-2011)
9. Tammy Bednar (2010), Oracle Database Auditing: Performance Guidelines, August, 2010
10. 9.Rakesh Agrawal, Peter J. Haas, Jerry Kiernan, Watermarking relational data: framework, algorithms and analysis, The VLDB Journal — The International Journal on Very Large Databases, v.12 n.2, p.157-169, August 2003
11. 10.Acronymics, Inc., 2006, Acronymics, Inc., (2006), Agent Builder – An Integrated Toolkit for Constructing Intelligent Software Agents.
12. Telecom Italia Lab, (2007). JADE – Java Agent DEvelopment Framework, Version 3.5
13. Telecom Italia Group, (2007). Telecom Italia Lab. <http://www.telecomitalia.com/> (Visited 2011 1-2-2011)
14. Agent Oriented Software Pty. Ltd., (2006). JACK(TM) Intelligent Agents Agent Manual, Release 5.2, http://www.agent-software.com/shared/demosNdocs/Agent_Manual_WEB/index.html
15. AOTLab, (2007), The Agent and Object Technology Lab at University of Parma, University of Parma, Italy, <http://aot.ce.unipr.it/>(Visited 2011 2-32011)
16. Bellifemine, F. L., Caire, G., Greenwood, D., (2007). Developing Multi-Agent Systems with Jade. New York: J.Wiley.

Appendix 1

```
package com.msc.project.parser;
import com.msc.project.ui.EtherealTreeNode;
/**
 *
 * @author Boniface Akuku
 */
public class TCPParser extends PacketParser {
    TCPParser() {
        dataLen = 1;
        dataStart = 20;
        typeLen = 2;
        typeStart = 2;
    }

    protected byte[] strip(byte[] packet) {
        int headerLen = ((packet[12] & 0xF0) >> 4) * 4;
        byte[] ret = new byte[packet.length - headerLen];
        System.arraycopy(packet, headerLen, ret, 0, ret.length);
        return ret;
    }

    @Override
    public String getProtocol(byte[] packet) {
        // Look inside
        if (strip(packet).length == 0)
            return "TCP";
        else
            return "TCP (" + Registry.lookup("TCP", getType(packet)).getProtocol(strip(packet)) + ")";
    }

    @Override
    public String getInfo(byte[] packet) {
        String flags = "";
        if ((packet[13] & 0x20) != 0) flags += "URG ";
        if ((packet[13] & 0x10) != 0) flags += "ACK ";
        if ((packet[13] & 0x08) != 0) flags += "PSH ";
        if ((packet[13] & 0x04) != 0) flags += "RST ";
        if ((packet[13] & 0x02) != 0) flags += "SYN ";
        if ((packet[13] & 0x01) != 0) flags += "FIN ";
        if (!flags.equals(""))
            flags = "(TCP " + flags + ") ";
        if (strip(packet).length != 0)
            flags += Registry.lookup("TCP", getType(packet)).getInfo(strip(packet)) + " (Port " +
            getType(packet) + ")";
        return flags;
    }

    @Override
    public EtherealTreeNode getTree(byte[] packet) {
        EtherealTreeNode root = new EtherealTreeNode("TCP Packet (" + packet.length + " bytes)", packet);
        int len = ((packet[12] & 0xF0) >> 2);
        dataStart = len;
        EtherealTreeNode header = new EtherealTreeNode("TCP Packet Header (" + len + " bytes)", 0, len);
        root.add(header);
        int src_port = ((packet[0] & 0xFF) << 8) | ((packet[1] & 0xFF));
        int dst_port = ((packet[2] & 0xFF) << 8) | ((packet[1] & 0xFF));
        long seq_num = ((packet[3] & 0xFF) << 24) | ((packet[4] & 0xFF) << 16) |
            ((packet[5] & 0xFF) << 8) | (packet[6] & 0xFF);
        long ack_num = ((packet[7] & 0xFF) << 24) | ((packet[8] & 0xFF) << 16) |
            ((packet[9] & 0xFF) << 8) | (packet[10] & 0xFF);
        header.add(makeNode("Src Port = " + src_port, 0, 2));
    }
}
```

```

header.add(makeNode("Dst Port = " + dst_port, 2, 2));
header.add(makeNode("Sequence Number = " + seq_num, 4, 4));
header.add(makeNode("Acknowledgement Number = " + ack_num, 8, 4));
header.add(makeNode("Header Size", len, 12, 1));

String ecn = "";
if ((packet[12] & 0x01) != 0) ecn += "Nonce Sum ";
if ((packet[13] & 0x80) != 0) ecn += "CWR ";
if ((packet[13] & 0x40) != 0) ecn += "ECE ";
if ((packet[13] & 0x20) != 0) ecn += "URG ";
if ((packet[13] & 0x10) != 0) ecn += "ACK ";
if ((packet[13] & 0x08) != 0) ecn += "PSH ";
if ((packet[13] & 0x04) != 0) ecn += "RST ";
if ((packet[13] & 0x02) != 0) ecn += "SYN ";
if ((packet[13] & 0x01) != 0) ecn += "FIN ";

header.add(makeNode("Control Bits = " + ecn, 13, 1));
header.add(makeNode("Window", packet, 14, 2));
header.add(makeNode("Checksum = 0x" + hexFormat(packet, 16, 2), 16, 2));
header.add(makeNode("Urgent Pointer", packet, 18, 2));
header.add(makeNode("Options and Padding", 20, len - 20));
if (strip(packet).length != 0)
root.add(nextNode(packet, "TCP", len));
return root;
} }

