

A Formal Model of a Financial Audit System

by

John Akinlabi Akinyemi

A thesis
presented to the University of Manitoba
in partial fulfilment of the
requirements for the degree of
Master of Science
in
Computer Science

Winnipeg, Manitoba, Canada, 2006

©John Akinlabi Akinyemi 2006

THE UNIVERSITY OF MANITOBA
FACULTY OF GRADUATE STUDIES

COPYRIGHT PERMISSION

A Formal Model of a Financial Audit System

BY

John Akinlabi Akinyemi

**A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of
Manitoba in partial fulfillment of the requirement of the degree**

OF

MASTER OF SCIENCE

John Akinlabi Akinyemi © 2006

Permission has been granted to the Library of the University of Manitoba to lend or sell copies of this thesis/practicum, to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film, and to University Microfilms Inc. to publish an abstract of this thesis/practicum.

This reproduction or copy of this thesis has been made available by authority of the copyright owner solely for the purpose of private study and research, and may only be reproduced and copied as permitted by copyright laws or with express written authorization from the copyright owner.

Abstract

This thesis presents a formal model of a financial audit system, as well as an architecture and a classification of concepts and relationships in a financial audit system. The formal model uses the Unified Modeling Language to describe the financial audit system, and Predicate Logic to provide a formal specification for the requirements of the system. Finally, a prototype implementation of a financial audit system specified using Predicate Logic is presented, as it is used to collect financial audit data. This prototype uses a similar method as that used for collecting audit and administrative data in major operating systems.

List of Publications

1. **John A. Akinyemi**, Sylvanus A. Ehikioya, and Bamidele Ola. "Formalizing a Financial Audit System Requirements." *Special Issue on Computer Auditing Journal*. Communications of the ICISA Vol.7, No.2, Fall 2005.
2. **John A. Akinyemi** and Sylvanus A. Ehikioya. "A Formal Specification of a Financial Audit System." In *3rd International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications (CSITeA04)*, Cairo, Egypt. (**Accepted:** December, 2004).
3. **John A. Akinyemi**, Sylvanus A. Ehikioya, Femi G. Olumofin, and Chima Adiele. "An Ontology and Knowledge Representation of a Financial Audit System." In *Proceedings of the IASTED International Conference on Knowledge Sharing and Collaborative Engineering (KSCE 2004)*, St. Thomas, Virgin Islands, USA, November 22-24, 2004, pp. 207-212.
4. **John A. Akinyemi** and Sylvanus A. Ehikioya. "A Predicate Logic Foundation for Financial Audit Systems." In *Proceedings of the 8th IASTED International Conference on Software Engineering and Applications (SEA 2004)*, Cambridge, MA, USA, November 9-11, 2004, pp. 339-344.

Dedication

To the glory of God, for divine provisions of all my physical, emotional, and spiritual needs during this program. Also, for providing me the necessary courage to keep going despite very challenging difficulties.

Acknowledgements

I want to thank my advisors Dr. Sylvanus A. Ehikioya and Dr. Dean Jin for providing guidance and support during the completion of this thesis. I also express my sincere gratitude to Dr. Janet Morrill of the Accounting and Finance Department, University of Manitoba, Dr. Parimala Thulasiraman and Dr. John Anderson of the Computer Science Department, University of Manitoba for their valuable time spent in reading my thesis and being part of my examination committee. Others who provided assistance are Dr. Neil Arnason and Dr. Christel Kemke.

Also, I would like to thank the University of Manitoba, the Computer Science Department and the entire staff of the Computer Science Department for the opportunity and all resources provided for my research.

In addition, I would like to convey my gratitude to all my friends especially Dr. Olanrewaju Ojo and Mr. Ola Bamidele and my pastors: Pastor Tokunbo Okunnu and his family, Pastor (Dr.) Akindele Odeshi and his family and Pastor Gideon Kpotsi Nekou and his family for their support and kindness during this program. May the Lord bless you abundantly, amen. Sincere appreciation goes to the following additional people: David Allenor, Femi Olumofin, Idowu Oduntan, Pastor Ikechukwu Isinguzo, Dr. and Mrs. Moses Owolabi, Christopher Iyogun and his family, Tope Owoeye and Dr. Thomas Imeh

Nathaniel. A big thank you also goes to all my other friends too numerous to mention here.

Special thanks go to my brother and friend, Mr. Akanni Ibukun Akinyemi and his family, you are indeed one in a billion. Thank you for all your support and encouragements even when it became very tough. I acknowledge all the previous contributions by my late mother Late (Mrs.) Elizabeth Olaide Akinyemi. A big thank you to Olusegun Aminu, Late (Mrs.) Margaret Titilayo Aminu and Late (Mrs.) Victoria Oyinlola Alonge (nee Akinyemi).

This thesis would have been practically impossible without the support, understanding, and sacrifices from my dearest sweetheart and lovely wife Elizabeth Olabisi and my precious baby Toluwanimi Shalom. You are so precious to me and I will always remember that you have a major stake in this academic achievement. Thank you very much.

Contents

1	Introduction	1
1.1	Benefits of a Financial Audit System	5
1.2	Problem Statement and Goals	6
1.3	Formal Specifications in the Financial Audit System	7
1.4	Theoretical Foundation of Financial Auditing	7
1.4.1	Management Assertions	8
1.4.2	Audit Objectives	9
1.4.3	Audit Analysis	10
1.4.4	Computer Assisted Audit Approaches	10
1.4.5	Electronic Audit Evidence	11
1.5	Organization of this Thesis	12
1.6	Conclusions	13
2	Auditing Concepts, UML, and Predicate Logic	14
2.1	Auditing Concepts	14

2.1.1	Audit Models and Simulations	15
2.1.2	Audit Requirements and Specifications	15
2.1.3	Audit Approaches in Computer Systems	16
2.1.4	Audit Analysis Software and Tools	17
2.1.5	Focus of this Thesis	18
2.2	Unified Modeling Language	18
2.2.1	Syntax and Semantics of UML Diagrams	20
2.3	Predicate Logic Specification Language	25
2.3.1	Reasoning and Inference Ability of Predicate Logic	26
2.3.2	Syntax of Predicate Logic	27
2.3.3	Semantics of Predicate Logic	29
2.4	Terminology Used in this Thesis	32
2.5	Conclusions	34
3	A Formal Model of a Financial Audit System	36
3.1	Architecture of a Financial Audit System	36
3.2	Classification and Model Description of a Financial Audit System	37
3.3	Requirements of a Financial Audit System	41
3.4	Overview of the Formal Model	42
3.5	UML Model of a Financial Audit System	43
3.5.1	Use Case Diagrams	43

3.5.2	Class Diagrams	47
3.5.3	Activity Diagrams	52
3.5.4	Statechart Diagrams	54
3.5.5	Collaboration Diagrams	57
3.6	Predicate Logic Specifications of a Financial Audit System	69
3.6.1	Informal Definitions of Terms and Symbols	69
3.6.2	Formal Definitions of Terms and Symbols	73
3.7	Requirements of a Financial Audit System and their Specifications with Predicate Logic	92
3.7.1	Accountability Requirements and Specifications	92
3.7.2	Security Requirements and Specifications	95
3.7.3	Transaction Monitoring Requirements and Specifications	98
3.7.4	Event Logging Requirements and Specifications	99
3.7.5	Reporting Requirements and Specifications	101
3.8	Consistency Checking	103
3.9	Conclusions	106
4	Financial Audit System Prototype Implementation	108
4.1	A Strategy for Collecting Financial Audit Data	108
4.2	Implementation Details of the Prototype Application	111
4.2.1	Implementation of Accountability Requirements in Section 3.7.1	111

4.2.2	Implementation of Security Requirements in Section 3.7.2	112
4.2.3	Implementation of Transaction Monitoring Requirements in Section 3.7.3	113
4.2.4	Implementation of Event Logging Requirements in Section 3.7.4 .	114
4.2.5	Implementation of Reporting Requirements in Section 3.7.5 . . .	115
4.3	Relationship of the Prototype Implementation with Formal Specifications in Section 3.7	122
4.4	Conclusions	124
5	Conclusions and Future Work	125
5.1	Conclusions	125
5.2	Summary of Contributions	126
5.3	Limitation of this Thesis	127
5.4	Future Work	127
5.4.1	Knowledge-based financial audit system	127
5.4.2	Formal verification of the dynamic (temporal) characterization of a financial audit system	128
5.5	Conclusions	129

List of Tables

3.1	Variables of a Financial Audit System Predicate Logic Specifications . . .	69
3.2	Predicates: FAS Predicate Logic Specifications	75

List of Figures

2.1	Use Case Diagram Symbols	23
2.2	Class Diagram Symbols	23
2.3	Statechart Diagram Symbols	24
2.4	Collaboration Diagram Symbols	24
3.1	Architecture of a Financial Audit System [AEOA04]	39
3.2	Classification of a Financial Audit System [AEOA04]	40
3.3	Use Case of a Business Transaction in a Financial System	45
3.4	Use Case of Processing a Financial Transaction	45
3.5	Use Case of Auditing a Financial Transaction	46
3.6	Class Diagram of a Financial System	50
3.7	Class Diagram of a Financial System Payment Process	51
3.8	Class Diagram of a Financial Audit System	60
3.9	Activity Diagram of a Financial System	61
3.10	Activity Diagram of a Financial Audit System	62

3.11 Financial Audit System Event Monitoring Statechart Diagram	63
3.12 Financial Audit System Transaction Authorization Statechart Diagram . .	64
3.13 Financial Audit System Payment Confirmation Statechart Diagram	65
3.14 Financial Audit System Data Encryption Statechart Diagram	66
3.15 Financial Audit System Exception Review Statechart Diagram	67
3.16 Collaboration Diagram of a Financial Audit System	68
4.1 Sample Transaction Processing Screen Shot	116
4.2 Payment and Receipt Vouchers for a Sample Financial Transaction	117
4.3 Auditors' Actions Audit Reporting Screenshots	118
4.4 Periodical Audit Reporting	119
4.5 User Access Audit Reporting	120
4.6 User Name Audit Reporting	121

Chapter 1

Introduction

According to the American Accounting Association (AAA) [AAA73], auditing is defined as:

“a systematic process of objectively *obtaining* and *evaluating evidence* regarding assertions about economic actions and events to ascertain the degree of correspondence between those assertions and established criteria and communicating the result to interested users.”

Financial auditing entails collection, analysis, and reporting of financial-related data for the purpose of detecting and preventing errors, exceptions, and fraud in the financial system. Effects of errors, exceptions, and fraud in a financial system range from loss of money, customer dissatisfaction, loss of jobs, to a complete collapse of business. Financial auditing is crucial because it provides a means for ascertaining user accountability, system monitoring, exception detection, and fraud prevention in the financial system. Audit data are captured and stored in an audit log (event log / audit trail), depicting an evidential document for financial transactions. Audit analysis (a process of testing and verifying

a selected sample of financial transactions for correctness, completeness, accuracy, compliance, and reliability [SA66, TH83]) performed on an audit log reveals exceptions and errors.

Financial systems record millions of transactions daily. Though the dollar value is difficult to estimate, Medjahed *et al.* [MBB⁺03] forecast the e-commerce market alone to be valued at US\$ 8.5 trillion by year 2005. Whether electronic or manual, financial transaction is a continuous activity.

The overwhelming volume and value of financial transactions warrants a careful and error-free process of monitoring and auditing. Auditing financial transaction has become complex [Kos04]. The advent and general acceptance of computer and electronic systems make auditing of payment systems more complex than auditing a purely manual system of just cash and cheque transactions. Financial transactions are crucial to individuals and organizations. Therefore, there is need for clients and stakeholders in the financial system to trust the system [Kos04, Mer03, Rez04]. A thorough auditing of financial systems through an accurate review and monitoring of all financial transactions will achieve a good level of correctness [Koc79], and will allow people to have confidence in the reliability and accuracy of financial systems.

According to Koskivaara [Kos04], a lot of big companies such as WorldCom, Xerox, Enron, Parmalat, and Siebel Systems have problems with their financial auditing systems. These problems have permitted audit data manipulation. In the case of Enron, for example, this had very serious consequences that lead to the collapse of the company, loss of thousands of jobs, and economic implications for the energy sector. Recently, the business world has called for a reassessment of the current audit system [Kos04, Mer03]. Dynamism in business communities encourages expansion, growth, mergers, acquisitions, and organizational changes that require an update to audit systems. In some cases, a complete redesign of the audit system is necessary. Failure to proactively re-evaluate the audit

models and incorporate necessary changes into the auditing processes of these organizations / companies may expose them to fraud and other related problems associated with financial transactions.

Processes for manual review and monitoring of financial transactions have been in existence for a long time. Auditors manually crosscheck source documents of transactions (paper vouchers – documentary and evidential records of financial transactions) with transaction journals. However, according to Akinyemi and Ehikioya [AE04], manual and informally designed audit systems suffer from the following problems:

- *Incorrect and incomplete design.* Manual and informally designed systems may suffer from design flaws due to design incompleteness and errors that exist in some aspects of the system. Hence, it becomes difficult to ascertain the validity, reliability, and accuracy of the incorrectly and incompletely designed systems. This flaw may lead to the repudiation of financial transactions [CICA03], whenever there is a dispute.
- *Manual auditing is inadequate to cope with the volume of financial transactions.* The manual audit system is unable to keep pace with the volume of transactions in online transaction processing and electronic commerce transaction environments. Hence, sensitive and potentially fraudulent transactions could pass through the financial system without being properly audited. This problem could lead to monetary loss.
- *Inconsistencies in the documentation of audit data and evidential documents in financial audit systems.* Some evidential documents are either misplaced, defaced, illegible, destroyed, shredded, or mishandled. This problem leads to misleading and wrong inferences. As a result, managerial decisions based on inconsistent audit data are sub-optimal.

- *Non-enforcement of security and control measures due to lack of audit trail.* This problem causes an abuse of the financial system, and gives a chance to hackers and fraudsters to carry out fraudulent and malicious financial transactions unnoticed. In the manual audit system, it is difficult to ascertain who does what, and when.
- *Breach of information confidentiality.* Manual systems can not completely shield confidential information from unauthorized access and use. Manual and informally designed systems can not guarantee the safety of all confidential data in the financial system. Hence, there is a potential for unauthorized access and use of financial data by hackers, fraudsters, and competitors.
- *Difficulty in audit information retrieval, reporting, and analysis.* Manual and informally designed systems usually tolerate 'open issues' (yet to be resolved problems). Due to the sensitivity and enormity of problems associated with financial systems, open issues are too risky to be allowed in a financial audit system. The information and reports in a manual audit system have the tendency to be inconsistent, as well as take too long to generate. These problems encourage the distortion of audit information and reports.
- *Auditing as a dynamic process.* As businesses evolve through growth, acquisitions, mergers, organizational and operational changes, and personnel changes, new transaction types and procedures become necessary. Failure to implement the necessary changes in the audit procedure in a timely and correct fashion can threaten the continued existence of a business.

A careful and error-free process monitoring and transactions auditing system is necessary. The system must ensure correctness and completeness in financial transactions through auditing.

1.1 Benefits of a Financial Audit System

According to the Canadian Institute of Chartered Accountants (CICA) [CICA03], “Electronic exchange of business documents has become commonplace.” Also, electronic commerce and online purchase of products have made selling across countries and continents be easier. Small, medium, and large companies perform trading activities across borders. The available forms of heterogeneous modes of payments, conversion of different currencies, and other intricacies associated with these heterogeneous modes of payments pose a major challenge to the accuracy, reliability, and validity of financial transactions. Therefore, there is a need to audit financial transactions. Audits verify and ascertain the accuracy, reliability, and validity of all financial transactions. Some of the benefits of a financial audit system (FAS) are:

1. Audit systems establish trust among all stakeholders in a financial system. Each stakeholder relies on a good audit process that detects errors and other forms of related exceptions that are associated with financial transactions. The audit system facilitates a notion of trust among stakeholders in a financial system.
2. A good and accurate audit system assists auditors in performing audit functions, as well as increases the productivity of auditors.
3. Auditing is a major tool that can instill accountability, and eliminate fraudulent malpractices. All transactions are verified and ascertained correct or otherwise by the audit system.
4. A good audit process and the audit of auditors and their activities will prevent financial loss.

1.2 Problem Statement and Goals

This thesis is an attempt to research and possibly provide some solutions to some identified problems in financial auditing. First, the thesis will focus on the identification of some requirements of a financial audit system, as well as the provision of a formal specification for the requirements that are identified.

Secondly, this thesis will provide a model for a financial audit system. The design of the financial audit system model will be based on formal specifications. Finally, the thesis will provide an implementation of a subset of the design and model of the financial audit system. This prototype implementation will demonstrate the practicability of implementing a financial audit system based on formal specification of financial audit system requirements. The implementation provides an opportunity to explore data capture for the financial audit system.

The goals of this thesis include a provision of an easy to understand visual descriptive model of a financial audit system. An audit system is the entire system that obtains and evaluates audit data, supporting all aspects of the auditing process. Also, this thesis will provide formal logic-based specifications and a design of the requirements of a financial audit system. Finally, this thesis will provide an implementation of a financial audit data collection (logging – an audit logging system is the part of an audit system that collects and stores audit data) system that is based on the formal specifications of the requirements of a financial audit system.

The thesis achieves these goals by providing a visual and descriptive model of a FAS with an *architecture* of a FAS, a *classification* of entities / objects in the FAS, as well as *associations* and *relationships* among entities / objects in a FAS. Also, this thesis provides a description of some requirements that are necessary for financial transaction audit work, as well as a formal logic based specification for these requirements. This thesis achieves

the final goal by providing a prototype implementation of a financial audit data logging technique that is based on the formal logic specifications.

1.3 Formal Specifications in the Financial Audit System

Due to the complexities associated with auditing online financial systems, audit processes become highly technical and require knowledgeable people to perform audit functions. The complexities in the audit processes partly stem from an ambiguous and unclear understanding of concepts, relationships, and communications that exist among financial audit system entities. Ambiguity in the meaning of concepts and the relationships that exist among them can be eliminated with the use of formal methods. Formal methods rely on mathematical notations to provide precision in the design of systems.

A number of benefits can be realized through the use of an approach based on formal specifications for modeling software systems:

1. It provides an easy, clear, and understandable abstract representation of the system.
2. It eliminates ambiguities in the design of software systems.
3. It eliminates design incompleteness and incorrectness.
4. It eliminates contradictions in the system design.
5. It provides a verifiable design of a system.

1.4 Theoretical Foundation of Financial Auditing

Auditing entails the collection and analysis of audit evidence to ascertain conformance with established criteria and communicating the result to interested users [AAA73, TH83].

Skinner and Anderson [SA66] describe the two components of financial auditing: *balance sheet audits* and *current audits* (transaction-related audit). A *balance sheet audit* is the year-end verification of asset and liability balances in the financial statements of companies. A *current audit* (also referred to as a day-to-day audit of transactions) is the verification of the correctness of individual transactions recorded daily in the financial system [RS84].

Arens *et al.* [ALLS00] describe the objectives of an audit based on whether a conclusion is being expressed on a balance sheet item or a transaction. The *transaction-related audit objectives* are required for transactions audit, while the *balance-related audit objectives* are required for balance sheet audit. Although the two audit objectives are closely related, they are somewhat different.

1.4.1 Management Assertions

Audit work evaluates audit evidences regarding assertions to ascertain conformance with established criteria [AAA73, TH83]. According to Arens *et al.* [ALLS00], assertions describe “implied or expressed representations by management about classes of transactions and related accounts in the financial statements.” Management provides assertions for each class of accounts in both the transaction and account balances.

Paragraph 5300.17 of the Canadian Institute of Chartered Accountants (CICA) [CICA03] *CICA Handbook* classifies management assertions into seven categories. These categories are: *existence* (of assets and liabilities), *occurrence* (of revenue and income), *completeness* (inclusion of all relevant transactions), *valuation* (of assets and liabilities), *measurement* (of revenues and expenses), *ownership* (of assets rights and liability obligations), and *statement presentation* (relating to disclosures in the classification and description of items in the financial statement).

1.4.2 Audit Objectives

Primarily, auditing determines whether management assertions about financial statements are justified by accomplishing certain established objectives [ALLS00]. These audit objectives provide a guideline to assist auditors in collecting adequate audit evidence for audit work. The *transaction-related audit* has the following objectives: *occurrence* (whether recorded transactions actually occurred), *completeness* (whether all relevant transactions were recorded), *accuracy* (whether correct amounts of recorded transactions were recorded and stated), *classification* (whether transactions included in client's records were properly classified), *timing* (whether transactions were recorded on the correct dates), and *posting and summarization* (whether recorded transactions were updated to the master files and correctly summarized).

Likewise, objectives of a *balance-related audit* include: *existence* (whether all amounts included in the financial statement actually exist), *completeness* (whether all amounts that should be included have been included), *valuation* (whether assets were included at their realizable values), *accuracy* (whether amounts included were stated at correct amounts), *classification* (whether amounts included in client's listing were properly classified), *cut-off* (whether transactions near the balance sheet date were recorded in the proper period), *detail tie-in* (whether transaction details sum to the master files amounts and subsidiary records agree with the total in the account balance in the general ledger), *rights and obligations* (whether assets were owned, and whether liabilities were obligations), and *presentation and disclosure* (whether account balances and related disclosure requirements were properly presented in the financial statements).

It is essential for auditors to gather adequate and appropriate audit evidences to support management assertions in the financial statement before commencing the audit work. Arens *et al.* [ALLS00] describe the four phases of an audit work. These phases include: the planning and designing of an audit approach, performance of tests of controls, per-

formance of analytical procedures and tests of details of balances, and completion of the audit and issuance of auditor's report.

1.4.3 Audit Analysis

According to Ricketts and Sorkin [RS84], analytical review methods consist of ratio analysis, trend analysis, simple linear regression tests, simple comparisons, common-size statements, time series analysis, and financial modeling. Likewise, quantitative approaches include price-level-adjusted time series analysis, economic order quantity models, sensitivity analysis, simulations, present value analysis, and quantitative risk assessment models.

According to Lemon *et al.* [LAL87], techniques of analyzing transaction audit data include: analytical review procedures, tests of transactions, review of transactions with affiliates and interplant accounts, analysis of account balances, direct testing of balance sheet accounts, and tests of allocations. Audit analysis of current transactions is crucial because it largely influences the correctness and completeness of balance sheet audit analysis. As a consequence, collecting correct and complete audit data is very important in ensuring the validity of both transaction and balance sheet audits.