```

Appendix 2

```

package com.msc.project.agent;
import com.msc.project.core.PersistenceImpl;
import com.msc.project.entity.AuditLog;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAException;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

```

```
/**
```

```
*
```

```
* @author Boniface Akuku
```

```
*/
```

```
public class SQLExtractorAgent extends Agent {
```

```
    private AID exceptionalCaseAgent;
```

```
    @Override
```

```
    protected void setup() {
```

```
        String agentName = null;
```

```
        Object[] args = getArguments();
```

```
        if (args != null && args.length > 0) {
```

```
            agentName = (String) args[0];
```

```

} else {
    // Make the agent terminate immediately
    System.out.println("Error: Agent name not specified!!!!");
    doDelete();
}
/*
 * Create directory facilitator description (DF) for the agent
 * and register this agent & its service type
 */
DFAgentDescription directorFacilatorDescription = new DFAgentDescription();
directorFacilatorDescription.setName(getAID());
ServiceDescription serviceDescription = new ServiceDescription();
serviceDescription.setType("AgentBasedDBAudit");
serviceDescription.setName("SQLExtractorAgent");
serviceDescription.setName(agentName);
directorFacilatorDescription.addServices(serviceDescription);
try {
    DFService.register(this, directorFacilatorDescription);
} catch (FIPAException fe) {
    fe.printStackTrace();
}
//add behaviors
addBehaviour(new ExtractSQL());
}

@Override
protected void takeDown() {
    try {
        DFService.deregister(this);
    } catch (FIPAException fe) {
        fe.printStackTrace();
    }
}

private class ExtractSQL extends CyclicBehaviour {
    @Override
    public void action() {
        MessageTemplate messageTemplate = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
        ACLMessage message = myAgent.receive(messageTemplate);
        if (message != null) {
            try {
                //Filter Packet with SQL Key words and persist
                AuditLog auditLog = (AuditLog) message.getContentObject();
                if (packetContainSQL(auditLog)) {
                    //persist log
                    new PersistenceImpl().persist(auditLog);
                }
                //reply to PacketCapture
                ACLMessage reply = message.createReply();
                reply.setPerformative(ACLMessage.INFORM);
                reply.setContentObject(auditLog);
                myAgent.send(reply);
            } catch (Exception ex) {
                Logger.getLogger(SQLExtractorAgent.class.getName()).log(Level.SEVERE, null, ex);
            }
        } else {
            block();
        }
    }

    public boolean packetContainSQL(AuditLog auditLog) {
        if (auditLog == null) {

```

```

        return Boolean.FALSE;
    }
    String[] sqlKeyWords = new String[]{"select", "insert", "delete", "update", "alter", "create", "drop"};
    List<Boolean> matchStatus = new ArrayList<Boolean>();
    for (String keyWord : sqlKeyWords) {
        Pattern pattern = Pattern.compile(keyWord);
        Matcher matcher = pattern.matcher(auditLog.getMessage().toLowerCase());
        matchStatus.add(matcher.find());
    }
    if (matchStatus.contains(Boolean.TRUE)) {
        System.out.println("=====MESSAGE=====");
        System.out.println(auditLog.getMessage());
        System.out.println("=====MESSAGE=====");
    }
    return matchStatus.contains(Boolean.TRUE);
}
public void locateExceptionalCaseAnalysisAgent() {
    DFAgentDescription template = new DFAgentDescription();
    ServiceDescription sd = new ServiceDescription();
    sd.setType("AgentBasedDBAudit");
    sd.setName("ExceptionalCaseAgent");
    template.addServices(sd);
    try {
        DFAgentDescription[] result = DFService.search(myAgent, template);
        exceptionalCaseAgent = result[0].getName();
    } catch (FIPAException fe) {
        fe.printStackTrace();
    } } }
}

```

Appendix 3

```

package com.msc.project.agent;
import com.msc.project.core.PersistenceImpl;
import com.msc.project.entity.AuditLog;
import com.msc.project.entity.ExceptionalCase;
import com.msc.project.entity.Util;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAException;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

```

```

/**
 *
 * @author Boniface Akuku
 */
public class ExceptionalCaseAgent extends Agent {
    private AID alertNotificationAgent;

```

```
private String alertMessage = "Testing...";
```

```
@Override
```

```
protected void setup() {  
    String agentName = null;  
    Object[] args = getArguments();  
    if (args != null && args.length > 0) {  
        agentName = (String) args[0];  
    } else {  
        // Make the agent terminate immediately  
        System.out.println("Error: Agent name not specified!!!!");  
        doDelete();  
    }  
    /*  
    * Create directory facilitator description (DF) for the agent  
    * and register this agent & its service type  
    */  
    DFAgentDescription directorFacilatorDescription = new DFAgentDescription();  
    directorFacilatorDescription.setName(getAID());  
    ServiceDescription serviceDescription = new ServiceDescription();  
    serviceDescription.setType("AgentBasedDBAudit");  
    serviceDescription.setName("ExceptionalCaseAgent");  
    serviceDescription.setName(agentName);  
    directorFacilatorDescription.addServices(serviceDescription);  
    try {  
        DFService.register(this, directorFacilatorDescription);  
    } catch (FIPAException fe) {  
        fe.printStackTrace();  
    }  
    //add behaviors  
    addBehaviour(new AnalyzeExceptionalCase());  
}
```

```
@Override
```

```
protected void takeDown() {  
    super.takeDown();  
}
```

```
private class AnalyzeExceptionalCase extends CyclicBehaviour {
```

```
@Override
```

```
public void action() {  
    MessageTemplate messageTemplate = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);  
    //Requested by the Referee to play  
    ACLMessage message = myAgent.receive(messageTemplate);  
    if (message != null) {  
        try {  
            AuditLog auditLog = (AuditLog) message.getContentObject();  
            //1) Time when the transaction was performed  
            //No operation is allowed in the database between 2000hrs and 0800hrs  
            if (prohibitedDuration()) {  
                alertMessage = "Violation of database access duration time. Access time occurred between 2000hrs and  
0800hrs. Accessed at :"+new Date().toString()+" !!";  
                //Log exceptional case  
                ExceptionalCase exceptionalCase = new ExceptionalCase();  
                exceptionalCase.setDescription(alertMessage);  
                exceptionalCase.setOccurrenceTime(new Date());  
                new PersistenceImpl().persist(exceptionalCase);  
                //reply to PacketCapture  
                ACLMessage reply = message.createReply();  
            }  
        }  
    }  
}
```

```

reply.setPerformative(ACLMessage.INFORM);
reply.setContent(alertMessage);
myAgent.send(reply);
}
//2) If drop statement is intercepted
//No alter or drop statement to accepted in the db
if (prohibitedSQLOperation(auditLog)) {
    alertMessage = "Prohibited operation at " + new Date().toString()+ " [ "+auditLog.getMessage()+" ]";
    ExceptionalCase exceptionalCase = new ExceptionalCase();
    exceptionalCase.setDescription(alertMessage);
    exceptionalCase.setOccurrenceTime(new Date());
    new PersistenceImpl().persist(exceptionalCase);
//reply to PacketCapture
ACLMessage reply = message.createReply();
reply.setPerformative(ACLMessage.INFORM);
reply.setContent(alertMessage);
myAgent.send(reply);
}

} catch (Exception ex) {
    Logger.getLogger(SQLExtractorAgent.class.getName()).log(Level.SEVERE, null, ex);
}
} else {
    block();
}
}

private boolean exceptionalCase(AuditLog auditLog) {
//To do: Analyze based on dimensions {Rule based/Production rules inference ==> IF...ELSE..}

//1) Time when the transaction was performed
//No operation is allowed in the database between 2000hrs and 0800hrs
if (prohibitedDuration()) {
    alertMessage = "Violation of database access duration time. Access time ocured between 2000hrs and 0800hrs";

    return true;
}

//2) If drop statement is intercepted
//No alter or drop statement to accepted in the db
if (prohibitedSQLOperation(auditLog)) {
    alertMessage = "Execution of prohibited database operation at " + new Date().toString();
    return true;
}

return false;
}

private boolean prohibitedSQLOperation(AuditLog auditLog) {

String[] sqlKeyWords = new String[]{"drop", "alter", "delete"};
List<Boolean> matchStatus = new ArrayList<Boolean>();
for (String keyWord : sqlKeyWords) {
    Pattern pattern = Pattern.compile(keyWord);
    if (auditLog != null) {
        Matcher matcher = pattern.matcher(auditLog.getMessage().toLowerCase());
        matchStatus.add(matcher.find());
    }
}
}