1.4.4 Computer Assisted Audit Approaches

Audit systems are either computer-based or manual. Computer-based audit systems are either continuous or periodic. A continuous audit system can provide real-time audit information because it has access to all relevant data on a real-time basis, whereas a periodic audit system can be achieved on a regular time interval only because the audit system does not have access to relevant data on a real-time basis. A manual audit system is only periodic. The Canadian Institute of Chartered Accountants (CICA) [CICA03]

details problems associated with using computer-based audit data collection mechanisms as audit evidential documents. Some of these problems are: difficulty in establishing proof of origin of electronic data, difficulty in detecting alterations of electronic data, difficulty of establishing authorized information, availability and accessibility of audit data, and difficulty in issuing appropriate and reliable electronic signatures. Despite all the challenges associated with computer-based and electronic evidence collection for audit purposes, it has become a widely used technique for electronic transactions in online, e-commerce, and mobile (m-commerce) financial transactions.

1.4.5 Electronic Audit Evidence

According to CICA [CICA03], electronic audit evidence may take various forms such as text, image, audio or video. They define electronic audit evidence as the “information created, transmitted, processed, recorded, and / or maintained electronically that supports the content of an audit report.” Accounting records, such as electronic invoices and receipts are forms of electronic audit evidence.

Guidelines for security, application, and general control measures that are relevant for electronic audit evidence are also outlined by CICA [CICA03]. These control measures include: (i.) segregation of incompatible duties and access controls, (ii.) retention, archiving, accessibility and destruction of electronic documents and data, (iii.) encryption, electronic signatures and digital certificates, (iv.) management and audit trails, (v.) controls relating to information authentication, (vi.) controls relating to information integrity, (vii.) non repudiation controls, (viii.) information authorization controls, (ix.) data availability controls, and (x.) information confidentiality controls. Also, since audit evidence obtained directly from the source by auditors are more reliable than evidence obtained from third parties [CICA03], it is imperative for auditors to have an independent means for collecting audit data.

The model in this thesis provides a means to assist auditors by programming criteria relating to audit objectives, so that each transaction can be analyzed against the criteria. Also, the selection of audit data is based on the criteria that satisfies audit objectives before they are analyzed by auditors.

The financial audit system in this thesis can provide data to auditors in order to perform certain analyses as described in Section 1.4.3. Finally, this model can screen for control objectives required and is suitable for providing evidences for financial audit system and electronic transactions.

1.5 Organization of this Thesis

This thesis is significant for the following reasons: it provides an easy to understand visual description and model of a financial audit system; it identifies and provides a formal specification for the requirements of a financial audit system; it provides a design of the financial audit system based on the formal specification, and it implements a subset of the design to demonstrate the practicability of this concept.

The remainder of this thesis is organized as follows. Chapter 2 presents auditing concepts related to this thesis work and background information on both the modeling language (UML) and the formal specification language (Predicate logic) used in this thesis. Chapter 3 presents a formal model of a financial audit system, as well as a description of a financial audit system with an architecture, a classification, and some requirements of a financial audit system. Chapter 4 describes a prototype implementation of the financial audit system. Chapter 5 concludes the thesis, and describes future work.

1.6 Conclusions

This chapter introduced the concept of financial auditing in general. The need for good financial audit systems, and problems that arise through the use of manually and informally designed financial audit systems were outlined. The need for formal specifications for financial audit systems was presented. A theoretical foundation for financial auditing was provided detailing the objectives of audit work and the significance of management assertions. Finally, electronic audit evidence was described as it relates to financial auditing.

The next chapter presents auditing concepts that are related to this thesis work and background information on the modeling and formal specification languages that are used in this thesis.

Chapter 2

Auditing Concepts, UML, and Predicate Logic

This chapter presents auditing concepts; structures and descriptions of UML modeling language and the syntax and semantics of Predicate logic formal language as they apply to this thesis. These auditing concepts are based on related work in the areas of both financial and computer systems auditing.

2.1 Auditing Concepts

In this section, the audit concepts that are related to this thesis are outlined. They are grouped into the following categories: audit models and simulations, audit requirements and specifications, audit approaches in computer systems, and audit analysis software and tools.

2.1.1 Audit Models and Simulations

Rezaee *et al.* [RES01] propose a system of continuous auditing; a system that entails selecting, monitoring, and analyzing electronic financial transaction data. Their model uses Extensible Business Reporting Language (XBRL), a standardized electronic language for business reporting. The continuous auditing model describes a system that continuously prepares, publishes, extracts and examines financial information for auditing systems.

Similarly, Yu *et al.* [YYC00] provide two auditing process models for evidence collection and validation in an electronic commerce auditing context. The models are the Periodical Auditing Process Model (PAPM) and the Continuous Auditing Process Model (CAPM). The PAPM facilitates periodic auditing of electronic transactions either annually or semi-annually. CAPM is a real-time transaction monitoring system that can detect exceptions and notify auditors of the occurrence of an exception.

Rezaee *et al.* [RES01] focus on audit data analysis, while Yu *et al.* [YYC00] emphasize detection and notification of exceptions in e-commerce data to auditors. This thesis concentrates on the collection of financial audit data, not just in the e-commerce system context. Nevertheless, I use their idea of online and continuous methods in [RES01] and [YYC00] to facilitate the collection of financial audit data.

2.1.2 Audit Requirements and Specifications

OpenGroup [TOG98] developed the Distributed Audit Service (XDAS) that defines requirements and specifications for generic and online security audit services. The XDAS is an elaborate research effort that defines and models a generic audit system, segregating audit functions into global and local levels. XDAS uses several application programming interfaces (APIs) to define abstract specifications of how to extract relevant audit criteria from applications. XDAS allows an analysis application to configure event pre-selection

criteria. The application then analyzes the criteria before triggering applicable actions.

XDAS provides specifications for some operations that are generic, but relevant for security audits; for example, XDAS provides specifications for system sign-on, and initiation and termination of communication sessions between systems and components in a security audit framework. Furthermore, XDAS specifications define APIs for several security audit related operations, such as an API to submit events to XDAS by applications, an API to read records from an XDAS audit trail, and an API to configure event pre-selection criteria for event submission to XDAS, and so on.

The research work by OpenGroup [TOG98] is focused on providing specifications for a generic audit system, based on APIs that are structured in a programming language format. However, this thesis focuses specifically on financial transactions audit systems. The specifications in this thesis are based on logic formalisms, and are not suited for a particular class of programming languages. My logic-based specifications are suited for any type of programming language, rather than the programming language based APIs in XDAS.

2.1.3 Audit Approaches in Computer Systems

Operating systems have built-in logging systems for security and diagnostic purposes. Examples of computer audit data collection systems built into operating systems are Syslog in UNIX®, and Event Viewer in Microsoft® Windows operating systems. Syslog is a central system message logging facility standard on all modern UNIX® systems. The Event Viewer in Microsoft® Windows operating system logs system, application, and security events that are related to the operating system and some of its applications.

The logging system in an operating system is system based, and it is designed for collecting data that are required for system-related activities. Similarly, the logging system

in operating systems is dependent on the designers of the operating system. The logging system in this thesis is designed to be incorporated into financial applications without any additional input from the designers of operating systems. Logging modules in an operating system track system-based events, while the logging system in this thesis tracks relevant application data in the financial auditing systems, solely for financial audit purposes. Nevertheless, the work in this thesis uses a data collection technique that is similar to the techniques used in Syslog and Event Viewer to collect application-based financial audit data.

2.1.4 Audit Analysis Software and Tools

Several commercially available audit analysis tools and software applications exist, and they are used by financial auditors. A notable audit data analysis software tool is Audit Command Language (ACL) [Wil83]. Structured Query Language (SQL) [DD97] is a query language for generating audit information from databases. Although the focus of this thesis is not on audit data analysis, it is worthwhile to mention that SQL and ACL approaches for analyzing audit data use a similar method to query the database of financial audit data and retrieve audit information from the prototype implementation of the financial audit system in this thesis.

Dalal [Dal05] presents the principle of nanoscience (a microscopic analysis) to extract trends and patterns from audit data. This approach was used to detect a currency counterfeiting fraud. It is based on extracting implicit patterns and trends from a database of financial transactions by conducting a “data-churning exercise” (application of specialized analysis) that identifies and detects anomalies. According to Dalal [Dal05], the auditor used queries (data sort, duplicates, gaps, and data filter) on audit data to detect anomalies and subsequently prevent further fraudulent activities.

In this thesis, a method similar to [Dal05] is used to extract trends and patterns from an

audit log in the prototype. However, the goal here is different from [Dal05]. Dalal [Dal05] used their method to detect anomalies. The prototype implementation in this thesis uses the same method to verify whether the process of logging audit data satisfies the requirements as well as the formal specifications of a financial audit system.

2.1.5 Focus of this Thesis

Some of the previous literature [Koc79, TOG98, YYC00] present models of audit systems. OpenGroup [TOG98] provides a means for specifying some requirements for a generic audit system. Syslog and Microsoft® Windows Event Viewer are suitable for achieving their auditing purposes in their respective operating systems. None of the previous related work provides a means for modeling and formally specifying a financial transaction audit system based on the Unified Modeling Language (UML) [Oes02, RJB99] and symbolic formal logic. This thesis provides a financial audit system model using UML and Predicate Logic [HR02, vEK76]. It also demonstrates a financial audit data collection technique that is based on approaches used in operating systems for collecting data for system audit and administration purposes.

2.2 Unified Modeling Language

According to Rumbaugh *et al.*, UML is:

“a general-purpose visual modeling language that is used to specify, visualize, construct, and document the artifacts of a software system.” [RJB99]

UML uses notations to capture, and provide descriptions and characteristics of a system. This characterization assists in the software development life cycle of the system. UML

supports modeling all the software design and development stages by providing static, dynamic, and organizational / implementation views of both simple and complex software artifacts. The benefits of using UML to model software artifacts [RJB99] include:

1. Presentation of a visual, and easy to understand model of the software system.
2. Domain independence.
3. The provision of a unified notation for model constructs from the requirements specifications to the deployment of the software, and
4. The semantic connotations of UML constructs.

UML models discrete systems, such as software systems, using views and diagrams. A view is a subset of a UML model of a system. It describes just one aspect of the system, employing one or two kinds of diagrams to provide visual descriptions of the concepts in each view. UML views include the static, use case, implementation, deployment, state machine, activity, interaction, and model management views. UML diagrams include the Class, Use Case, Component, Deployment, Statechart, Activity, Sequence, and Collaboration diagrams.

The static behavioural view of UML provides a structure for entities in a model. Software entities and their characteristics are clearly identified through relationships, associations (aggregation, composition, links, and bidirectionality), generalizations, inheritance, classification, dependencies, and constraints in the model. Static views of UML consist of the Use Case and Class diagrams. UML also has dynamic behavioural views. These dynamic views provide temporal and evolving characteristics of entities, showing behaviours when specific actions act on them. Transitions and communications among cooperative objects in a model can be shown. The dynamic view consists of Object, Statechart, Activity, Sequence, and Collaboration diagrams. The organizational view of UML

models software components into smaller pieces of packages that establish a grouping of run-time software entities into components. The organizational views are the Component and Deployment diagrams.

2.2.1 Syntax and Semantics of UML Diagrams

This section presents the syntax and semantics of UML diagrams that are used in this thesis. These diagrams are the Use Case, Class, Activity, Statechart, and Collaboration diagrams.

Use Case Diagrams

Use case diagrams are used to describe high-level entities and processes on systems. These entities are referred to as *actors*, while the processes are referred to as *use cases*. A Use Case diagram shows the use case view of a system as seen by users that interact with the system. Figure 2.1 shows symbols that are used for drawing use case diagrams. The *communication association* represents the flow of messages and interactions between *actors* and *use cases*, while the *system boundary* provides a boundary for the system.

Class Diagrams

A class diagram is a pictorial representation of the generalizations, inheritance, associations, dependencies, relationships, interfaces, and collaborations of classes in a system. Figure 2.2 shows symbols that are used for drawing class diagrams.

A *class* is an entity of a system that has attributes and can perform certain operations (methods). *Generalization*, also referred to as *inheritance* describes the “is-a” relationship between a parent class and a child class. For example, considering a case of two

classes *person* and *student*, we say “*student is-a person*” denotes *student* class inherits the properties of *person* class. *Aggregation* describes the *consists-of* relationship, i.e., a whole / part relationship where the *whole* contains the *part*, whereas a *part* can not contain the *whole*. *Composition* is a form of *aggregation* that is strictly bound, for example, there is a composition relationship between a computer and its processing unit, whereas there is only an aggregation relationship between a computer and a floppy drive. A computer without a processing unit ceases to be a computer, whereas a computer without a floppy drive remains a computer. *Dependency* shows a relationship between two entities in which one entity depends on the other entity. A *link* depicts an association between two entities, and a *binary association* describes an association between exactly two classes.

Statechart Diagrams

Statechart diagrams are used to model dynamic behavioural views of a system. Statechart diagrams capture and provide isolated views of an object showing the initial state, its reactive actions to events, and transitions to a new state. Statecharts are UML representation of a State Machine. Figure 2.3 shows a list of symbols that are used to draw Statechart diagrams.

A *state* (the *state* of an object) depicts the characteristic condition (properties or values) of object attributes whenever it satisfies some conditions, performs some activities, or waits for some events. The *initial state* indicates the beginning of the State Machine, while the *final state* indicates the end.

The *decision* box allows the construction of the “*if-then-else*” language construct. A *transition* is a directed (ordered) arrow that indicates an entity and its transition from one state to the other. A *fork* transition represents a complex transition in which one source state is replaced by two or more target states, resulting in an increase in the number of active states. A *join* transition in a state machine shows which two or more states combine to yield one resulting state.

Activity Diagrams

Symbols that are used in the Activity diagram are similar to the ones used for constructing Statechart diagrams. However, a *transition* in Statecharts is referred to as *control flow* in Activity diagrams.

Collaboration Diagrams

Collaboration diagrams model the dynamic interaction of objects and their associations with other objects in a system. A Collaboration diagram uses links to represent the associations between objects, as well as ordered numbers to specify the sequence of messages between all the system objects. Figure 2.4 presents a list of symbols that are used for drawing Collaboration diagrams.

A *classifier role* is a slot that describes the role played by a participating entity in a collaboration. An *association role* connects two classifier roles within a collaboration, as well as represents an association between two classifiers in a collaborating system.

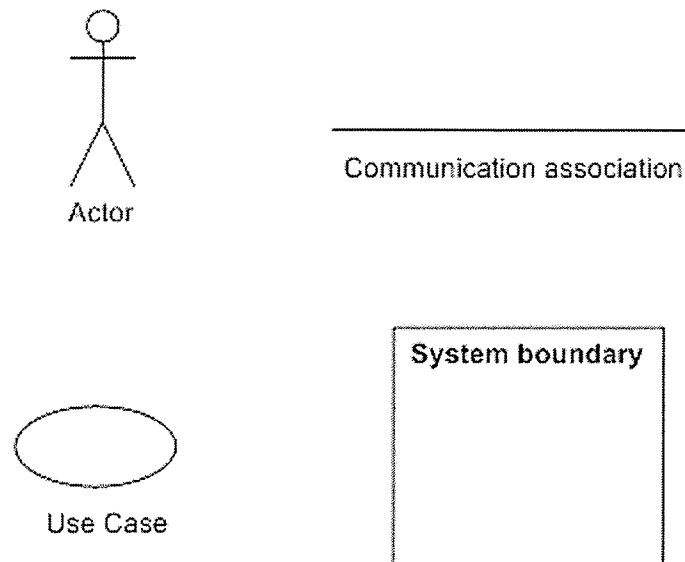


Figure 2.1: Use Case Diagram Symbols

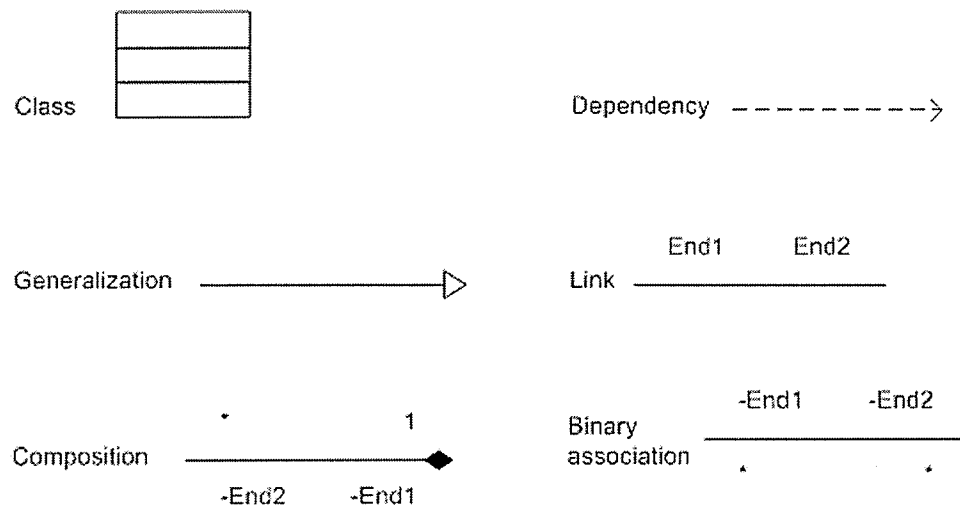


Figure 2.2: Class Diagram Symbols

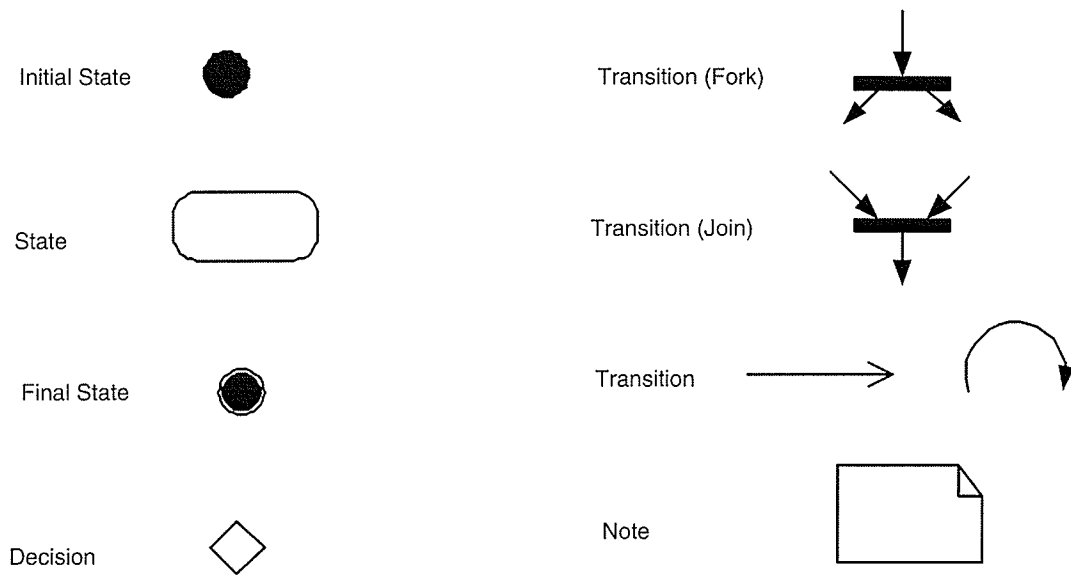


Figure 2.3: Statechart Diagram Symbols

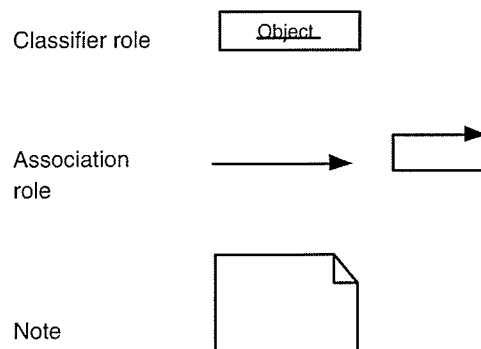


Figure 2.4: Collaboration Diagram Symbols

2.3 Predicate Logic Specification Language

Predicate Logic is a specification language that uses predicates, functions, variables, constants, connectives, quantifiers, terms, and sentences to represent knowledge. The syntax, semantics and inference rules of predicate logic is based on a well-formed formal language construct.

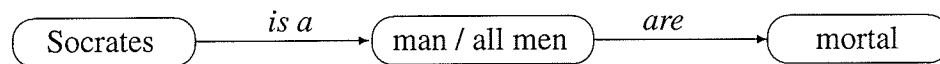
Definitions:

- A *predicate* is a function whose co-domain is the set of Boolean logical constants $\{TRUE, FALSE\}$.
- A *function* is a rule for deriving a value, say v , from another value, say w . Value w is called the argument and value v is the corresponding result.
- A *variable* is a symbol or name that represents a value.
- A *constant* is a value that does not change.
- *Connectives* are logical operators that connect atomic statements into more complex statements. These connectives are described in Section 2.3.2.
- *Quantifiers* are operators that specify for which values of a variable a formula is true. The universal quantifier (\forall) means “for all values”, existential quantifier (\exists) means “there exists some value”, and the unique existential quantifier (\exists_1) means “there exists a unique (one and only one) value”.
- A *term* is a part of speech representing something, but which is not true or false in its own right, for example “man”.
- *Atomic formula* or *sentence* is a predicate name followed by a list of variables such as $P(x, y)$, where P is a predicate name, and x and y are variables.

- *Predicate logic sentences* are built up from atomic sentences. An *atomic sentence* is a predicate name followed by a list of arguments / terms. *Complex sentences* use connectives and quantifiers on atomic sentences.
- A *premise* refers to a formula or sentence that is considered *TRUE* in the domain of discourse.
- An *inference* or *conclusion* is a fact that is deduced from certain premises.

2.3.1 Reasoning and Inference Ability of Predicate Logic

Predicate logic aids reasoning about propositional connectives and quantifications. Consider an example from Luger and Stubblefield [LS98]:



In this example, we can assert two premises, namely:

Premise 1: *All men are mortal.*

Premise 2: *Socrates is a man.*

From Premises 1 and 2, we can infer that *Socrates is mortal*. Predicate logic has the ability to represent complex sentences (facts) in a domain of discourse, as well as infer / derive new facts from previous ones (premises). Predicate logic guarantees the validity of inference(s) / conclusion(s) that are based on previously established facts. In this case, the facts are the two premises. The reasoning capability of Predicate logic deduces a connection between Premise 1 and Premise 2. In this way, Predicate logic implicitly derives a new rule that is based on the two previous premises, which is *man belongs to all men*.