```

```

    }
    return matchStatus.contains(Boolean.TRUE);
}

private boolean prohibitedDuration() {
    Date currentTime = new Date();
    int hours = currentTime.getHours();
    if (hours > 20) {
        return true;
    }
    if (hours < 8) {
        return true;
    }
    return false;
}

public void locateAlertNotificationAgent() {
    DFAgentDescription template = new DFAgentDescription();
    ServiceDescription sd = new ServiceDescription();
    sd.setType("AgentBasedDBAudit");
    sd.setName("AlertNotificationAgent");
    template.addServices(sd);
    try {
        DFAgentDescription[] result = DFService.search(this, template);
        this.alertNotificationAgent = result[0].getName();
    } catch (FIPAException fe) {
        fe.printStackTrace();
    }
}

```

Appendix 4

```
package com.msc.project.agent;
```

```

import com.msc.project.core.PersistenceImpl;
import com.msc.project.email.MailHandler;
import com.msc.project.email.MailSender;
import com.msc.project.entity.Mail;
import com.msc.project.entity.MailConfig;
import com.msc.project.sms.ModemHandler;
import com.msc.project.sms.SMSNotifier;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAException;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
/**
 *
 * @author Boniface Akuku
 */
public class AlertNotificationAgent extends Agent {

```

```
@Override
```

```

protected void setup() {
    String agentName = null;
    Object[] args = getArguments();
    if (args != null && args.length > 0) {
        agentName = (String) args[0];
    } else {