Informally, this inference is derived through a sequence of different substitutions as shown below:

1. All men are mortal – Premise 1
2. Socrates is a man – Premise 2
3. Man belongs to all men – Derived from 1 and 2
4. Socrates belongs to all men – Derived from 2 and 3
5. Socrates is mortal - Derived from 1 and 4

2.3.2 Syntax of Predicate Logic

The syntax of Predicate logic consists of logical symbols, variables, constants, and other symbols. The following tables describe the syntax of Predicate logic language used in this thesis.

Logical Symbols:

<i>Symbol</i>	<i>Meaning</i>	<i>Usage</i>
\neg	negation (“NOT”)	unary connective
\wedge	conjunction (“AND”)	binary connective
\vee	disjunction (“OR”)	binary connective
\Rightarrow	implication (“IMPLIES”)	binary connective
\Leftarrow	consequence (“FOLLOWS FROM”)	binary connective
\Leftrightarrow	equivalence (\equiv)	binary connective
\forall	universal quantifier (“FOR ALL”)	quantifier
\exists	existential quantifier (“THERE EXISTS”)	quantifier

Variables and Constants:

<i>Symbol</i>	<i>Meaning</i>	<i>Usage</i>
x, y, z, \dots	variable	variable
<i>Socrates, a, \dots</i>	constant	constant

Other Symbols:

<i>Symbol</i>	<i>Meaning</i>	<i>Usage</i>
f, g, h	symbols	Function symbols
$X, Y, MORTAL$	symbols	Predicate symbols
,	comma	used to separate symbols
:	colon	such that symbol
()	parenthesis	brackets symbol
=	equals	equality symbol

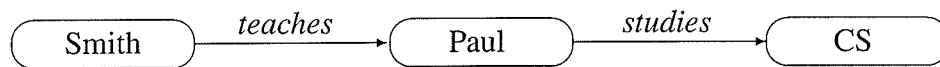
A combination of logical alphabets, variables and constants, and other symbols produces well-formed formulae (wff), which are constructed using the following rules:

1. *TRUE* and *FALSE* are wffs.
2. Each propositional constant (i.e., specific proposition), and each propositional variable (i.e., a variable representing propositions) are wffs.
3. Each atomic formula (i.e., a specific predicate with variables) is a wff.
4. If A , B , and C are wffs, then so are $\neg A$, $(A \wedge B)$, $(A \vee B)$, $(A \Rightarrow B)$, and $(A \Leftrightarrow B)$.
5. If x is a variable (representing objects of the universe of discourse), and A is a wff, then so are $\forall x A$ and $\exists x A$.

2.3.3 Semantics of Predicate Logic

Semantics of Predicate logic formulae is derived through an “*Interpretation*” of the formulae. *Interpretation* assigns symbols of the Predicate logic language to entities of the domain being specified. The *Interpretation* specifies enough premises that can be used to make an inference whether a Predicate logic formulae is *TRUE* or *FALSE*. The *Interpretation* uses an *interpretation function* which does a mapping of the Predicate logic language to entities of the domain. An *Interpretation function* determines the semantics of Predicate logic formulae.

An Example:



In this example:

1. Paul is a student that *studies* Computer Science (CS).
2. Smith is a professor that *teaches* Paul.

Let:

- D represent a non-empty domain (universe of discourse),
- c represent the set of constants in the domain,
- v represent the set of variables in the domain,
- f represent the set of functions in the domain,

- P represent the set of Predicates in the domain, and
- I represent the Interpretation function in the domain.

Formally, the Interpretation (I) is defined thus [Fro86]:

- $I(c) \in D$
- $I(v) \subseteq D$
- $I(f) \subseteq D^n \rightarrow D$ (for an n -ary function)
- $I(P) \subseteq D^n$ (for an n -ary predicate)

The Predicate logic language has the following definitions:

- *Predicates*: Teaches and Studies.
- *Variables*: x, y, z
- *Functions*: student-of, professor-of
- *Constants*: Paul, Smith, CS

Also, the structure of the domain is detailed below:

- *Domain objects*: Paul Williams, Dr. Ian Smith, and Computer Science.
- *Relations*: student, professor.

Interpretation of the Example

1. Interpretation of the Predicate Logic *constants*:

- $I(\text{Paul}) = \text{Paul Williams}$
- $I(\text{Smith}) = \text{Dr. Ian Smith}$
- $I(\text{CS}) = \text{Computer Science}$

2. Interpretation of the Predicate Logic *functions*:

- $I(\text{student-of}) \subseteq \text{student-of}(\text{Smith}) \rightarrow \text{Paul}$
- $I(\text{professor-of}) \subseteq \text{professor-of}(\text{Paul}) \rightarrow \text{Smith}$

3. Interpretation of the Predicate Logic *predicates*:

- $I(\text{Teaches}) \subseteq \text{professor}$
- $I(\text{Studies}) \subseteq \text{student}$

Variables of a Predicate logic language use the concept of *Valuation* [Fro86] to assign values to variables. According to Frost [Fro86], a valuation is a value assignment function which assigns entities of a relational structure to variables of the associated language. These values are domain objects. If v denotes the valuation / value assignment of a Predicate logic language, then the valuation definitions below are valid.

- $v(x) = \text{Paul Williams}$
- $v(y) = \text{Dr. Ian Smith, and}$
- $v(z) = \text{Computer Science}$

2.4 Terminology Used in this Thesis

Formalizing the specifications of a system requires a precise, and unambiguous description of the terms that are used in the specifications. However, for the sake of clarity, formal methods require some informal (verbose) definitions of some of the terms used in the formal specifications. This thesis uses a lot of terms, and these terms are formally defined in Chapter 3. This section presents some verbose descriptions and definitions of terms and concepts that are used to capture the formal specifications in the system. These descriptions are presented below:

- An *audit report* is a report that provides audit information.
- An *auditor* is a person that performs audit functions.
- A *customer* is a person that initiates, instructs, or receives payments through a financial system.
- A *processor* is a person that is authorized by a financial system to process financial transactions.
- An *authorizer* is a person that authorizes financial transactions, processed by processors.
- A *payer* is an agent that requests a financial transaction.
- A *receiver* is an agent that ultimately receives proceeds of a financial transaction.
- An *initiator* is an agent that gives instruction to financial systems to process a financial transaction.
- A *voucher* is a documentary evidence of a financial transaction.

- A *beneficiary* is a receiver of funds from a financial transaction.
- A *transaction* is a transfer of funds from a payer to a receiver through a financial system.
- *High-valued transaction* is a transaction with a pre-determined high monetary value.
- *Transaction detail* is the content and description of a financial transaction.
- *Audit log* is an electronically stored documentary evidence and transaction detail of financial transactions.
- *Audit trigger* is a financial transaction pre-condition that raises a warning flag that alerts for an immediate action.
- A *user* is a person that is authorized to perform assigned functions in a financial system.
- An *audit log user* is a user that performs audit functions.
- *Duty* is a specific function that is assigned to a financial system user.
- *Idle user* is a user that leaves the financial system software unattended for a certain pre-defined period.
- *Transaction status* is the state of a financial transaction, such as complete, incomplete, failed, or successful transactions.
- *Transaction monitoring* is the process of logging transaction details, user information, user actions, user activities, and the status of financial transactions.
- *Activity* is the combination of all operations of a user in the financial system.
- *Action* is a specific unit of user activity in the financial system.

- *Transaction activity* is the user activity in relation to a transaction.
- *Data* is financial data, or a unit of record in the financial system.
- *Huge volume data* is a large-sized data.
- *Encryption utility* is a subsystem in the financial audit system that encrypts audit data.
- *Decryption utility* is a subsystem in the financial audit system that decrypts audit data.
- *User identification* is a piece of information / data that uniquely identifies financial system users.
- *User password* is a secret sequence of characters used to authenticate users in a financial system.
- *Sign-on attempts* is a users' effort to gain access into the financial system.
- *Successful sign-on attempt* is a user sign-on attempt that is successful.
- *Failed sign-on attempt* is a user sign-on attempt that fails.
- *Complete transaction* is a transaction that is successfully concluded.
- *Incomplete transaction* is a transaction that either fails / aborts or is suspended / stopped.

2.5 Conclusions

This chapter presented auditing concepts and research directly related to the thesis in the areas of audit models and simulations, audit requirements and specifications, audit

approaches in Computer systems, and audit analysis software and tools. The tools (UML and Predicate logic) that are used for formal specifications in the thesis were described. This description covered the syntax and semantics of both UML and Predicate logic. Verbose definitions of the terms used to formalize the specifications in the thesis were outlined.

In the next chapter, I present a formal model of a financial audit system. I use the UML modeling language to provide a visual model of a financial audit system. I also provide the requirements of a financial audit system as well as formal specifications of these requirements with Predicate logic specifications language.

Chapter 3

A Formal Model of a Financial Audit System

This chapter presents a formal model of a financial audit system. This formal model is based on UML and Predicate logic formal specifications language. However, the chapter begins by providing a description of a financial audit system that is based on an architecture, a classification, and the requirements of a financial audit system.

3.1 Architecture of a Financial Audit System

Figure 3.1 shows an architecture of the financial audit system that is described in this thesis. The architecture shows three main components: (i.) *financial transaction customers*, (ii.) *financial system*, and (iii.) *financial audit system*. These components interact cooperatively to facilitate financial transactions. The goal of a financial transaction is the transfer of funds from one entity (customers of a financial system) to another. A customer initiates a transaction, and the financial system implements the actual transfer from a *payer* to a

receiver. The financial system consists of a *transaction processing system*, a *transaction verification* mechanism, a *transaction execution* mechanism, and a *transaction database*. An *audit log* is an important part of the transaction database.

The financial audit system consists of subsystems that implement *event monitoring*, *transaction auditing*, and *audit information retrieval*. It is important that the transaction auditing subsystem interacts with the audit log in the transaction database. The *transaction auditing* aspect of the architecture intelligently cooperates with the audit information retrieval subsystem.

3.2 Classification and Model Description of a Financial Audit System

Figure 3.2 shows a hierarchical classification of concepts and their relationships in the financial audit system. The classification shows the “*is-a*” subsumption property, and the “*is-part-of*” composition property in the FAS. The relation “*is-a*” indicates that an entity is a subset of the upper level entity, whereas the “*is-part-of*” relation indicates that the upper level entity is strictly bound by the combination of all lower level entities. The absence of any lower level entity in the “*is-part-of*” relationship renders the upper level object incomplete.

There is a “*is-part-of*” relation between some concepts in a FAS and the *financial system*, for example, a financial system is incomplete without any of these four concepts: *user*, *customer*, *transaction*, and *audit system*. Also, other concepts (*audit module*, *encryption utility*, and *de-encryption utility*) have an *is-part-of* relation with the *audit system*. Other types of relationships between the concepts in a FAS include *uses*, *has*, and so on. *Audit user* has the relation *uses* with the *audit system* (audit user uses the audit

system), and *audit system* has relation *has* with the *audit log* (audit system has an audit log).

The *audit log* looks after recording user *action*, *activity*, and *transaction details*. A *transaction* has both *transaction details*, and *evidential documents*. An *initiator* initiates *transactions*, a *processor* processes the *transaction*, and an *authorizer* authorizes the transaction. *Date* and *time* are part of transaction details.

The *receipt voucher* and *payment voucher* are *evidential documents* in the financial transaction system. A *customer* is an *agent*; also, a *user* of a financial system is an agent. The user has *action* and *activity*. The *user* consists of the *audit* and *non-audit* users. The *processor*, and *authorizer* are *non-audit users*. The *report generator*, *report analyst*, and *audit log custodians* are *audit users*. The *initiator*, *payer*, and *receiver* are *customers*, and the *payer* pays the *receiver*.

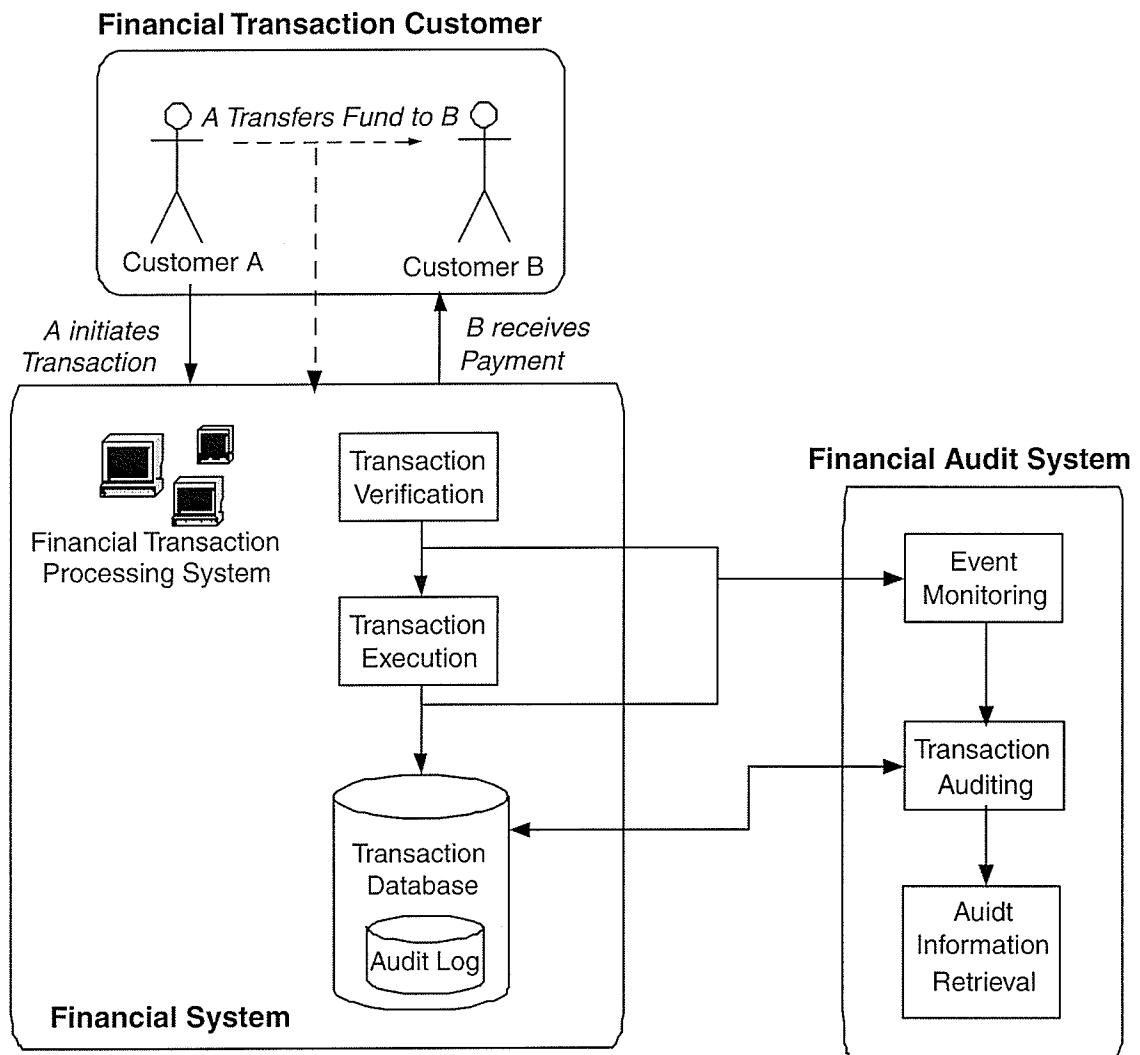


Figure 3.1: Architecture of a Financial Audit System [AEOA04]

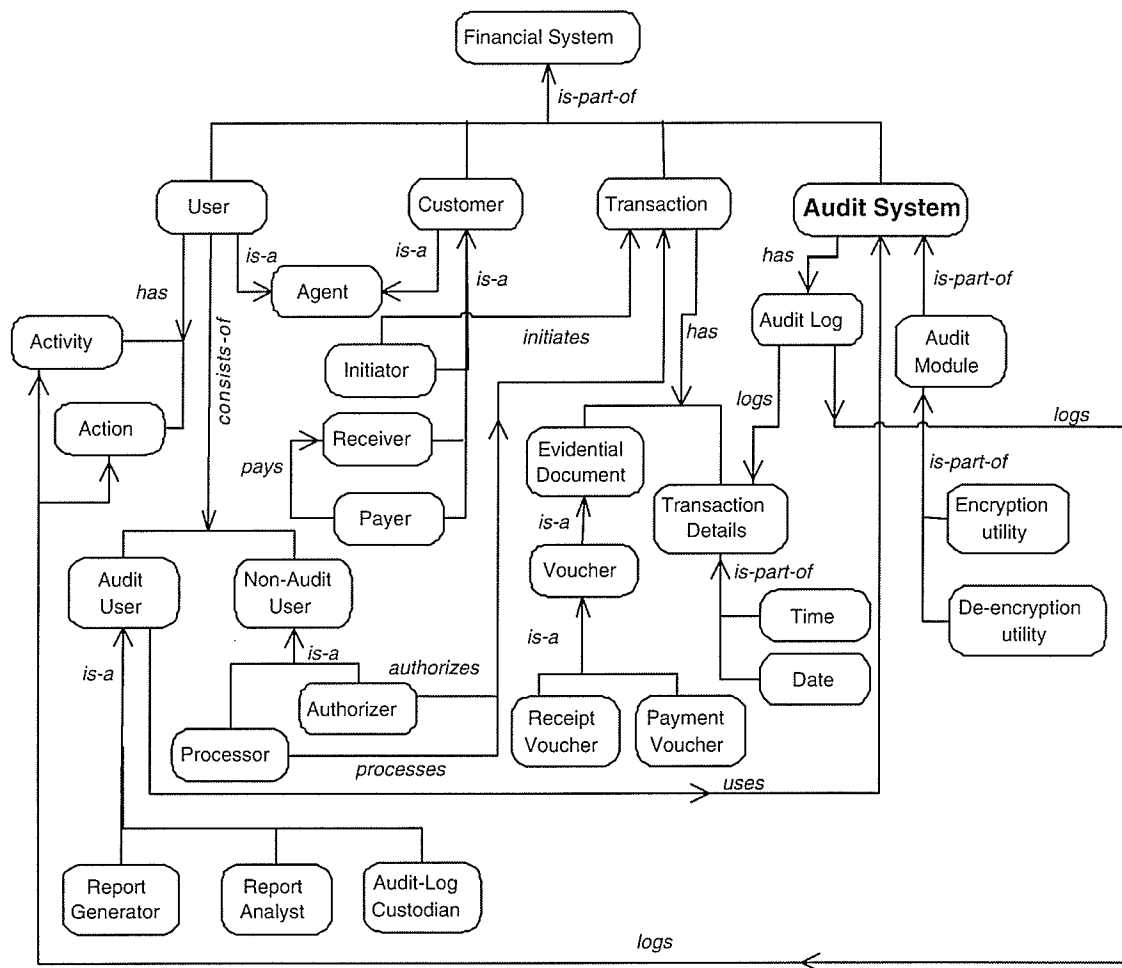


Figure 3.2: Classification of a Financial Audit System [AEOA04]

3.3 Requirements of a Financial Audit System

The exchange of goods and services requires payments. Payments are financial transactions carried out using cash, cheques, debit or credit cards. Despite the continuous flow of financial transactions, there is need to ensure the correctness of all financial data. The financial system is responsible for ensuring the correctness of individual transactions. Part of this process involves affixing electronic time (e-Time) stamps on relevant financial transactions system activities for the purpose of auditing.

Accurate documentation of financial records and timely retrieval of information are crucial factors in financial transactions management. To achieve the required accuracy and consistency of data, the financial transactions' evidential documents should be permanent records. Unfortunately, paper vouchers (a form of evidential documentation) lose value as they pass from hand to hand and age. Similarly, due to inappropriate filing and mishandling of paper vouchers, vital evidential documents are mutilated. The audit system must capture and maintain transaction details and activity logs of both processed and failed transaction data [AE04]. The audit system must preserve evidential documents by maintaining a log of relevant transactions and activities [AE04]. The preserved log of transactions and activity data will significantly aid managerial decision making in both the financial and financial audit systems.

Financial transactions require security and control. An audit system can achieve accountability through the implementation of functional rights and systems authorization limits. The audit system should capture and log user activities on the system; hence, the system should keep an account [Mer03] for who does what, where, when, why, and how. Some financial data require a high level of confidentiality. The incorporation of data encryption into the audit system will achieve a good level of information confidentiality [AE04]. The data encryption module should restrict access, while the activity log will monitor activities on such confidential data.

The mode and manner of reporting and retrieval of audit information in the financial audit system is very important. It is essential that audit reporting is easy to generate and analyze as well as done timely. Audit information generated by different auditors based on the same audit work should be consistent, since the system generates all relevant information and reports from the same data source.

The safety of historical financial transactions data is paramount, as it can serve as evidential documentation. The audit system should eliminate physical movements of audit papers from desk to desk. The use of electronic documents (payment and receipt voucher e-Documents) should leverage the information sharing capabilities of computer technology. In this way, all required processes can use e-Documents to review, verify, and monitor the process of initiating, processing, and finalizing the process of making and receiving payments.

3.4 Overview of the Formal Model

A model is an abstract representation of real world software artifacts. Dutra [Dut02] describes a model as a surrogate of real life applications. Models can be visually represented for easier understanding, or based on rigorous mathematical principles for a concise description of the semantics in the domain of discourse. Modeling financial audit systems requires a modeling language that can succinctly represent the different characteristics of a financial audit system in its entirety.

A model simplifies huge and complex systems into easier, more focused, and understandable units that can be separately analyzed. It decomposes a complex system into more readable, clearer, and simpler systems. Understanding the individual smaller units increases the understanding of the bigger and complex system. Modeling and decomposition of systems enables reuse as well as extensibility of complex systems.

3.5 UML Model of a Financial Audit System

The UML model of a financial audit system in this thesis is captured with static, use case, state machine, activity, and interaction views. These views are captured and represented with the Class, Use Case, Statechart, Activity, and Collaboration diagrams that follow.

3.5.1 Use Case Diagrams

This section presents use case diagrams of a financial audit system. These use case diagrams are categorized into the financial payment use case, financial transaction processing use case, and financial transaction auditing use case.

Financial Payment Use Case

Figure 3.3 shows a high-level interaction of a *Payer* and a *Receiver* with the financial system. Actor *Payer* interacts with the financial system through the use case *Make Payment*, while another actor (*Receiver*) interacts with the financial system through the *Receive Payment* use case.

The *Make Payment* use case represents the entire set of activities that are involved when a *Payer* requests to make a payment through the financial system. Also, the *Receive Payment* use case represents the entire set of processes that are involved whenever the beneficiary receives a payment.