```



```

// Make the agent terminate immediately
System.out.println("Error: Agent name not specified!!!!");
doDelete();
}
/*
 * Create directory facilitator description (DF) for the agent
 * and register this agent & its service type
 */
DFAgentDescription directorFacilatorDescription = new DFAgentDescription();
directorFacilatorDescription.setName(getAID());
ServiceDescription serviceDescription = new ServiceDescription();
serviceDescription.setType("AgentBasedDBAudit");
serviceDescription.setName("AlertNotificationAgent");
serviceDescription.setName(agentName);
directorFacilatorDescription.addServices(serviceDescription);
try {
    DFService.register(this, directorFacilatorDescription);
} catch (FIPAException fe) {
    fe.printStackTrace();
}
PersistenceImpl persistenceImpl = new PersistenceImpl();
MailConfig mailConfig = persistenceImpl.getMailConfig();
//Schedule SMS and Email submission
ModemHandler modemHandler = new ModemHandler(mailConfig.getModemPort(),
mailConfig.getModemPassword(), "Safaricom", "Huawei", "E220", "0.0", 115200);
MailHandler mailHandler = new MailHandler();
//add behaviors
addBehaviour(new SendSMSAndEmail());

//add behaviors
// addBehaviour(new SendEmail());
}
private class SendSMSAndEmail extends CyclicBehaviour {
    @Override
    public void action() {
        MessageTemplate messageTemplate = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
        ACLMessage message = myAgent.receive(messageTemplate);
        if (message != null) {
            System.out.println("=====ALERT NOTIFICATION: SENDING SMS=====");
            PersistenceImpl persistenceImpl = new PersistenceImpl();
            MailConfig mailConfig = persistenceImpl.getMailConfig();
            SMSNotifier smsNotifier = new SMSNotifier();
            smsNotifier.sendSMS("Database Audit Alert !" + message.getContent(), mailConfig.getRecipientNumber(),
mailConfig.getModemNumber(), "Safaricom");
            System.out.println("=====ALERT NOTIFICATION: SENDING EMAIL=====");
            MailSender mailSender = new MailSender(mailConfig.getServerIp(), mailConfig.getUserName(),
mailConfig.getPassword());
            Mail mail = new Mail();
            mail.setMailSubject("Database Audit Alert");
            mail.setMailContent(message.getContent());
            mail.setReceiver(mailConfig.getRecipient());
            persistenceImpl.persist(mail);
            //reply to PacketCapture
            ACLMessage reply = message.createReply();
            reply.setPerformative(ACLMessage.INFORM);
            myAgent.send(reply);
        } else {
            block();
        }
    }
}

```

```

}
private class SendEmail extends CyclicBehaviour {

    @Override
    public void action() {
        MessageTemplate messageTemplate = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
        ACLMessage message = myAgent.receive(messageTemplate);
        if (message != null) {
            System.out.println("=====ALERT NOTIFICATION: SENDING EMAIL=====");
            PersistenceImpl persistenceImpl = new PersistenceImpl();
            MailConfig mailConfig = persistenceImpl.getMailConfig();
            MailSender mailSender = new MailSender(mailConfig.getServerIp(), mailConfig.getUserName(),
mailConfig.getPassword());
            Mail mail = new Mail();
            mail.setMailSubject("Database Audit Alert");
            mail.setMailContent(message.getContent());
            mail.setReceiver(mailConfig.getRecipient());
            persistenceImpl.persist(mail);
            //mailSender.sendMail("Database Audit Alert: ", message.getContent(), mailConfig.getRecipient());

            //reply to PacketCapture
            ACLMessage reply = message.createReply();
            reply.setPerformative(ACLMessage.INFORM);
            reply.setContent("MAIL");
            myAgent.send(reply);
        } else {
            block();
        }
    }
}

private class SendSMS extends CyclicBehaviour {
    @Override
    public void action() {
        MessageTemplate messageTemplate = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
        ACLMessage message = myAgent.receive(messageTemplate);
        if (message != null) {
            System.out.println("=====ALERTNOTIFICATION: SENDING SMS=====");
            PersistenceImpl persistenceImpl = new PersistenceImpl();
            MailConfig mailConfig = persistenceImpl.getMailConfig();
            SMSNotifier smsNotifier = new SMSNotifier();
            smsNotifier.sendSMS("Database Audit Alert :" + message.getContent(), mailConfig.getReceipientNumber(),
mailConfig.getModemNumber(), "Safaricom");
            //reply to PacketCapture
            ACLMessage reply = message.createReply();
            reply.setPerformative(ACLMessage.INFORM);
            myAgent.send(reply);
        } else {
            block();
        }
    }
}

@Override
protected void takeDown() {
    super.takeDown(); }
}

```