Financial Transaction Processing Use Case

Figure 3.4 shows the interaction of a *Processor* and an *Authorizer* with the financial system. The *Processor* interacts with the financial system through the *Process Transaction*

use case, and the *Authorizer* interacts with the system through the *Authorize Transaction* use case.

The *Process Transaction* use case is a high-level representation of all that happens whenever a *Processor* receives a payment / transfer request, and executes the payment instruction. The *Authorize Transaction* use case denotes the activities of an *Authorizer* whenever there is an authorization request.

Financial Transaction Auditing Use Case

Figure 3.5 shows the interactions and actions of an *Auditor* within the financial system. An auditor interacts with the financial system through use cases *Monitor Financial & User Activities*, *Prevent Financial Loss*, *Ensure Transaction Log*, *Review Transaction Log*, *Detect Exceptions*, *Investigate Exceptions*, and *Correct Exceptions*.

The *Prevent Financial Loss* use case represents the primary goal of an auditor, and all activities that an auditor does in order to prevent financial loss. The *Monitor Financial & User Activities* use case depicts a high-level interaction of an auditor with the financial system in order to monitor both financial and user related activities. The *Ensure Transaction Log* describes the duty of a financial auditor in ascertaining that financial transactions are being logged for subsequent auditing / review with the *Review Transaction Log* use case. The *Detect Exception* use case represents the aspect of an auditor's function in detecting exceptions that exist in financial transactions. The *Investigate Exceptions* use case describes what an auditor does whenever an exception is detected. The *Correct Exceptions* use case represents further action to be taken in order to correct an exception and prevent it from happening again.

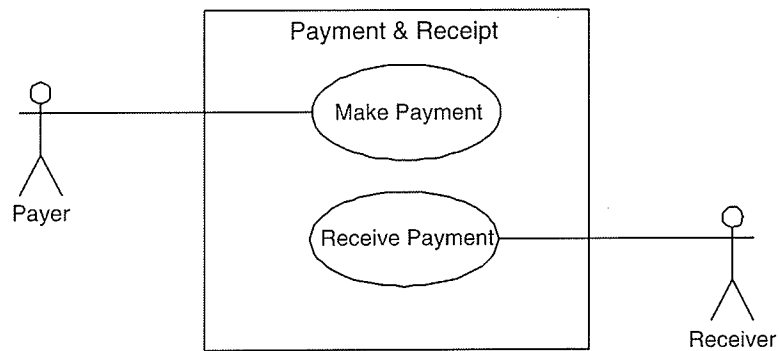


Figure 3.3: Use Case of a Business Transaction in a Financial System

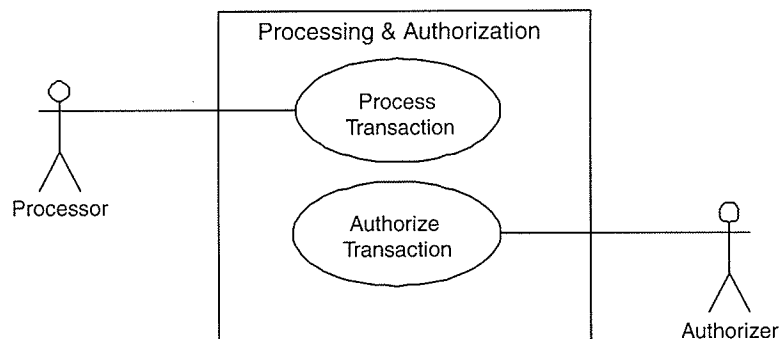


Figure 3.4: Use Case of Processing a Financial Transaction

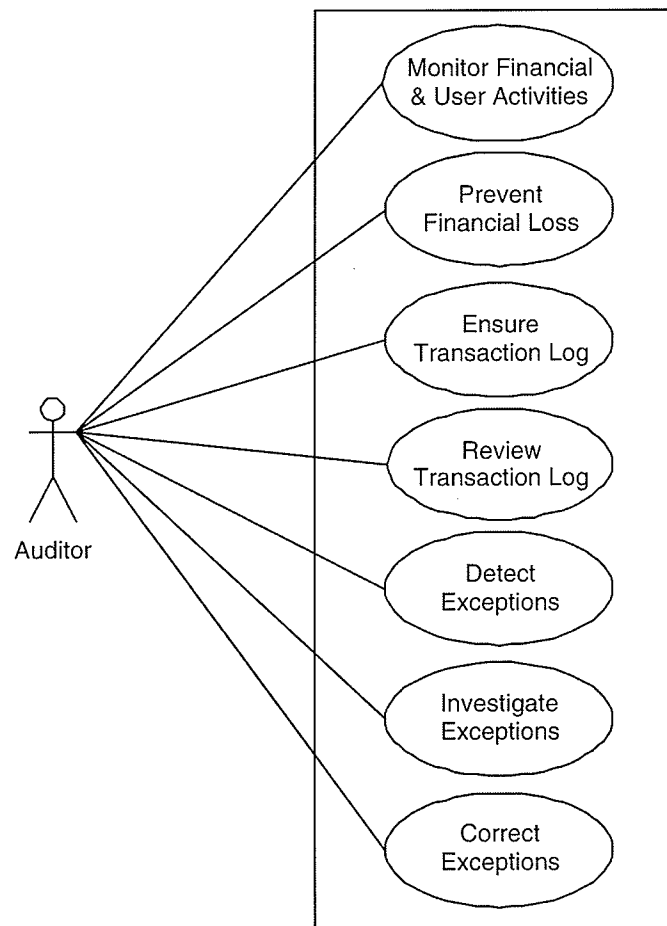


Figure 3.5: Use Case of Auditing a Financial Transaction

3.5.2 Class Diagrams

This section presents class diagrams that describe a financial audit system. These class diagrams are categorized into the financial system class diagram, payment processing class diagram, and financial audit system class diagram.

Financial System Class Diagram

Figure 3.6 shows a financial system class diagram. The class diagram represents a collection of subclasses in a financial system, as well as their attributes and the operations that they perform. The financial system class has attributes *name*, *type* and *database*. These attributes depict that a financial system is identified by its name, the type (there are several types of financial systems, for example, banking, e-commerce, m-commerce, brokerage, insurance, and so on), and a database (a financial system stores its financial data in a database).

Operations in a financial system include transfer of fund, i.e., *TransferFund()* (which encompasses several activities that result in the transfer of fund from one source to another, for example, withdrawing money from a bank account in the form of cash withdrawal, or making a payment from one bank account to another), maintaining deposits in form of savings, i.e., *SaveFund()* (for example, making a cash deposit into a bank account), processing transactions, i.e., *ProcessTransaction()*, and transaction auditing, i.e., *AuditTransaction()*.

The financial system class diagram consists of other subclasses, namely *Processor*, *Transaction*, *Audit System*, and *Authorizer* classes. The class diagram has a composition relationship with these subclasses. The *Processor* class describes attributes and operations of a processor. These attributes include the *Name*, *IdentificationNumber*, user function (*Function*), and access rights (*AccessRight*). Operations of the *Processor* subclass

include: *ProofTransaction()* (a pre-verification of the quality and validity of a payment request, for example, a cheque that is not signed can be detected at this stage by this operation), *RejectTransaction()* (rejecting a payment request that lacks enough details, for example, a cheque without a signature is rejected from being paid), *ProcessTransaction()* (the actual processing / implementation of a payment request, for example, processing a monthly pre-authorized payment request from a bank account to another account in another bank), and *SeekAuthorization()* (seeking an authorization for a transaction that requires additional level of approval, for example, when a withdrawal that exceeds the limit of a particular customer of a bank is requested, an additional authorization is required.)

The *Transaction* class describes attributes and operations of a transaction. A transaction has attributes *Payer*, *Receiver*, *Amount*, *Description* (other payment details) and *Mode* (cash, cheques, debit or credit cards, and so on.) Operations on a transaction include *Succeed()* (a successful transaction), *Fail()* (a failed transaction), *Abort()* (an aborted transaction), *TimeOut()* (a transaction that timed out due to inactivity), *Incomplete()* (an incomplete transaction), and *Complete()* (a completed transaction.)

The *Audit System* class has both the *Name* and *Type* attributes. Operations on the *Audit System* class include *MonitorEvent()* (checking and keeping track of the activities in a financial system for audit purposes), *LogEvent()* (a detailed logging of activities in a financial system that is used for audit purposes), *DetectException()* (discovering the existence of an exception or a deviation that contravenes the rules and procedures in a financial system), *AnalyzeEventLog()* (breaking down the whole audit log into simpler parts for a scientific study in order to discover exceptions to the rules and procedures in a financial system), and *EncryptEventLog()* (encoding the audit log of a financial audit system.) Finally, the *Authorizer* class has attributes *Name*, *IdentificationNumber*, *Function*, and *AccessRight*. Operations on the *Authorizer* class include *ProofTransaction()*, *RejectTransaction()*, and *AuthorizeTransaction()* (justifying and approving the processing of a

financial transaction that requires an additional control from an authorizer.)

Payment Processing Class Diagram

Figure 3.7 shows the Financial System Payment Process class diagram. This class diagram describes relationships that exist between the *Payer* class and the *Financial System* class, as well as between the *Financial System* class and the *Receiver* class. The class diagram consists of the *Payer* class, the *Receiver* class, and the *Financial System* class. The payer class has the “*instructs*” relationship with the financial system. The “*instructs*” relationship describes the process of giving an instruction to a financial system to carry out a financial transaction on behalf of the *Payer*. The Financial System class has the “*pays*” relationship with the *Receiver* class. The “*pays*” relationship describes the action of a financial system that makes a transfer of money to a *Receiver* on behalf of a *Payer*. This class diagram describes the concept that a payer gives a financial system an instruction to transfer funds to a receiver. Also, it describes that a financial system ultimately pays the requested fund to the receiver.

The *Payer* class has *Name*, *BirthDate*, and *IdentificationNumber* attributes, as well as the following operations: *InitiatePayment()* (facilitating the beginning of a payment), *AcknowledgeReceipt()* (admitting and disclosing the receipt of some amount of money), *MakeComplaint()* (formally expressing a dissatisfaction about a financial transaction), and *PaymentDetails()* (providing a detailed description of all the parameters that collectively constitute a financial transaction, for example, payment details of the beneficiary of the transfer.)

The *Financial System* class has been previously described in the Financial System class diagram. The *Receiver* class has attributes *Name*, *Address*, and *IdentificationNumber*, as well as operations *ReceivePayment()*, *AcknowledgeReceipt()*, and *MakeComplaint()*.

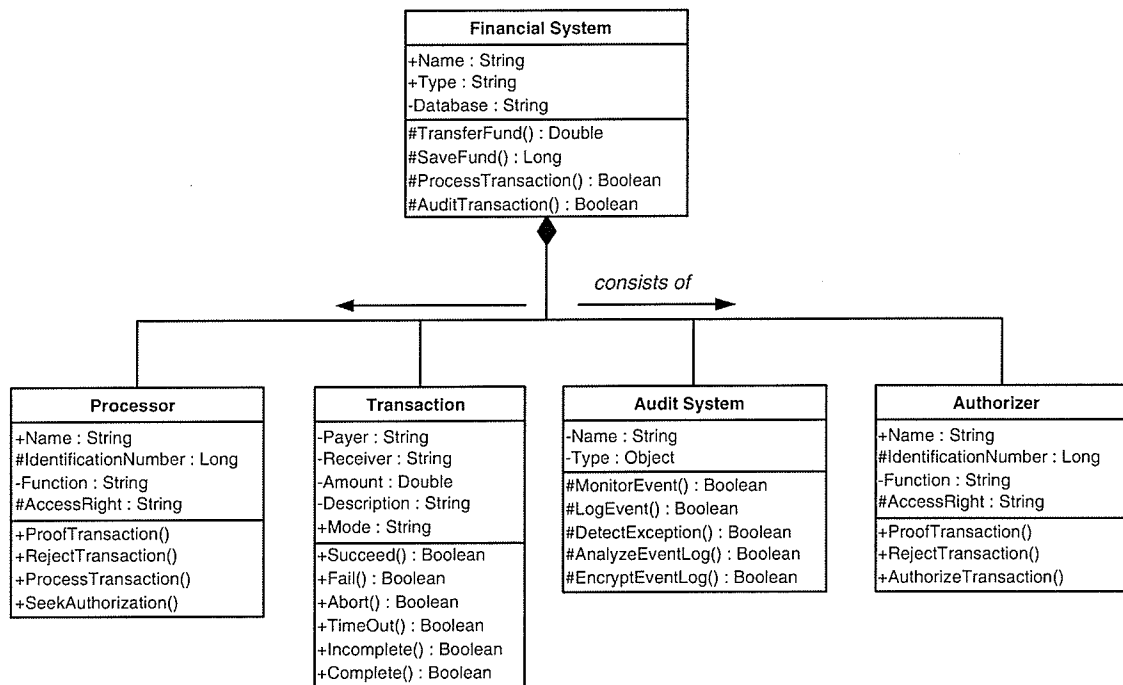


Figure 3.6: Class Diagram of a Financial System

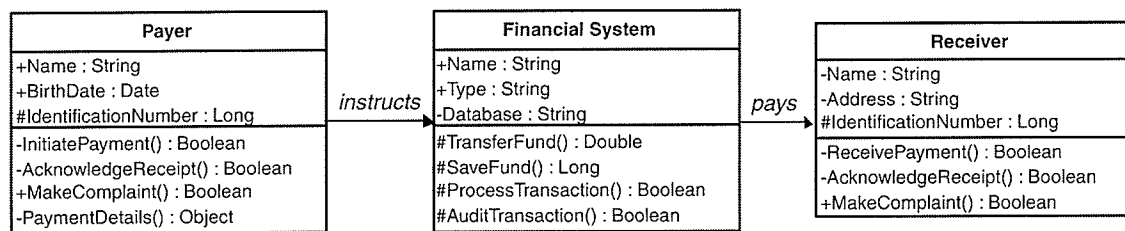


Figure 3.7: Class Diagram of a Financial System Payment Process

Financial Audit System Class Diagram

Figure 3.8 shows the *Financial Audit System* class diagram, and depicts the *Audit System* class as a composition of *Event Log*, *Exception Trigger*, *Exception Handler*, and *Event Analyzer* classes. This class diagram describes associations and relationships that exist in a financial audit system. The *Exception Handler* class has the “initiates” association with the *Exception Action* class. This association indicates that the *Exception Handler* class will initiate the (*Exception Action*) required whenever an exception occurs in the audit system.

Attributes and operations of the *Audit System* class have been previously described. Attributes of the *Event Log* class are *Name* and *Type*, and its operations include: *Record()* (an event log keeps a record / logs the details of a financial transaction), *Encrypt()* (event log encrypts the data in an audit log), *FlagException* (the event log raises a caution / warning flag / message whenever an exception occurs in the audit log), *Review* (event log is reviewed / analyzed regularly), *Report* (exceptions in an audit log are reported), *Backup* (the event log is backed up regularly to ensure the safety of audit log), and *Archive* operation (data of a certain age on the audit log is archived for timely retrieval of audit information from the audit log).

The *Exception Trigger* class works in conjunction with the *Event Log* class to detect exceptions. It works with the *Exception Handler* class to handle exceptions. The *Event Analyzer* class associates with the *Event Log* to provide an analysis for the data in an audit log. Finally, the *Exception Action* class operates to resolve and terminate exceptions.

3.5.3 Activity Diagrams

This section presents activity diagrams of a financial audit system. These activity diagrams are categorized into the financial system activity diagram, and financial audit sys-

tem activity diagram.

Financial System Activity Diagram

The activity diagram in Figure 3.9 shows a logical flow of activities in a financial transaction system. These activities start when a *Payer* initiates a payment process, and it ends when the after-service auditing is done (after the financial system pays the Receiver) as shown in Figure 3.7. The activity diagram shows several activities which include: *Event Monitoring*, *Transaction Initiation*, *Transaction Verification*, concurrent and synchronized activities (*User ID Verification* and *Password Authentication*), *User Rights Granting*, *Transaction Processing*, *Transaction Authorization*, *Payment Voucher Generation*, *Fund Transfer*, *Receipt Voucher Generation*, *Exception Handling*, and *After-Service Auditing*.

The flow of these activities commences from the *Event Monitoring* activity to the *Transaction Initiation*, which leads to *Transaction Verification* activity. Processing the transaction requires the operations of a processor, therefore the concurrent activities of verifying *User ID* and authenticating the user *Password*. In case the user signon is unsuccessful, the flow of activities either ends or loops back in order to do user signon operation again. A successful signon activity leads to granting functional rights for the user, and subsequent processing of the transaction. If an exception occurs, *Exception Handling* activity is triggered, otherwise the transaction processing either requires an authorization or it is successfully done. Thereafter, the after-service auditing activity is initiated, and it encapsulates a series of other coordinated activities (these activities are described in Figure 3.10).

Financial Audit System Activity Diagram

The activity diagram in Figure 3.10 shows detailed after-service auditing activities. The audit system concurrently does user verification and password authentication, and synchronizes the two activities. Thereafter, the system grants users their functional rights. The audit user specifies a range of time to be audited, and the system produces the data in an unencrypted form. The audit user analyzes the data, and produces an audit report.

The *Audit Data Reporting* activity generates several informative audit reports, namely *Payer Audit Report*, *Receiver Audit Report*, *Processor Audit Report*, and *Other Audit Reports* (customized to suit each situation). The *Exception Investigation* and *Exception Resolution* activities handle all exceptions that the audit reports reveal.

3.5.4 Statechart Diagrams

This section presents statechart diagrams that describe dynamic and temporal situations that occur in the financial audit system model. Actions that cause state transitions and the composition of new states resulting from transitions due to temporal activities are outlined. These statecharts capture and describe transitions and state changes in the dynamic aspects of the financial audit system. They are categorized into event monitoring, authorization, payment confirmation, data encryption, and exception review statecharts.

Event Monitoring Statechart

Figure 3.11 shows the event monitoring statechart diagram. This statechart describes the events that occur based on actions that take place in the event monitoring of a financial audit system. The *Payment Initiation* transition causes a state change from the *Event Monitor Inactivity* state to *Event Monitor Activity* state. From here the state changes back

to *Event Monitor Inactivity* on transition *Payment Ends*, a situation that happens when a payment is accomplished.

Payment Processing (No exception in payment criteria) recursively transits the *Event Monitor Activity* state back to itself, and when the *Payment Ends* transition is accomplished, the state changes to the *Event Monitor Inactivity* state. The *Payment Processing (Exception in payment criteria)* transition causes a state change from *Event Monitor Activity* to *Exception Handler Activity* state. This is a financial transaction which cannot be completed due to an invalid condition. This occurs when one of the payment criteria is compromised (for example, an attempt to overdraw the bank account of a company or individual.) *Exception Resolution* transits the *Exception Handler Activity* state to the *Exception Handler Inactivity* state. At this time, *Payment Processing (post exception resolution)* causes a state change from the *Exception Handler Inactivity* state back to the *Event Monitor Activity* state. The *Payment Ends* transition changes the *Event Monitor Activity* state to *Event Monitor Inactivity*. *Payment Process Termination* transits the state machine to the terminal *Final State*.

Transaction Authorization Statechart

The transaction authorization statechart diagram is shown in Figure 3.12. This statechart presents the states and transitions that occur in the transaction authorization aspect of both the financial system and financial audit system. In this statechart diagram, transition *Authorization Request* causes the *Authorizer Inactivity* state to change to the *Authorizer Activity* state.

If there is an exception, such as when there is an attempt to overdraw the bank account of a company, then the *Exception Exists* transitions the *Authorizer Activity* state to the *Exception Handler Activity* state. In the absence of an exception, the *No Exception* transition changes the *Authorizer Activity* state to *Payment Made* state. The *Payment Ends*

transition either transits the *Payment Made* state into the *Authorizer Activity* state (i.e. if *Another Authorization Request* is received), or *Authorizer Inactivity* state (i.e. if *Authorization Terminates* transition occurs.) Finally, *No Authorization Request* transitions on the *Authorizer Inactivity* state leads to the *Final State*.

Payment Confirmation Statechart

The payment confirmation process statechart for the financial audit system is shown in Figure 3.13. This statechart describes the dynamic behavior related to the payment confirmation aspect of a financial system and financial audit system. The *Generate Payment Voucher* transition changes the *Payment Made* state to *Payment Confirmation* state. Transition *Generate Receipt Voucher* changes the *Payment Confirmation* state to *Receipt Confirmation*.

The *Review Transaction* transition changes both the *Payment Confirmation* and *Receipt Confirmation* states to the *Transaction Auditing* state. From here the *Audit Ends* transition leads to the *Final State*.

Note that the audit information forwarded to *Transaction Auditing* differs depending on whether the *Review Transaction* transition is taken from *Payment Confirmation* state or *Receipt Confirmation* state.

Data Encryption Statechart

Figure 3.14 shows the data encryption statechart diagram. This statechart presents the states and transitions that cause state changes in the data encryption aspect of a financial audit system. Initially, the *Event Monitor Activity* state occurs. *Audit Data Logging Starts* changes the *Event Monitor Activity* state to *Data Encryption Activity*. When *Audit Data Logging Ends*, the *Data Encryption Inactivity* state is reached.

Another transition, *Logging Auditor Actions* returns the system to the *Event Monitor Activity* state. When *Event Monitoring Ends* transition is triggered, the *Final State* is reached.

Exception Review Statechart

The exception review process statechart diagram is shown in Figure 3.15. This statechart describes the states and transitions that cause state changes in the exception review aspect of a financial audit system. Auditing begins when an *Audit Information Request* transition changes the *Auditing Inactivity* state to *Auditing Activity* state. When *Audit Data Is Available*, it transitions the *Auditing Active* state to *Audit Information Analysis* state. An exception at this time will (through *Audit Exceptions*) transition *Audit Information Analysis* to *Exception Review* state. The *Final State* is reached if *No Audit Exceptions* are present.

The *Audit Action Recommendation* transitions the *Exception Review* state to the *Audit Action Implementation* state. Finally, the *Audit Action Implementation Ends* transition leads the *Audit Action Implementation* state to the *Final State*.

3.5.5 Collaboration Diagrams

Collaboration diagrams show systems of objects that cooperatively work together to achieve a common purpose. Collaboration diagrams show behavioural interactions among several constituent objects in a system, and how they exchange messages amongst themselves.

Financial Audit System Collaboration Diagram

Figure 3.16 is a collaboration diagram that shows an interaction among objects in the financial audit system. The classifier roles in the system are *Payer*, *Processor*, *SoftwareApp*

(software application), *Receiver*, *Authorizer*, *AuditLog*, *Auditor*, *AuditReport*, and *Management* (managerial decision making process in a financial system).

The *Payer* and *Receiver* classifier roles belong to *Persons*. The *Processor*, *Authorizer*, *SoftwareApp*, and *Management* belong to the *Financial System*. The *Auditor* and *Audit Log* belong to the *Audit System*. Finally, the *Audit Report* classifier role belongs to the Management Information System (*MIS*). This is a system that helps in making managerial decisions with the use of Information Technology. The *SoftwareApp* represents the entire software components (user interface and the database) of a financial system including audit data. The association roles are presented with ordered numbers. A list of the association roles in the collaboration diagram and their meanings is presented below:

- 1 *initiatesPayment()* – when a *Payer* initiates payment in the financial system, the payment instruction subsequently gets to the *processor*. This represents an action between a *Payer* and a *Processor*.
- 2.1 *logsAuditData()* – the *Processor* inputs financial data, while the audit system logs transactions details into an audit log – this represents an action between a *Processor* and the *AuditLog*.
- 2.2 *storesProcessDetails()* – the financial software application / database stores details of financial transactions – this represents an action between the *Processor* and the *SoftwareApp* (software application.)
- 2.3 *requestsAuthorization()* – whenever a transaction requires an authorization, and a *Processor* requests for the authorization – this represents an action between a *Processor* and an *Authorizer*.
- 2.4 *confirmsAuthorization()* – the system provides a confirmation for an authorization – this represents an action between a *Processor* and an *Authorizer*.

- 3.1 *logsAuditData()* – the audit system logs the authorization details of the transaction – an action between an *Authorizer* and the *AuditLog*.
- 3.2 *storesAuthDetails()* – the financial software application / database stores details of the transaction authorization – an action between an *Authorizer* and the *SoftwareApp*.
- 4 *finalizePayment()* – the financial system finalizes a payment. The *Receiver* gets the value of the transaction – an action between a *Receiver* and the *SoftwareApp*.
- 5.1 *queriesAuditData()* – an *Auditor* queries the *AuditLog* for audit data – an action between an *Auditor* and the *AuditLog*.
- 5.2 *analyzesAuditData()* – an *Auditor* analyzes an audit data – an action between an *Auditor* and the *AuditLog*.
- 5.3 *generatesAuditReport()* – an *Auditor* generates an audit report – an action between an *Auditor* and the *AuditReport*.
- 6.1 *usesAuditReport()* – the *Management* uses the audit report for managerial decisions – an action between the *Management* and the *AuditReport*.
- 6.2 *makesFinancialDecisions()* – the *Management* uses the audit report for financial-related decisions – an action between the *Management* and the *AuditReport*.
- 6.3 *makesSecurityDecisions()* – the *Management* uses the audit report for security-related decisions – an action between the *Management* and the *AuditReport*.

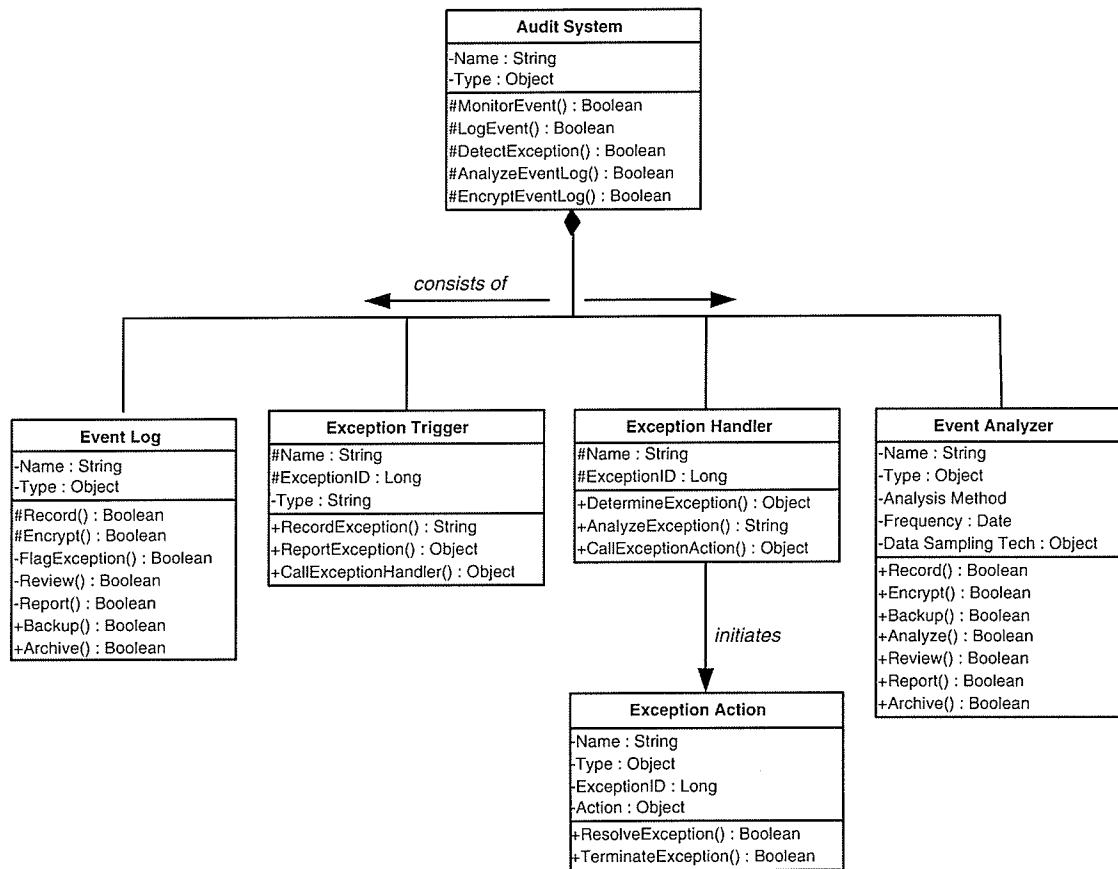


Figure 3.8: Class Diagram of a Financial Audit System

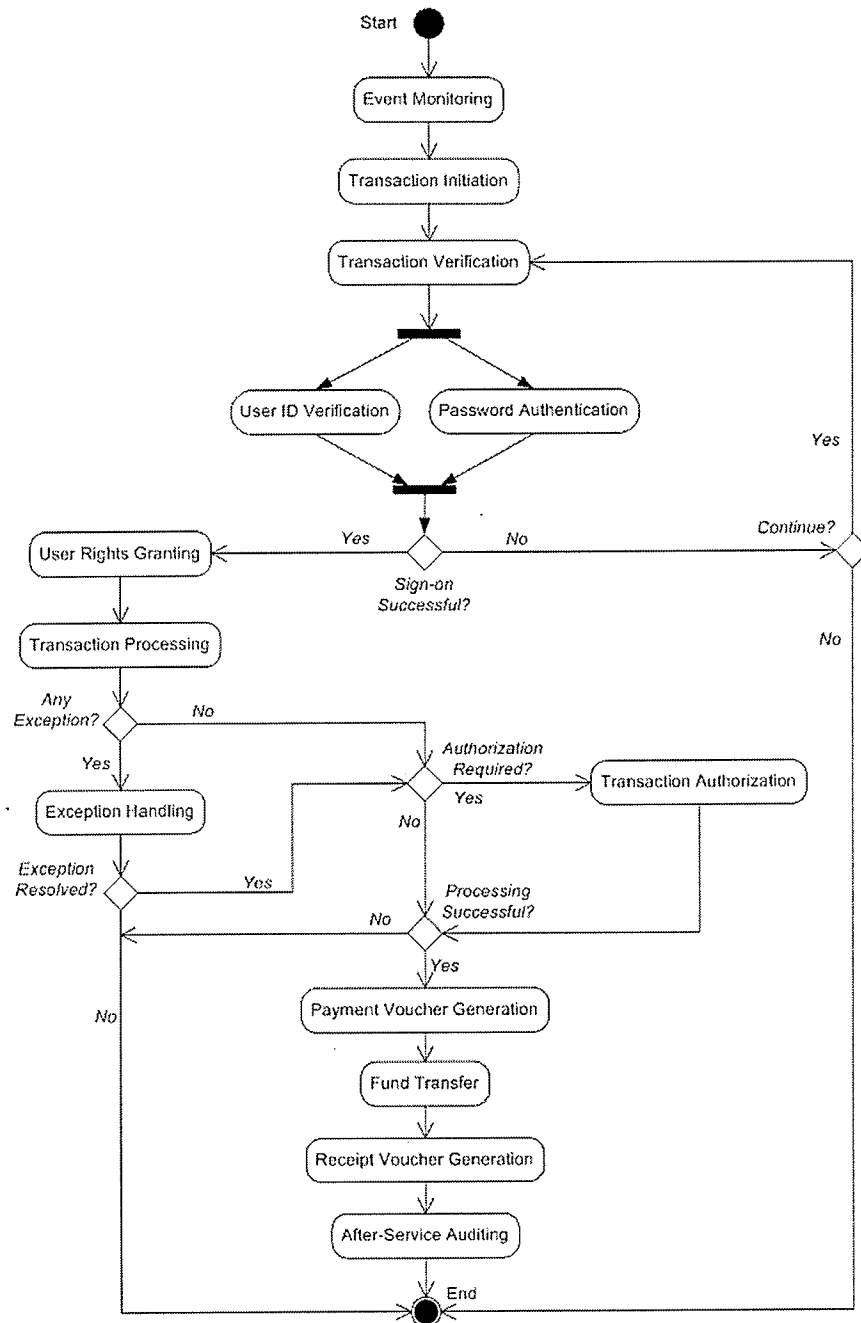


Figure 3.9: Activity Diagram of a Financial System

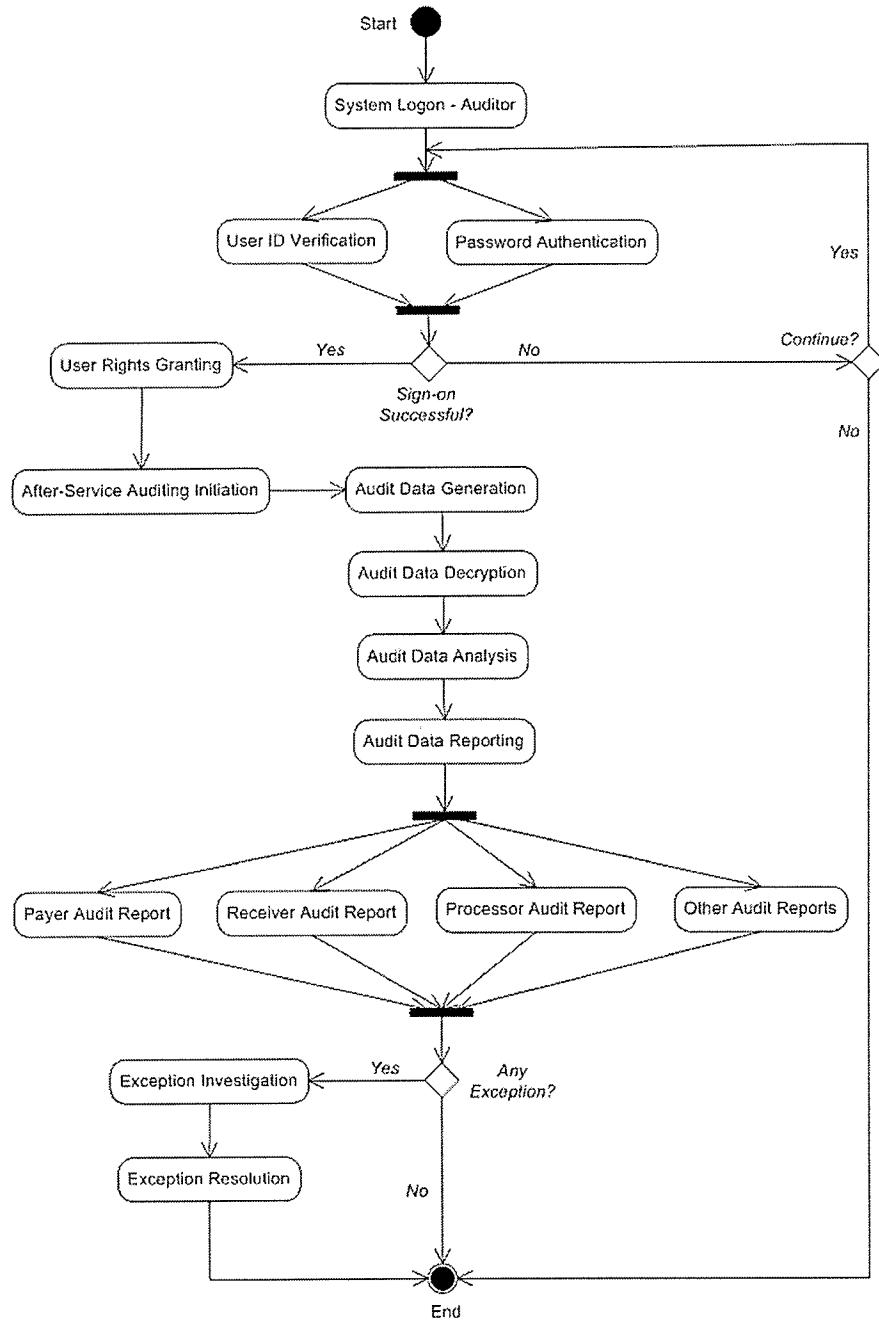


Figure 3.10: Activity Diagram of a Financial Audit System

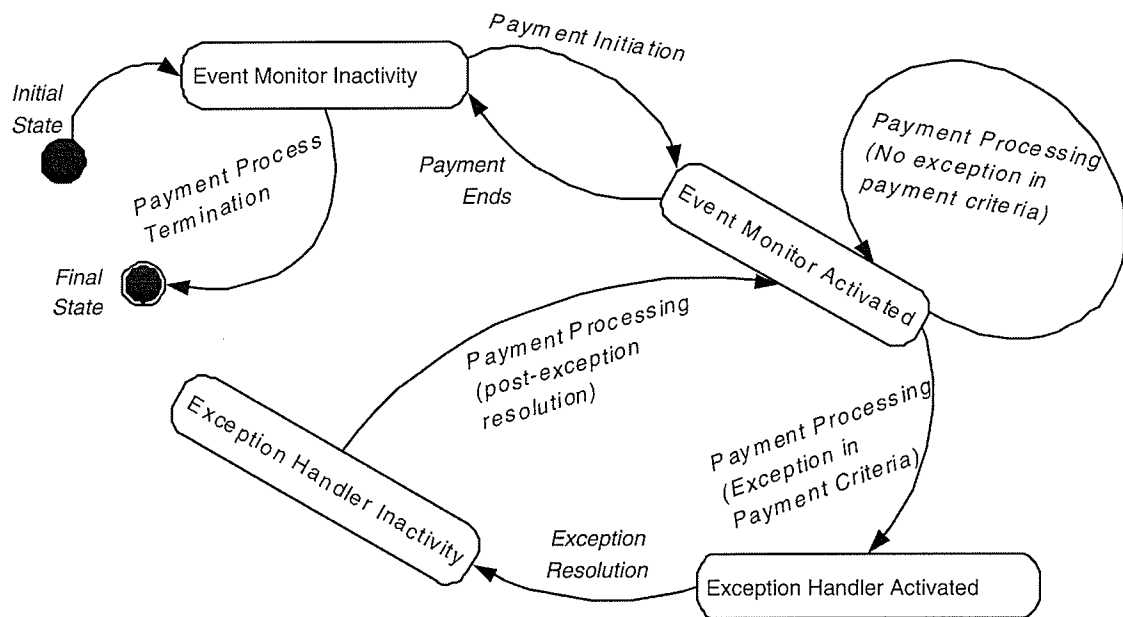


Figure 3.11: Financial Audit System Event Monitoring Statechart Diagram

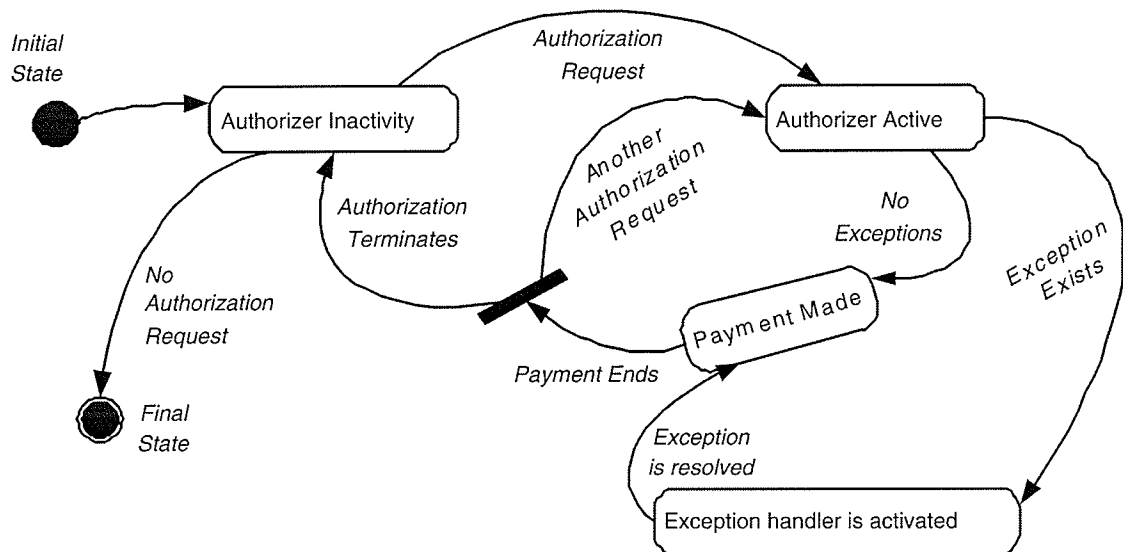


Figure 3.12: Financial Audit System Transaction Authorization Statechart Diagram

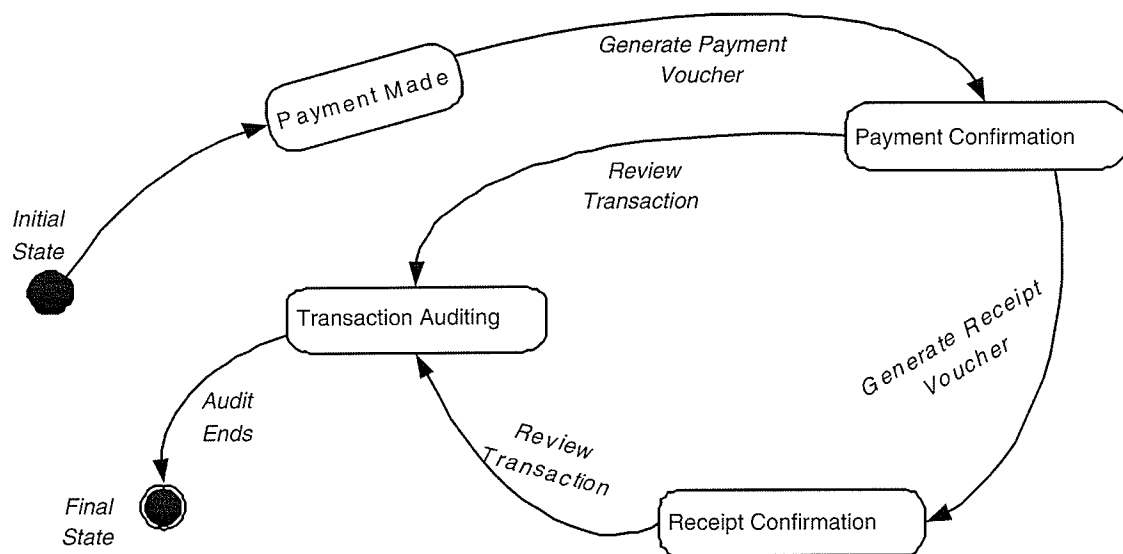


Figure 3.13: Financial Audit System Payment Confirmation Statechart Diagram

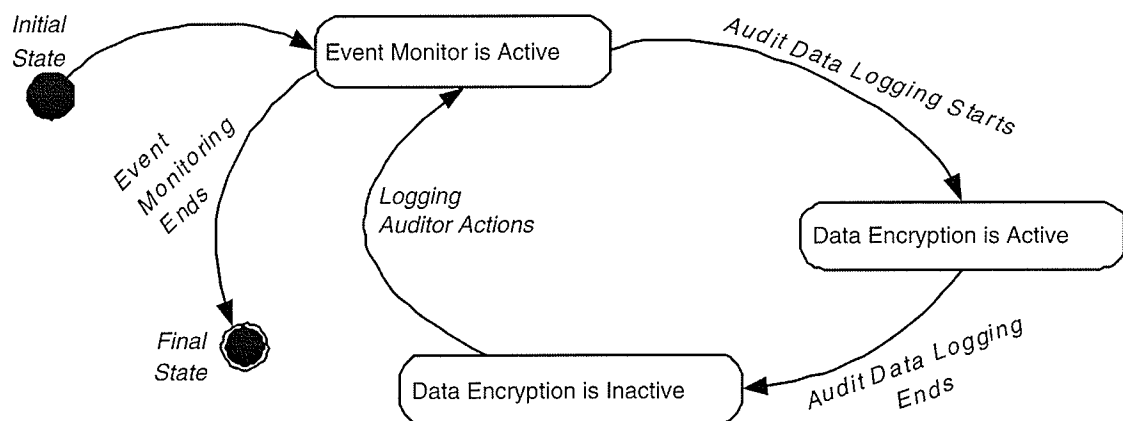


Figure 3.14: Financial Audit System Data Encryption Statechart Diagram

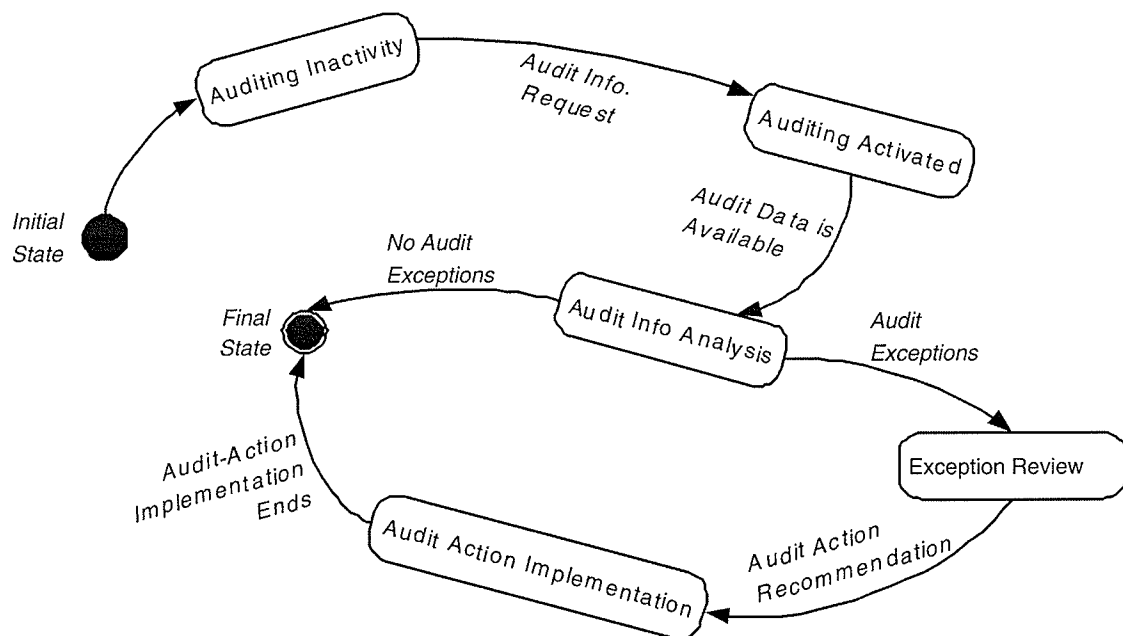


Figure 3.15: Financial Audit System Exception Review Statechart Diagram

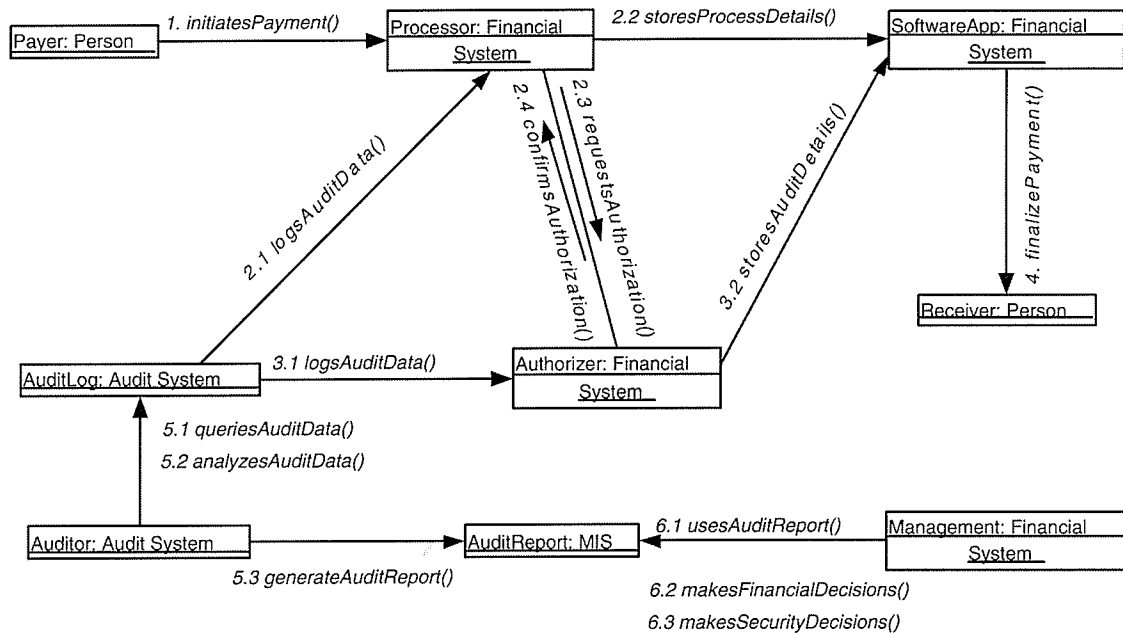


Figure 3.16: Collaboration Diagram of a Financial Audit System

3.6 Predicate Logic Specifications of a Financial Audit System

This section presents a formal specification of the requirements of a financial audit system in predicate logic language. In order to aid the understanding of the formal specifications, informal definitions of the terms and symbols I use are provided first. This is followed by formal definitions of the terms and symbols.

3.6.1 Informal Definitions of Terms and Symbols

Informal descriptions of the terms used in the predicate logic specifications in this thesis are outlined below. For each term, the meaning and an explanation / description of the term is provided.

Variables:

Table 3.1: Variables of a Financial Audit System Predicate Logic Specifications

Term	Meaning	Description
ac	Action	An atomic unit of user activity in the financial system.
ana	Audit Analysis	Analysis of audit log.
au	Authorizer	A person that verifies the quality of financial transactions processed by processors.
av	Activity	A combination of all operations of a user in a financial system.
\bar{a}	Unauthorized Access	Unauthorized access to a resource in a financial system.

continued on next page

Table 3.1 – *continued from previous page*

Term	Meaning	Description
ctr_A	Additional Control	The process of an authorizer verifying some high-valued transactions prior to completing the transaction processing.
d	Duty(ies)	Specific function(s) assigned to a user.
d_M	Decryption Module	Decryption module that decrypts audit log data for subsequent analysis.
$data$	Data	Units of financial records.
$data_H$	Huge Volume	Large volume (count) of the record of transactions in a financial audit log.
$date$	Date	The date of a financial transaction.
e_M	Encryption Module	Encryption module that encrypts the data in an audit log.
f	Signon Attempts	The count of a user sign-on attempts into a financial system.
i	Initiator	An agent that initiates a financial transaction.
l	Log	Audit log / trail of financial transactions.
l_A	Log Access	Logging of user access into a financial system.
l_U	Log Usage	Logging the usage of a financial system.
p	Payer	An agent that provides the money to be transferred in a financial transaction.
pid	Payer Id	Payer identification symbol.

continued on next page

Table 3.1 – *continued from previous page*

Term	Meaning	Description
<i>pr</i>	Processor	A person that is authorized to process financial transactions.
<i>pv</i>	Payment Voucher	A documentary evidence of financial transactions, given to an initiator or a payer that is involved in a financial transaction.
<i>pwd</i>	User Password	User secret password.
<i>r</i>	Receiver	An agent that ultimately receives money through the financial transaction.
<i>rid</i>	Receiver Id	Receiver identification symbol.
<i>rpt</i>	Reporting	Report of the result of an audit work.
<i>rv</i>	Receipt Voucher	A documentary evidence of financial transactions, given to a receiver that is involved in a financial transaction.
<i>s_F</i>	Failed Signon	Failed logon attempt into a financial system.
<i>s_S</i>	Successful Signon	Successful logon attempt into a financial system.
<i>st</i>	Status	Status of a financial transaction.
<i>sys_A</i>	System Access	User access into a financial system.
<i>sys_U</i>	System Usage	Usage of the financial system by a user.
<i>td</i>	Transaction Details	Specific details / properties of a financial transaction.
<i>td_C</i>	Complete Transaction	Details of transactions that are completed.

continued on next page

Table 3.1 – *continued from previous page*

Term	Meaning	Description
td_I	Incomplete Transaction	Details of transactions that are incomplete (incomplete transaction.)
tg	Pre-defined Audit Triggers	Pre-defined triggers that initiate certain audit processes.
$time$	Time	The time of a financial transaction.
$time_I$	Idle Time	The duration of time a user stays idle on a financial system.
tx	Transaction	Transfer of fund from one person (a payer) to another person (a receiver.)
tx_H	High-Valued Transactions	Financial transactions that involve a high monetary value as previously described by the financial organization.
u	User	A person that is authorized to perform assigned functions in a financial system.
u_A	Audit Log User	A person that is authorized to perform audit functions in a financial audit system.
u_N	Non-Audit Log User	A person that is not authorized to perform audit functions in a financial audit system.
uid	User Id	User identification symbol.
\bar{u}	Unauthorized Use	Unauthorized usage of a resource in a financial system.

3.6.2 Formal Definitions of Terms and Symbols

In this section, I present formal definitions and descriptions of the terms, symbols, constants, functions, and predicates that are used in the predicate logic specifications in this thesis.

Constants:

These are predicate logic terms that do not change because their values are fixed.

Constant	Meaning	Data Type
$time_O$	Permissible system idle time	Numeric
f_P	Permissible signon fail attempts	Numeric

Functions:

These are relations between the elements of a set in a domain. They are used for deriving a value from another.

Function	Usage	Meaning
Monitor	$monitor(x, y)$	x monitors y . For this system, $x' = x \hat{y}$; where x represents a current state of an audit log, y represents a new audit data, and x' represents a new state of the audit log (with the new audit data appended to the current audit log).
Number of	$numberof(x)$	number of x (a count of the number of objects in x).

Predicates:

These are functions whose co-domain is the set of Boolean logical constants $\{TRUE, FALSE\}$. Predicate logic functions used in this thesis are specified in the following format:

- **Predicate Symbol:** The name given to a predicate function. For example, *Initiated*.
- **Arity:** The number of terms in the predicate logic formula. For example, the arity of *Initiated*(x, y) is 2 (there are two variables x and y).
- **Atomic Sentence:** The usage of a predicate logic formula. For example, *Initiated*(x, y) is a predicate logic atomic sentence.
- **Meaning / Description:** A description of the predicate logic atomic sentences, and their meanings.

Table 3.2: Predicates: FAS Predicate Logic Specifications

Predicate Symbol	Atomic Sentence / Usage	Meaning / Description
<i>Initiated</i>	<i>Initiated</i> (x, y).	<p>x is initiated by y, where x represents a financial transaction, and y represents an initiator of a financial transaction; the arity is 2, and we say, “transaction x is initiated by initiator y.”</p> <p>Formally,</p> $\forall x \exists y : \text{Initiated}(x, y) \Rightarrow (y \neq \emptyset)$ $\wedge (x \neq \emptyset) \wedge (\forall z : \text{Initiated}(x, z) \Rightarrow (y = z))$

continued on next page

Table 3.2 – continued from previous page

Predicate Symbol	Usage / Atomic Sentence	Meaning / Description
<i>Unique</i>	$Unique(x, y).$	<p>x has a unique y, where $x \in \{Payer, Receiver, User\}$, and $y \in \{Payer\ id, Receiver\ id, User\ id\}$. The arity is 2, and we could say, for example,</p> <ol style="list-style-type: none"> 1. “Payer x has a unique Payer id y”, 2. “Receiver x has a unique Receiver id y”, and 3. “User x has a unique User id y”. <p>Formally,</p> $\forall x \exists_1 y : Unique(x, y) \Rightarrow$ $(x \neq \emptyset) \wedge (y \neq \emptyset) \wedge$ $(\forall z \notin \emptyset : Unique(x, z) \Rightarrow$ $(y = z))$

continued on next page

Table 3.2 – continued from previous page

Predicate Symbol	Usage / Atomic Sentence	Meaning / Description
<i>Minimum</i>	<i>Minimum</i> (x, y).	<p>x has a minimum number of y, where $x \in Transaction$, and $y \in \{Payer, Receiver, Payment\ voucher, Receipt\ voucher\}$. The arity is 2, and we could say “transaction x has a minimum number of Payer y, Receiver y, Payment voucher y, and Receipt voucher y”. This minimum number is 1. Also, see Section 3.2 for a description of the ‘has’ relationship.</p> <p>Formally,</p> $\forall x \exists y : Minimum(x, y) \Rightarrow$ $x \notin \emptyset \wedge y \notin \emptyset \wedge$ $(y \geq 1) \wedge (x \text{ has } y)$

continued on next page

Table 3.2 – continued from previous page

Predicate Symbol	Usage / Atomic Sentence	Meaning / Description
<i>Verified</i>	$Verified(x, y, z).$	<p>High-valued transaction x, processed by y, is verified by z, where $x = \{High-Valued Transaction\}$, $y = Processor$, and $z = Authorizer$. The arity is 3.</p> <p>Formally,</p> $\forall x \exists y, z : Verified(x, y, z) \Rightarrow x \neq \emptyset \wedge y \neq \emptyset \wedge z \neq \emptyset \wedge (y \neq z)$

continued on next page

Table 3.2 – continued from previous page

Predicate Symbol	Usage / Atomic Sentence	Meaning / Description
<i>Known</i>	<i>Known</i> (<i>x</i> , <i>y</i>).	<p><i>x</i> has a known <i>y</i>, where $x \in \{Transaction, Activity\}$, and $y \in \{Date, Time, User\ id\}$. The arity is 2, and we could say, for example,</p> <ol style="list-style-type: none"> 1. “Transaction <i>x</i> has a known Date <i>y</i>, Time <i>y</i>, and User id <i>y</i>”, and 2. “Activity <i>x</i> has a known Date <i>y</i>, Time <i>y</i>, and User id <i>y</i>”. <p>Formally,</p> $\forall x \exists y : Known(x, y) \Rightarrow$ $(x \neq \emptyset) \wedge (y \neq \emptyset)$

continued on next page

Table 3.2 – continued from previous page

Predicate Symbol	Usage / Atomic Sentence	Meaning / Description
<i>Log</i>	$Log(x, y).$	<p>x is logged into y, where $x = Activities$, and $y = Audit\ log$; the arity is 2, and we say “Activities x is logged into Audit log y”.</p> <p>Formally,</p> $\forall x \exists y : Log(x, y) \Rightarrow$ $x \neq \emptyset \wedge y \neq \emptyset \wedge y' = y \hat{=} x$
<i>Has</i>	$Has(x, y).$	<p>x has y, where $x = User$, and $y \in \{Duties, Activities\}$. The arity is 2, and we could say, for example,</p> <ol style="list-style-type: none"> 1. “User x has Duties y”, and 2. “User x has Activities y”. <p>Formally,</p> $\forall x \exists y : Has(x, y) \Rightarrow x \neq \emptyset \wedge y \neq \emptyset$

continued on next page

Table 3.2 – continued from previous page

Predicate Symbol	Usage / Atomic Sentence	Meaning / Description
<i>Documented</i>	<i>Documented(x).</i>	<p>x is documented into y, where $x \in \{Date, Time, Transaction\ details\}$, and $y = Audit\ log$. The arity is 1, x is a bound variable, y is a free variable, and we could say, for example,</p> <ol style="list-style-type: none"> 1. “Date x is documented”, 2. “Time x is documented”, and 3. “Transaction details x is documented”. <p>Formally,</p> $\forall x \exists y : Documented(x) \Rightarrow$ $x \neq \emptyset \wedge y' = y \hat{x}$

continued on next page

Table 3.2 – continued from previous page

Predicate Symbol	Usage / Atomic Sentence	Meaning / Description
<i>Identified</i>	<i>Identified</i> (x, y).	<p>x is identified by y, where $x \in \{Payer, Receiver, User\}$, and $y \in \{Payer\ id, Receiver\ id, User\ id\}$. The arity is 2, and we could say, for example,</p> <ol style="list-style-type: none"> 1. “Payer x is identified by Payer id y”, 2. “Receiver x is identified by Receiver id y”, and 3. “User x is identified by User id y”. <p>Formally,</p> $\forall x \exists_1 y : Identified(x, y) \Rightarrow$ $(x \neq \emptyset) \wedge (y \neq \emptyset) \wedge$ $(\forall z \neq \emptyset : Identified(x, z) \Rightarrow$ $(y = z))$

continued on next page

Table 3.2 – continued from previous page

Predicate Symbol	Usage / Atomic Sentence	Meaning / Description
<i>Protected</i>	<i>Protected</i> (x, y).	<p>x is protected from y, where $x \in \{\text{Audit log, Audit triggers, Audit analysis}\}$, and $y = \{\text{Unauthorized user}\}$. The arity is 2, and we could say, for example,</p> <ol style="list-style-type: none"> 1. “Audit log x is protected from Unauthorized user y”, 2. “Audit trigger x is protected from Unauthorized user y”, and 3. “Audit analysis x is protected from Unauthorized user y”. <p>Formally,</p> $\forall x, y : \text{Protected}(x, y) \Rightarrow$ $x \neq \emptyset \wedge y \neq \emptyset \wedge$ $(\neg \exists y : \neg \text{Protected}(x, y))$

continued on next page

Table 3.2 – continued from previous page

Predicate Symbol	Usage / Atomic Sentence	Meaning / Description
<i>Denied</i>	<i>Denied</i> (x, y).	<p>x is denied y, where $x = User$, and $y \in \{System\ access, System\ usage\}$. The arity is 2, and we could say, for example,</p> <ol style="list-style-type: none"> 1. “User x is denied System access y”, and 2. “User x is denied System usage y”. <p>Formally,</p> $\forall x, y : Denied(x, y) \Rightarrow$ $x \neq \emptyset \wedge y \neq \emptyset \wedge$ $\forall x \neg \exists y : \neg Denied(x, y)$

continued on next page

Table 3.2 – continued from previous page

Predicate Symbol	Usage / Atomic Sentence	Meaning / Description
<i>Segregated</i>	<i>Segregated</i> (x, y, z).	<p>Duties x of Audit users y is segregated from Duties x of Non-Audit users z, where $x = Duties$, $y = Audit\ users$, and $z = Non-Audit\ users$. Segregation describes the mutual exclusivity of the duties of audit and non-audit users. The arity is 3.</p> <p>Formally,</p> $\forall y, z \exists x : Segregated(x, y, z) \Rightarrow$ $x \neq \emptyset \wedge y \neq \emptyset \wedge z \neq \emptyset \wedge$ $(\forall x : Has(y, x) \neq Has(z, x))$

continued on next page

Table 3.2 – continued from previous page

Predicate Symbol	Usage / Atomic Sentence	Meaning / Description
<i>Encrypted</i>	<i>Encrypted</i> (x, y).	<p>x is encrypted by y, where $x = \{\text{Audit data in an audit log}\}$, and $y = \{\text{Encryption module}\}$; the arity is 2, and we say “Audit data in an audit log x is encrypted by an Encryption module y”.</p> <p>Formally,</p> $\forall x \exists y : \text{Encrypted}(x, y) \Rightarrow$ $x \neq \emptyset \wedge y \neq \emptyset \wedge$ $\neg \exists x : \neg \text{Encrypted}(x, y)$

continued on next page

Table 3.2 – continued from previous page

Predicate Symbol	Usage / Atomic Sentence	Meaning / Description
<i>Decrypted</i>	<i>Decrypted</i> (<i>x</i> , <i>y</i>).	<p><i>x</i> is decrypted by <i>y</i>, where <i>x</i> = {<i>Audit data extracted for analysis</i>}, and <i>y</i> = {<i>Decryption module</i>}; the arity is 2, and we say “Audit data extracted for analysis <i>x</i> is decrypted by a Decryption module <i>y</i>”.</p> <p>Formally,</p> $\forall x \exists y : \text{Decrypted}(x, y) \Rightarrow$ $x \neq \emptyset \wedge y \neq \emptyset \wedge$ $\neg \exists x : \neg \text{Decrypted}(x, y)$

continued on next page

Table 3.2 – continued from previous page

Predicate Symbol	Usage / Atomic Sentence	Meaning / Description
<i>Logoff</i>	<i>Logoff</i> (<i>x</i>).	<p><i>x</i> is logged off, where $x = User$, system idle time $t \in Time$, permissible system idle time $Time_I \in Time$, $y \in Authorized\ access$, and $z \in Authorized\ usage$; the arity is 1, and we say “User x is logged off”. This predicate evaluates to <i>TRUE</i> when the system idle time exceeds the permissible time threshold that is set for the system to idle. Also, the User x will be denied both system access and system usage.</p> <p>Formally,</p> $\forall x : Logoff(x) \Rightarrow$ $\exists t \forall y, z : Denied(x, y) \wedge$ $Denied(x, z) \wedge x \neq \emptyset \wedge$ $t > Time_I \wedge y \neq \emptyset \wedge z \neq \emptyset$

continued on next page

Table 3.2 – continued from previous page

Predicate Symbol	Usage / Atomic Sentence	Meaning / Description
<i>Transparent</i>	<i>Transparent</i> (<i>x</i> , <i>y</i>).	<p><i>x</i> is transparent to <i>y</i>, where $x = \{\text{Audit transaction data monitoring activities}\}$, and $y = \text{User}$. The predicate <i>Transparent</i> is valid iff (if and only if) User <i>y</i> is not aware of <i>x</i>, and the monitoring activities <i>x</i> does not usurp too much of system resources to impact on the performance of the financial system. The arity is 2, and we say “Audit transaction data monitoring activities <i>x</i> is transparent to User <i>y</i>”.</p> <p>Formally,</p> $\forall x \forall y : \text{Transparent}(x, y) \Rightarrow$ $x \neq \emptyset \wedge y \neq \emptyset \wedge$ $\neg \exists y : \neg \text{Transparent}(x, y)$

continued on next page

Table 3.2 – continued from previous page

Predicate Symbol	Usage / Atomic Sentence	Meaning / Description
<i>Report</i>	<i>Report</i> (x, y).	<p>x reports y, where $x = \{\text{Audit report}\}$, and $y \in \{\text{User id, Transaction details, Successful user signon, Failed user signon, Completed transaction details, Incomplete transaction details}\}$. The arity is 2, and the predicate <i>Report</i> describes the classification of audit reporting in a FAS. For example, an auditor might require audit information based on several criteria. Audit reporting criteria identified in this thesis include user identification audit report. In this case, we say “Audit report x reports User id y”, meaning that the audit report query is based on user identification. Similarly, the Audit report x can provide audit information that based on Transaction details y, Successful user signon y, Failed user signon y, Completed transaction details y, and Incomplete transaction details y. Formally,</p> $\forall y \exists x : \text{Report}(x, y) \Rightarrow y \neq \emptyset \wedge x \neq \emptyset$

continued on next page

Table 3.2 – continued from previous page

Predicate Symbol	Usage / Atomic Sentence	Meaning / Description
<i>Analyzed</i>	<i>Analyzed</i> (x, y).	<p>x is analyzed by y, where $x = \{\text{Decrypted audit data}\}$, and $y = \text{Auditor}$. The predicate <i>Analyzed</i> describes the analysis of audit information / reports by auditors. This analysis can only be done on decrypted audit information / reports. The arity is 2, and we say “Decrypted audit data x is analyzed by Auditor y”. Any data that is not decrypted (i.e. $\neg x$) can not be analyzed by Auditor y.</p> <p>Formally,</p> $\forall x \exists y : \text{Analyzed}(x, y) \Rightarrow$ $x \neq \emptyset \wedge y \neq \emptyset \wedge$ $(\text{Analyzed}(\neg x, y) \Rightarrow \text{FALSE})$

3.7 Requirements of a Financial Audit System and their Specifications with Predicate Logic

So far I have formally outlined the terms and symbols I use to specify a financial audit system. Now, I will use them to specify the system. These requirements are categorized into accountability, security, transaction monitoring, event logging, and reporting requirements. This categorization follows from [AE04], [CICA03], and [TOG98].

3.7.1 Accountability Requirements and Specifications

An audit system should be able to clearly provide information about who uses the system, when the system was used, and what the system was used for. All these are captured in accountability requirements. These accountability requirements and their formal specifications are expressed as follows:

Accountability Requirement 1: For every transaction, there is one and only one initiator [AE04]:

$$\forall tx \exists_1 i : Initiated(tx, i)$$

where \exists_1 indicates a unique (one and only one) existential quantification.

Every financial transaction requires an initiator. This initiator can be a payer, a receiver, or a third party to both the payer and the receiver of the payment. For example, a payer initiates the payment for a transaction in a grocery store by going to pay for items bought

at the checkout counter. A receiver initiates the payment for a service in a pre-authorized payment arrangement. For example, a banking service establishes an automatic and periodical service charges that are deducted from customers chequing accounts. A third party initiates a payment whenever a collections agent (or a similar body) initiates or requests a payment from a party on behalf of another party. For example, an agent for an insurance company that requests insurance premium payment from a person on behalf of an insurance company is an initiator.

Accountability Requirement 2: For every transaction, there is at least one payer and one receiver [AE04]:

$$\forall tx \exists p, r : \text{Minimum}(tx, p) \wedge \text{Minimum}(tx, r) \wedge \\ \text{numberof}(p) > 0 \wedge \text{numberof}(r) > 0$$

Every financial transaction entails a transfer of funds from one source to a destination.

Accountability Requirement 3: Every transaction has at least one payment, and one receipt voucher [AE04, CICA03]:

$$\forall tx \exists pv, rv : \text{Minimum}(tx, pv) \wedge \text{Minimum}(tx, rv) \wedge \\ \text{numberof}(pv) > 0 \wedge \text{numberof}(rv) > 0$$

For example, in a transaction that entails the transfer of money from an employer to an employee, the pay stub that companies and organizations give to employees represents a payment voucher for the company, as well as a receipt voucher for the employee.

Accountability Requirement 4: Each payer and beneficiary is unique, and uniquely identified [AE04]:

$$\forall p \forall r : \text{Unique}(p, pid) \wedge \text{Unique}(r, rid) \wedge \\ \text{Identified}(p, pid) \wedge \text{Identified}(r, rid)$$

Uniqueness is a very important factor that eliminates discrepancies and fraudulent situations in a financial transaction. An example of a uniquely identified beneficiary can be seen in an organization that employs two people with the same name in different departments with differing levels of compensation. In this situation, the organization can use an employee identification number to uniquely identify each of the employees. An example of a uniquely identified payer can be seen when the two employees with the same name prepare and send their respective tax information to the Canada Revenue Agency (CRA, the agency responsible for administering tax laws in Canada.) In this case, the CRA will be able to distinguish the two people by uniquely identifying them with their social insurance numbers.

Accountability Requirement 5: For every transaction, the date and the time are known and documented [AE04, CICA03, TOG98]:

$$\forall tx \exists date, time : \text{Known}(tx, date) \wedge \text{Known}(tx, time) \wedge \\ \text{Documented}(date) \wedge \text{Documented}(time)$$

Accountability Requirement 6: For every transaction, transaction details are known and documented [AE04, CICA03, TOG98]:

$$\forall tx \exists td : \text{Known}(tx, td) \wedge \text{Documented}(td)$$

Accountability Requirement 7: For transactions that require other agents (processors and authorizers), the processor and the authorizer are uniquely identified, and the date and time of their actions are known and documented [AE04, CICA03]:

$$\begin{aligned} \forall tx : \exists pr, au \implies & Unique(pr, prid) \wedge Unique(au, auid) \wedge \\ & Identified(pr, prid) \wedge Identified(au, auid) \wedge \\ & Known(ac, date) \wedge Known(ac, time) \wedge \\ & Documented(date) \wedge Documented(time) \end{aligned}$$

This requirement emphasizes the non-repudiation requirement of a financial transaction. An example is the case of a person that requires a large sum of cash advance from a credit card through a teller in a bank. The teller (processor of the transaction) processes the transaction; however, the teller requires an authorization from a supervisor before the transaction can be completed due to the value of money involved. The banking software that is used to carry out this transaction will log the user identification numbers of both the teller and the supervisor (acting here as an authorizer) as other agents that were involved in this particular transaction. The information about the users that is logged in this case is different from a situation where the person involved uses an automatic bank machine for the cash advance. In this situation, other agents are not involved in the completion of the transaction.

3.7.2 Security Requirements and Specifications

Security requirements provide a safety framework for a financial system. The process of ascertaining safety requires a good level of control through access and usage restrictions and authorizations. Security requirements and their formal specifications for the financial

audit system are expressed as follows:

Security Requirement 1: For all high-valued transactions, an additional control must be satisfied. For example, at least two agents (a processor, and an authorizer) must verify a high-value transaction [AE04, CICA03]:

$$\forall tx_H \exists ctr_A \exists pr, au : Verified(tx_H, pr, au)$$

Security Requirement 2: Audit logs must be protected from unauthorized access and use [AE04, CICA03, TOG98]:

$$\forall l : \exists \bar{a}, \bar{u} \implies Protected(l, \bar{a}) \wedge Protected(l, \bar{u})$$

Computer systems that contain audit information and logs are protected from unauthorized access and usage physically through the use of a safe and locked location and logically through the use of software as a means for establishing and enforcing access restrictions.

Security Requirement 3: Protection of pre-defined audit triggers from unauthorized access and use [AE04, CICA03, TOG98]:

$$\forall tg : \exists \bar{a}, \bar{u} \implies Protected(tg, \bar{a}) \wedge Protected(tg, \bar{u})$$

This requirement is critical to avoid a situation where an audit ‘red flag’ is fraudulently suppressed. An unauthorized access or usage of pre-defined audit triggers could result in a situation where a red flag (a caution that is programmed into the system) is suppressed. A suppressed red flag could subsequently be exploited by fraudsters to carry out fraudulent transactions.

Security Requirement 4: Protection of audit analysis from unauthorized access and use [AE04, CICA03, TOG98]:

$$\forall ana : \exists \bar{a}, \bar{u} \implies Protected(ana, \bar{a}) \wedge Protected(ana, \bar{u})$$

Unauthorized access or usage of audit analysis could result in a situation that suppresses the accurate analysis of a fraudulent transaction. A misleading audit analysis could result into poor decisions that could have a serious negative impact on the organization affected.

Security Requirement 5: Several failed attempts lead to service denial [AE04, TOG98]:

$$\forall f \forall u : (f > f_P) \implies Denied(u, sys_A) \wedge Denied(u, sys_U)$$

When the count of a user's consecutive failed signon attempts to the financial system reaches a maximum threshold number, the signon attempt is considered fraudulent. The user is denied access into the system until the signon identification is reset by a superior officer. This requirement prevents unauthorized access into the system. Also, user identification that consistently requires a reset could be a likely target for fraudsters and hackers.

Security Requirement 6: Segregation of duties related to audit log [AE04, CICA03, TOG98].

$$\forall u \forall d : Has(u, d) \implies Segregated(d)$$

Functions for audit log users, report generators, audit log analysts, and custodians of audit log (backup engineers, audit log database administrators) must be kept separate.

Security Requirement 7: Maintaining and protecting the log recording audit-log user activity [AE04, CICA03, TOG98]:

$$\forall u \forall av : Has(u, av) \wedge Log(av, l) \implies Protected(l, \bar{a}) \wedge Protected(l, \bar{u})$$

This is a crucial requirement for preventing inappropriate and fraudulent use of audit data. It also protects the results of audit analysis carried out by auditors. A detailed log of actions and activities of auditors makes the work of auditors subject to another level of scrutiny. This is especially relevant to the shareholders of an organization. It prevents fraudulent collaboration between management staff in an organization and the internal and external auditors. Inappropriate misrepresentation of financial records and auditor analysis is not possible when the actions of auditors are recorded.

Security Requirement 8: Idle users are automatically logged off [AE04]:

$$\forall u : (\exists time_I \wedge (time_I > time_O)) \implies Logoff(u)$$

This requirement prevents unauthorized access and usage of the financial system. When a user session is left unattended for a pre-defined period, the system logs off the user, preventing others from accessing the system.

3.7.3 Transaction Monitoring Requirements and Specifications

The process of providing audit information requires a mechanism for collecting data that is relevant for audit purposes. In this case, an audit logging system is required to create audit data from transaction data. An audit system requires a transparent view of financial transactions processes. Transaction monitoring requirements provide this transparency and are expressed as follows:

Transaction Monitoring Requirement 1: The system shall document the transaction details for all transactions [AE04, CICA03, TOG98]:

$$\forall tx \exists l, td : Log(td, l) \Leftrightarrow l' = l \wedge td$$

Transaction details include the initiator of the transaction, all users linked with the transaction, date and time stamps for all actions on the transaction, and the status of the transaction (complete, incomplete, successful, or fail.)

Transaction Monitoring Requirement 2: Transaction monitoring is transparent to users [AE04]:

$$\forall tx \exists l : Transparent(monitor(l, tx), u)$$

This requirement prevents the monitoring and logging of audit data from having any major impact on the efficiency of processing transactions in the financial system.

Transaction Monitoring Requirement 3: Users that are not audit log users cannot access or use audit logs [AE04, CICA03]:

$$\forall u_N \forall l : \exists logaccess, logusage \implies Protected(l, u_N) \wedge \\ Denied(u_N, logaccess) \wedge Denied(u_N, logusage)$$

3.7.4 Event Logging Requirements and Specifications

Event logging requirements describe details and properties of audit data that are being captured and logged into an audit log. The safety of audit data requires a process that

makes it unreadable by unauthorized persons. Also, the safety of user's identity and password is crucial, and should be shielded from every other users of the financial system and the audit log. These event logging requirements and their formal specifications are presented as follows:

Event Logging Requirement 1: The event log captures all information about activities related to the audit log [AE04, CICA03]:

$$\forall av \exists l : \text{Log}(av, l) \implies l' = l^{\wedge} av$$

This includes information on who accessed the audit data, when this access was granted, what data was accessed and where the access occurred. Capturing this information is essential to guard against the repudiation of a financial transaction using an audit log. This requirement aids the investigation of suspicious transactions and satisfies the non-repudiation requirement of a financial transaction.

Event Logging Requirement 2: The audit log is capable of storing a huge volume of data [AE04]:

$$\forall data_H \exists l : \text{Log}(data_H, l)$$

Event Logging Requirement 3: The encryption module encrypts all data in the log [AE04, CICA03]:

$$\forall data : \text{Log}(data, l) \implies \exists e_M : \text{Encrypted}(data, e_M)$$

Event Logging Requirement 4: The de-encryption module de-encrypts audit data that are selected for analysis [AE04, CICA03]:

$$\forall data \forall u_A : Analyzed(data, u_A) \implies \exists d_M : Decrypted(data, d_M)$$

Event Logging Requirement 5: Event logs do not log user passwords [AE04]:

$$\forall l \forall pwd : Log(\neg pwd, l) \implies l' = l$$

This requirement prevents audit log users from having unauthorized access to passwords of other users.

3.7.5 Reporting Requirements and Specifications

The result of an audit should be reported in a usable manner. Providing a large amount of audit data without narrowing it into a small, useful report could result in insufficient audit work. Audit reporting requirements provide a focus for audit work. These audit reporting requirements and their formal specifications are expressed as follows:

Reporting Requirement 1: The audit report includes user identification [AE04]:

$$\forall rpt \forall uid : Report(rpt, uid)$$

This requirement makes a quick audit of a suspicious user possible, allowing auditors to focus on the actions and activities of a particular user of the financial system over a specific period.

Reporting Requirement 2: Audit report includes details of sequentially evolving transaction activities [AE04, CICA03]:

$$\forall rpt \forall td : Report(rpt, td)$$

This requirement provides audit information that is based on a particular suspicious transaction. The transaction details, users, agents, and all other facts associated with the transaction are provided to permit an accurate analysis of the transaction by auditors.

Reporting Requirement 3: The audit report includes successful and failed signon attempts [AE04]:

$$\forall rpt : \exists s_S, s_F \implies Report(rpt, s_S) \wedge Report(rpt, s_F)$$

This requirement provides audit information for user signon attempts. An example of the usage of this requirement is the investigation of why a user began to consistently fail signon attempts. This is a probable security issue that could reveal an attempt by hackers to gain unauthorized access into the financial system through the fraudulent use of an existing user identification.

Reporting Requirement 4: The audit report includes details of both complete and incomplete transactions [AE04]:

$$\forall rpt : \exists td_C, td_I \implies Report(rpt, td_C) \wedge Report(rpt, td_I)$$

This requirement provides audit information for both completed and incomplete transactions. This is especially useful for audit analysis of incomplete transactions. Consider the case of a merchant who provides an online (e-commerce) shopping service. Several incomplete transactions in the audit analysis warrant further investigation. This investi-

gation can reveal a malfunctioning module in the checkout system. It may indicate an attempt by credit card fraudsters to defraud the organization and the credit card owner.

3.8 Consistency Checking

According to Frost [Fro86], the validity of any proposition that a formal system accepts depends on the result obtained from a “Consistency Checking” exercise performed on the proposition. The consistency checking looks for a contradiction between the set of facts in the system and the proposition being verified. If there is a contradiction between a fact and the proposition, then the proposition cannot be accepted because it is inconsistent with an already established fact. Otherwise, the proposition is acceptable by the system. This concept is referred to as *decidability*. For example, according to Frost [Fro86], the following set of assertions (S2) is inconsistent:

$$S2 = \{\text{Jan is NOT a woman,} \\ \text{Jan is a woman AND Jan is tall}\}$$

This is because,

- (a) Jan is NOT a woman is equivalent to
Jan is a woman = FALSE
- (b) Jan is a woman AND Jan is tall is equivalent to
{Jan is a woman = TRUE,
Jan is tall = TRUE}

The Predicate logic specifications in this thesis will aid the financial audit system herein. It will perform a consistency checking exercise on any proposition brought to the system. In order to ascertain the manner in which the set of rules in the requirement specification

will carry out a consistency check in the system, I provide some propositions and their consistency checks as follows:

- Proposition 1: A transfer request without an initiator. Formally, we say,

$$\exists tx : \neg \exists i \implies (numberof(i) = 0)$$

Proposition 1 contradicts Accountability Requirement 1. Therefore, it resolves to *FALSE* because it is inconsistent with Accountability Requirement 1.

- Proposition 2: A transaction that does not state a *Receiver* for its funds. Formally, we say,

$$\exists tx : \neg \exists r \implies (numberof(r) = 0)$$

Proposition 2 contradicts Accountability Requirement 2. Therefore, it resolves to *FALSE* because it is inconsistent with Accountability Requirement 2.

- Proposition 3: A transaction has no *Payment Voucher*. Formally, we say,

$$\exists tx : \neg \exists pv \implies (numberof(pv) = 0)$$

Proposition 3 contradicts Accountability Requirement 3. Therefore, it resolves to *FALSE* because it is inconsistent with Accountability Requirement 3.

- Proposition 4: A transaction cannot uniquely identify its *Payer*. Formally, we say,

$$\exists tx : \neg \exists_1 p \implies \neg Unique(\neg p, pid)$$

Proposition 4 contradicts Accountability Requirement 4. Therefore, it resolves to *FALSE* because it is inconsistent with Accountability Requirement 4.

- Proposition 5: The date and time of a transaction are unknown and undocumented.

Formally, we say,

$$\begin{aligned} \exists tx : \neg Known(tx, date) \wedge \neg Known(tx, time) \wedge \\ \neg Documented(date) \wedge \neg Documented(time) \end{aligned}$$

Proposition 5 contradicts Accountability Requirement 5. Therefore, it resolves to *FALSE* because it is inconsistent with Accountability Requirement 5.

- Proposition 6: An audit system cannot provide a report based on *User ID*. Formally, we say,

$$\exists rpt, uid : \neg Report(rpt, uid)$$

Proposition 6 contradicts Reporting Requirement 1. Therefore, it resolves to *FALSE* because it is inconsistent with Reporting Requirement 1.

- Proposition 7: An audit system report cannot provide a sequentially evolving details of a transaction. Formally, we say,

$$\exists rpt \exists td : \neg Report(rpt, td)$$

Proposition 7 contradicts Reporting Requirement 2. Therefore, it resolves to *FALSE* because it is inconsistent with Reporting Requirement 2.

- Proposition 8: An audit system cannot provide a report of either successful or failed transactions. Formally, we say,

$$\exists rpt : \neg Report(rpt, s_S) \vee \neg Report(rpt, s_F)$$

Proposition 8 contradicts Reporting Requirement 3. Therefore, it resolves to *FALSE* because it is inconsistent with Reporting Requirement 3.

- Proposition 9: An audit system cannot provide a report of either completed or incomplete transactions. Formally, we say,

$$\exists rpt : \neg Report(rpt, td_C) \vee \neg Report(rpt, td_I)$$

Proposition 9 contradicts Reporting Requirement 4. Therefore, it resolves to *FALSE* because it is inconsistent with Reporting Requirement 4.

3.9 Conclusions

In this chapter, I provided formal specifications and a model for a financial audit system. An architecture, a classification of entities and concepts, and the requirements of a financial audit system were presented. An overview of the formal model was presented using static, use case, state machine, activity, and interaction views of the UML as captured and represented by UML diagrams.

The chapter presented both informal and formal definitions for all terms used in the specifications. Formal specifications for the requirements of a financial audit system using the Predicate logic specification language were outlined. These specifications were further clarified with examples that outline the use of the formal specifications described in the chapter. I carried out a *consistency checking* exercise on some propositions. I leveraged the *decidability* property of Predicate logic to determine their truth values and whether they are acceptable by the financial audit system or not. I also explained how the formal specifications will determine inconsistencies in these propositions.

In the next chapter, I provide an implementation of a financial audit system prototype. This prototype demonstrates a strategy for collecting audit data (based on formal

specifications in this chapter) in a financial system.

Chapter 4

Financial Audit System Prototype Implementation

In the previous chapter, I modelled and formally specified the non-technical and technical requirements for a financial audit system. In this chapter, I demonstrate the successful implementation of a prototype of a financial audit system.

I start by citing a strategy for collecting financial audit data. Using this strategy, the prototype implementation is shown to satisfy the requirements for accountability, security, transaction monitoring, event logging and reporting established for the auditing system.

4.1 A Strategy for Collecting Financial Audit Data

In this section, I provide an implementation of a financial system prototype that incorporates the audit data collection techniques outlined in this thesis. The prototype performs the basic activities that are necessary in a financial system, and incorporates a logging

sub-system that records relevant audit data in a Microsoft® SQL Server™ 2000 database. Visual Basic .NET framework [Bal02] was used to provide the user interface implementation.

The prototype implementation for the financial system and corresponding audit sub-system is based on the following assumptions:

- A financial transaction is the transfer of funds (i.e., money) from entity *A* to entity *B* through a financial system.
- The following actors / entities exist in the financial system: *payer*, *receiver / beneficiary*, financial system *operators – processor* and *authorizer*, and *auditors*.
- The prototype financial system is able to process financial transactions (*make / receive payments*) and *audit transactions*.
- The financial system has operational capabilities to allow user creation (for both audit and non-audit users), uniquely identifies users, assigns functional rights to users, creates dummy passwords for users, modifies functional rights for users, deletes users from the system, allows users to change their passwords, and facilitates the processing of financial transactions.
- The audit aspect of the system has capabilities to collect audit data and extract audit information in the following categories:
 1. *User audit*. The ability to provide audit information that describes the creation of users in the financial system, assignment of functional rights to users, modification of functional rights for users, and deletion of users from the system.

2. *Periodic transaction audit.* The ability to provide financial audit information that describes a sequence of transactions over a specified period.
3. *Successful sign-on audit.* The ability to provide audit information that records the date and time that users successfully sign-on into the financial system.
4. *Failed sign-on audit.* The ability to provide audit information that describes users that could not sign-on into the financial system. This information can be used to detect attempts by unauthorized users to gain access to the financial system.
5. *Completed transaction audit.* The ability to provide audit information that describes transactions that are successfully completed.
6. *Incomplete transaction audit.* The ability to provide audit information that describes transactions that could not be successfully completed. This information can be used to detect transactions that encounter problems.
7. *Audit of the actions of auditors.* The ability to provide audit information that describes the activities of auditors in a financial system. It is advisable to have a monitoring system that provides audit trails for actions of auditors.
8. *Audit of user access and activities.* The ability to provide audit information that describes the sign-on, sign-off, and usage of the financial system by its users. This can be used to achieve a good level of accountability for individual users, and it can prevent repudiation of financial transactions in cases of disputes.

The prototype implementation of the financial audit system provides a log of all relevant actions, and activities in the financial system. Also, the audit system provides detailed

information retrieval and reporting options to ascertain the correctness of the log system. This electronic logging system can be used to provide evidential documents in the financial audit system.

4.2 Implementation Details of the Prototype Application

An experimental evaluation of the financial audit data collection strategy was done. Some of the queries and results of the queries are presented in Figures 4.3, 4.4, and 4.5. This prototype implementation of a financial audit system is built upon the formal specifications of the requirements for a financial audit system presented in Section 3.7.

Figures 4.1 to 4.5 present screen shots from the prototype implementation of the financial audit system in this thesis. Figure 4.1 shows a screen shot of a typical financial transaction. This sample transaction indicates that on February 08, 2004 a payer (*Jim Black*) purchased a used car valued at \$3,500.00 from a receiver (*Ken Cook*). The corresponding payment and receipt vouchers in Figure 4.2 indicate the time (*16:57:16*) and date (*08/02/2004*) of the transaction, as well as the status of the transaction (both the payment and the receipt of funds were successfully carried out.)

The sections that follow provide examples and explanations of the prototype implementation as it relates to the audit requirements described in Section 3.7.

4.2.1 Implementation of Accountability Requirements in Section 3.7.1

The sample transaction shown in Figure 4.1 satisfies the Accountability Requirements stated in 3.7.1 as follows:

1. Accountability Requirement 1: The payer (Jim Black) is the unique initiator.
2. Accountability Requirement 2: Jim Black is the payer, and Ken Cook is the receiver.
3. Accountability Requirement 3: Figure 4.2 shows the payment and the receipt vouchers.
4. Accountability Requirement 4: The payer (Jim Black) is uniquely identified by his name, and the receiver (Ken Cook) is equally identified uniquely by his name.
5. Accountability Requirement 5: The date (“08/02/2004”), and the time (“16:57:16”) of the transaction are known and documented in the system.
6. Accountability Requirement 6: The transaction details, i.e., the value of 3,500.00, and the description of the transaction (“Purchase – Used car”) are known and documented in the system.
7. Accountability Requirement 7: This does not apply since no agent was involved in the transaction.

4.2.2 Implementation of Security Requirements in Section 3.7.2

Figure 4.3 shows two screen shots of information related to auditor actions. Figure 4.3(a) shows an audit query criteria entry screen. This screen allows a stakeholder (for example, shareholders of a company or a business owner) to enter start and end dates for auditor monitoring. Figure 4.3(b) shows the result of a sample query. It lists the auditor name, date and time they accessed the financial system, the activity they carried out, a description for the audit and the status of each audit. This satisfies the Security Requirement 7

outlined in Section 3.7.2, because details of auditor activities are maintained in the audit log.

Although the Auditors' Actions Audit Report in this case is from January 01, 2004 to February 07, 2004, this screen shot shows a part of the audit information and activities of only one audit user (named "Auditor") on February 07, 2004. The audit report indicates that on February 07, 2004, audit user *Auditor* performed a successful "Fail Sign-On Audit" for the period from January 01, 2004 to February 07, 2004, a "Periodic Audit" for the period from January 01, 2004 to February 07, 2004, an "Auditors Audit" for the period from January 01, 2004 to February 07, 2004, a "User Audit" on the user named "*auditor*" (three times), another "Periodic Audit" for the period from January 01, 2004 to February 07, 2004, another "Fail Sign-On Audit" for the period from January 01, 2004 to February 07, 2004 (two times), and a "Complete Tran Audit" (a completed transactions audit) for the period from January 01, 2004 to February 07, 2004. The details in this report allows the identification of trends which can be of use in determining what activities are being performed by auditors in a financial audit system.

4.2.3 Implementation of Transaction Monitoring Requirements in Section 3.7.3

Figure 4.4 shows the prototype implementation of the Transaction Monitoring Requirement 1 outlined in Section 3.7.3. The system documents transaction details by providing the name of users, date of transaction, the payer, the receiver, the value of transaction, the status of the transaction, and information for the authorizers involved in the transaction.

The Periodical Audit reporting screen shot shown in Figure 4.4(a) shows an entry of

audit query criteria. This criteria provides audit information over a period of time, for example, from January 01, 2004 to February 07, 2004. Figure 4.4(b) shows the result of the query. The Periodical Audit report provides detailed audit data which includes signon failure, successful signon, successful transfer, exit, failed transfer, and so on.

In addition, the report shows transactions that the system could not accept, but which can be useful for extracting a trend in the system. For example, an attempt to transfer a negative value would be shown.

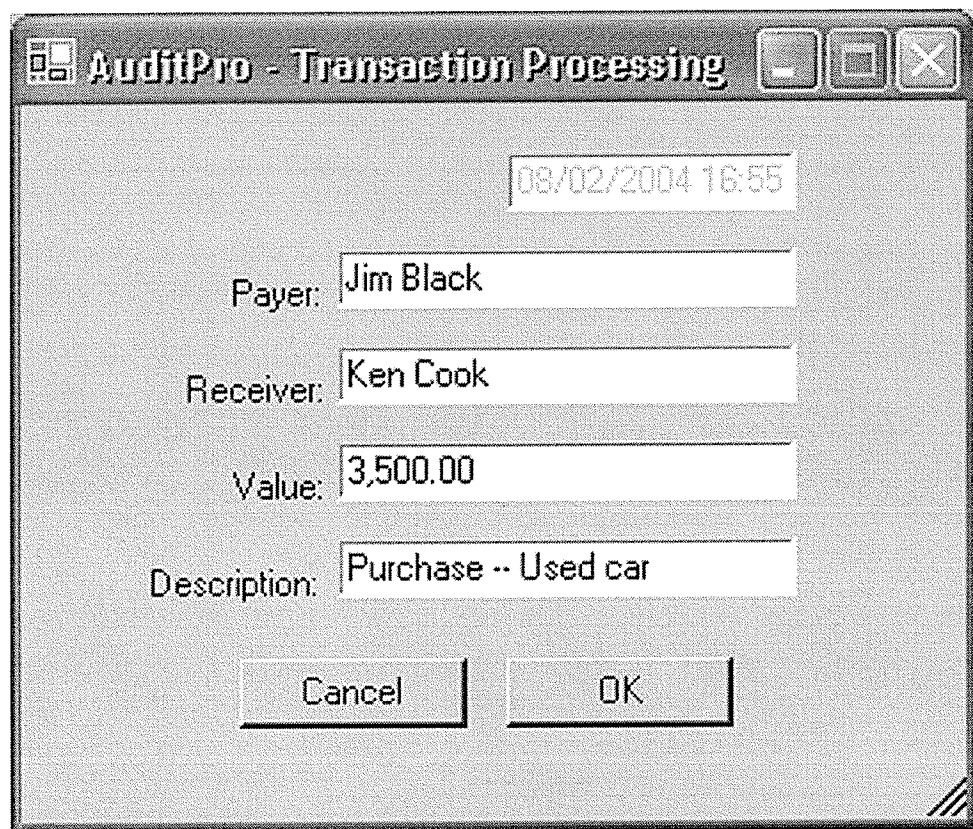
4.2.4 Implementation of Event Logging Requirements in Section 3.7.4

Figures 4.3, 4.4, and 4.5 provide audit information that satisfies the Event Logging Requirement 1 outlined in Section 3.7.4. Each figure includes information on who accessed the audit data, when this access was granted, what data was accessed and where the access occurred. In addition, Figures 4.3 and 4.5 indicate the activity and description of transactions and Figure 4.4 indicates the transaction details (for example, payer, receiver and value) for each transaction.

Figure 4.5 shows activities that relate to an attempt to, or the successful creation, modification, or deletion of users or their functional rights in the system. In Figure 4.5(b), we see that on February 04, 2004, user *Authorizer* attempted to delete another user *mabisi* from the system. The deletion attempt failed twice. In the same figure on February 06, 2004, the user *Authorizer* successfully deleted user *JohnDoe* from the system. Also, the password for user *Authorizer* was successfully changed on February 06, 2004.

4.2.5 Implementation of Reporting Requirements in Section 3.7.5

Figures 4.3, 4.4, and 4.5 provide audit information that satisfy the Reporting Requirements outlined in Section 3.7.5. The Reporting Requirement 1 is satisfied in Figures 4.3, 4.4, and 4.5 by the *Name*. Reporting Requirements 2 and 3 are satisfied in Figure 4.4. Finally, Reporting Requirement 4 is satisfied in Figures 4.3, 4.4, and 4.5 where both complete and incomplete transactions are shown.



The screenshot shows a window titled "AuditPro - Transaction Processing". In the top right corner of the window, there is a timestamp "08/02/2004 16:55". Below the timestamp, there are four labeled text input fields: "Payer:" with the value "Jim Black", "Receiver:" with the value "Ken Cook", "Value:" with the value "3,500.00", and "Description:" with the value "Purchase -- Used car". At the bottom of the window, there are two buttons: "Cancel" and "OK".

Field	Value
Payer:	Jim Black
Receiver:	Ken Cook
Value:	3,500.00
Description:	Purchase -- Used car

Figure 4.1: Sample Transaction Processing Screen Shot

The screenshot shows a window titled "AuditPro - Payment & Receipt Vouchers". The window contains two sections separated by a horizontal line. The top section is titled "Payment Voucher for Jim Black" and contains the following information: Receiver: Ken Cook, Value: 3,500.00, Description: Purchase -- Used car, and Status: Transfer Successful. The bottom section is titled "Receipt Voucher for Ken Cook" and contains the following information: Payer: Jim Black, Value: 3,500.00, Description: Purchase -- Used car, and Status: Transfer Successful. The date and time "08/02/2004 16:57:16" are displayed in the top right corner of the window and below the horizontal line. A "Close" button is located in the bottom right corner of the window.

AuditPro - Payment & Receipt Vouchers

08/02/2004 16:57:16

Payment Voucher for Jim Black

Receiver: Ken Cook
Value: 3,500.00
Description: Purchase -- Used car
Status: Transfer Successful

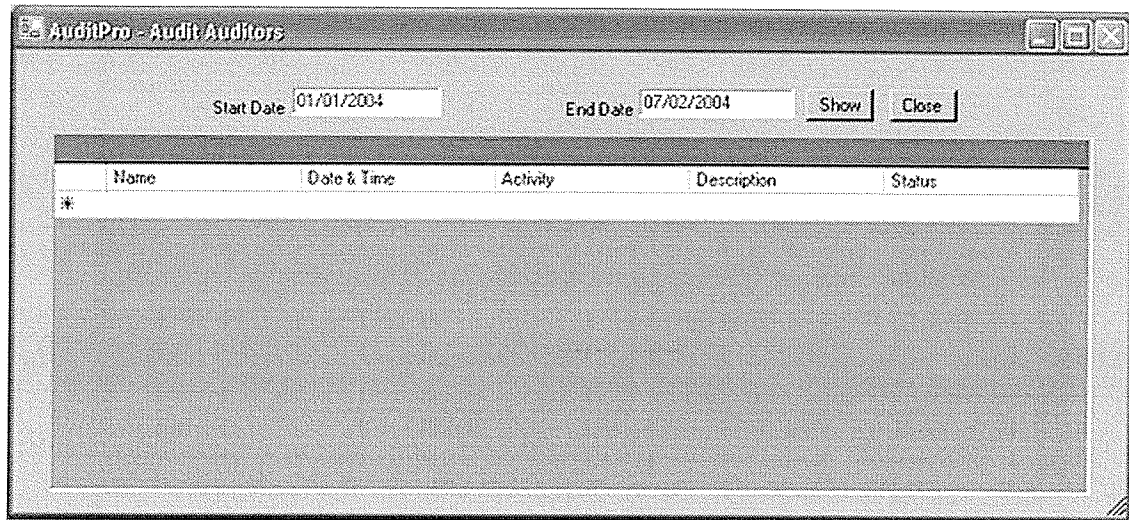
08/02/2004 16:57:16

Receipt Voucher for Ken Cook

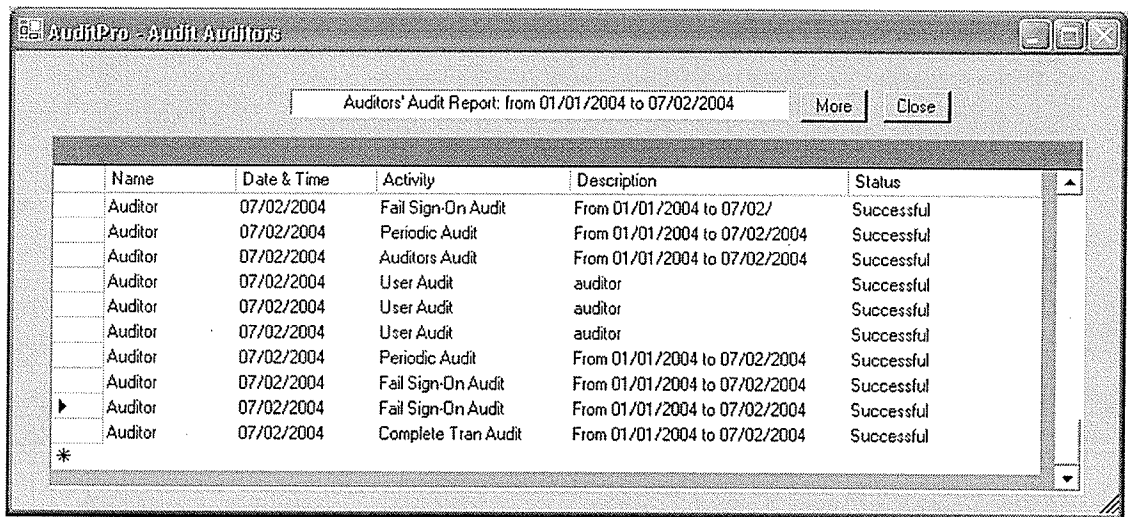
Payer: Jim Black
Value: 3,500.00
Description: Purchase -- Used car
Status: Transfer Successful

Close

Figure 4.2: Payment and Receipt Vouchers for a Sample Financial Transaction



(a) Auditors' Actions Audit Report (Query Input Screen)



(b) Auditors' Actions Audit Report (Result Output Screen)

Figure 4.3: Auditors' Actions Audit Reporting Screenshots

(a) Periodical Audit Report (Query Input Screen)

Name	Date & Time	Payer	Receiver	Value	Status	Authorizer
processor	04/02/2004	[null]	[null]	[null]	Sign-On Failure	[null]
processor	04/02/2004	[null]	[null]	[null]	Sign-On Successful	[null]
Processor	04/02/2004	Olis	David	2,500.00	Transfer Successful	[null]
Processor	04/02/2004	[null]	[null]	[null]	Exit	[null]
authorizer	04/02/2004	[null]	[null]	[null]	Sign-On Successful	[null]
authorizer	04/02/2004	[null]	[null]	[null]	Exit	[null]
auditor	04/02/2004	[null]	[null]	[null]	Sign-On Successful	[null]
Auditor	04/02/2004	[null]	[null]	[null]	Exit	[null]
processor	04/02/2004	[null]	[null]	[null]	Sign-On Successful	[null]
Processor	04/02/2004	China	Sylvanus	0.25	Transfer Successful	[null]
Processor	04/02/2004	China	Sylvanus	23	Transfer Failed	[null]
Processor	04/02/2004	China	Sylvanus	-50	Transfer Failed	[null]
Processor	04/02/2004	[null]	[null]	[null]	Transfer Failed	[null]
processor	04/02/2004	[null]	[null]	[null]	Exit	[null]
authorizer	04/02/2004	[null]	[null]	[null]	Sign-On Successful	[null]

(b) Periodical Audit Report (Result Output Screen)

Figure 4.4: Periodical Audit Reporting

(a) User Access Audit Report (Query Input Screen)

Name	Date & Time	Activity	Description	Status
Authorizer	04/02/2004	Delete User	mabiti	Failure
Authorizer	04/02/2004	Delete User	mabiti	Failure
Authorizer	06/02/2004	Delete User	JohnDoe	Successful
Authorizer	06/02/2004	Password Change	Authorizer	Successful
Authorizer	04/02/2004	Delete User	mabiti	Failure

(b) User Access Audit Report (Result Output Screen)

Figure 4.5: User Access Audit Reporting

AuditPro - User Name Audit Report

Enter User Name:

Name	Date & Time	Payer	Receiver	Value	Status	Authorizer
*						

(a) User Name Audit Report (Query Input Screen)

AuditPro - User Name Audit Report

User Audit Report for auditor

Name	Date & Time	Payer	Receiver	Value	Status	Authorizer
Auditor	04/02/2004	(null)	(null)	(null)	Sign-On Successful	(null)
Auditor	04/02/2004	(null)	(null)	(null)	Exit	(null)
auditor	06/02/2004	(null)	(null)	(null)	Sign-On Successful	(null)
Auditor	06/02/2004	(null)	(null)	(null)	Exit	(null)
auditor	06/02/2004	(null)	(null)	(null)	Sign-On Successful	(null)
Auditor	06/02/2004	(null)	(null)	(null)	Exit	(null)
auditor	06/02/2004	(null)	(null)	(null)	Sign-On Successful	(null)
Auditor	06/02/2004	(null)	(null)	(null)	Exit	(null)
auditor	06/02/2004	(null)	(null)	(null)	Sign-On Successful	(null)
auditor	06/02/2004	(null)	(null)	(null)	Sign-On Successful	(null)
auditor	06/02/2004	(null)	(null)	(null)	Sign-On Successful	(null)
auditor	06/02/2004	(null)	(null)	(null)	Sign-On Successful	(null)

(b) User Name Audit Report (Result Output Screen)

Figure 4.6: User Name Audit Reporting

4.3 Relationship of the Prototype Implementation with Formal Specifications in Section 3.7

So far, I have provided a formal specification of the requirements of a financial audit system and a prototype implementation of the system. Major highlights in this thesis are related to how the system can assist auditors perform audit functions and how stakeholders in a financial system (for example, shareholders of a company) can utilize the system to their advantage.

A descriptive usage of the system as it relates to auditors is expressed as follows:

1. The system provides a means that assist auditors to extract audit data independently and solely for audit purposes. This satisfies Transaction Monitoring Requirement 1.
2. The periodical audit report as shown in Figure 4.4(b) provides useful audit information as they are sequentially processed. Any suspicious transaction detail can be further investigated in detail. This report is an implementation of Reporting Requirement 2 and Accountability Requirements 5 and 6.
3. Audit report based on user name as shown in Figure 4.6(b) assists in providing detailed investigation on transactions that relate to a particular transaction under audit scrutiny. This report is an implementation of Reporting Requirement 1.
4. User access audit report as shown in Figure 4.5(b) provides user actions that relate to the creation, modification and deletion of users. An auditor can review this

report to ascertain the assigned duties of a financial user at a given time under investigation. This report also provides information whether a user right was changed or manipulated to carry out an unauthorized transaction that would not have been possible with the authorized user rights and functions in the financial system.

Usage of the system as it applies to stakeholders in a financial system is expressed as follows:

1. A shareholder can utilize the audit report in Figure 4.3(b) to review all successful audit exercises within a certain period. In this case, any audit report that is omitted can be flagged for subsequent auditing. This satisfies Reporting Requirement 3 and Security Requirement 7.
2. The report in Figure 4.3(b) provides a description of each of the audit exercises shown. A review of this description by a stakeholder may reveal a concealment in the description of each audit exercise. For example, consider a situation where auditors were supposed to carry out an audit exercise starting from January 01, 2004 to February 10, 2004. Instead, they performed the audit between January 01, 2004 to February 07, 2004. A stakeholder will be able to flag this omission because it might be a fraudulent non-disclosure or misrepresentation of facts. This satisfies Reporting Requirement 4 and Event Logging Requirement 1.
3. The audit report based on user name as shown in Figure 4.6(b) can be used to provide detailed information about the financial audit system. This information can relate to the activity of a suspicious user in the financial system. This satisfies Reporting Requirement 1 and Transaction Monitoring Requirement 1.

4.4 Conclusions

In this chapter I have provided a strategy for collecting audit information in a financial system. I also demonstrate a prototype implementation of a financial audit system based on the formal specifications and requirements described in Section 3.7. Screen shots of the prototype implementation were presented that demonstrate satisfaction of the accountability, security, transaction monitoring, event logging and reporting requirements outlined in Section 3.7.

Based on the results obtained (as shown in Figures 4.3, 4.4, 4.5, and 4.6), the audit data collection approach explored in this thesis provides audit information from the audit log. The prototype implementation demonstrates the use of formal methods in the evolution and implementation of software systems.

In the next chapter, I conclude this thesis and provide a road map for future work.

Chapter 5

Conclusions and Future Work

This chapter concludes the thesis, provides a summary of its contributions, and describes the limitations encountered during the implementation of its concepts. A road map of future work is also presented.

5.1 Conclusions

This thesis presented the use of formal specifications in the design and implementation of a financial audit system. Chapter 1 discussed the benefits and problems associated with financial audit systems, the suitability of formal methods in the domain, a theoretical foundation for financial auditing, and the need for electronic evidential documents. Chapter 2 presented related work in the area of financial audit systems, as well as some other background information about UML and the Predicate logic specification language. Chapter 3 presented an architecture, a classification, a model description, a set of requirements for a financial audit system, and outlined the use of formal methods in modeling a

financial audit system with UML and the Predicate logic specifications language. Chapter 4 provided a demonstration of a financial audit data collection strategy based on the formal specifications of financial audit system requirements described in Chapter 3. Screen shots and explanations of the implementation details of the system were presented.

5.2 Summary of Contributions

This thesis explored the financial system and financial audit system problem domains. It leveraged two broad research areas to exploit the modeling capabilities of Computer Science techniques using (i.) the principles of software engineering, and (ii.) formal methods that are generally suitable for providing semantic precision in models.

The contributions of this thesis include:

- providing an architecture and a classification of a financial audit system,
- providing visual descriptions and a model of a financial audit system with UML diagrams,
- providing formal specifications of a financial audit system requirements with Predicate logic, and
- implementing a prototype of a financial audit system based on formal specification and modeling techniques, suitable for audit purposes through the collection of financial audit data.

5.3 Limitation of this Thesis

This thesis did not implement all the financial audit system requirements previously stated. However, the implementation herein covers the main goal of this thesis. The emphasis is on providing a record of transaction details as it relates to financial auditing. This record should assist auditors in performing financial audit functions. Also, the record should assist stakeholders to review the work of their auditors.

Due to the sensitive nature of real-life financial data, the synthetic data in the audit database was not collected from a real-life financial system database. Also, the volume of transactions that were captured, and subsequently reported was minimal. Although this does not have any impact on the overall and specific goals of this thesis, it is of note that a significant increase in the size of the audit log could impact on the response time it takes the audit system to extract audit information.

5.4 Future Work

The use of financial systems is universal. This thesis focused on one aspect of financial systems (mainly auditing.) The restricted scope of this thesis makes further work in this area necessary. The following section provides brief descriptions of areas that require further research.

5.4.1 Knowledge-based financial audit system

It is desirable that 'non-technical' stakeholders in the financial system can have additional insights into financial audit work. This will encourage an audit of the activities

of auditors by business owners (shareholders). There is a possibility that an ontology-driven knowledge-based system can be coupled into a financial audit system, with the knowledge base providing implicit / hidden knowledge in a 'non-technical' manner. It is hoped that current research in knowledge engineering (specifically, the PowerLoomTM [CMR⁺99] knowledge representation environment) can assist in this regard. In addition, if the knowledge base is ontology-driven, it is equally hoped that the knowledge base will be suitable for heterogeneous financial audit systems. Finally, the addition of a natural language processing module to the knowledge base would be expected to facilitate easier natural language interaction of 'non-technical' users with the knowledge base.

5.4.2 Formal verification of the dynamic (temporal) characterization of a financial audit system

Specifying the temporal aspects of a financial audit system is essential. Due to the changing nature of states as a result of several actions happening concurrently, it is important to know that the specifications of dynamic properties are not conflicting, and that they do not result into a continuous loop that can make the system reach an unstable state. It is hoped that Temporal Logic of Actions (TLA) [Lam94], and its specification language (TLA+) [Lam02] can provide a formal specification for the temporal aspects of a financial audit system (as described with Statecharts in this thesis). The TLC [YML99] could then be used to perform model checking on the TLA specifications. Model checking the specifications will ensure that all the various states are generated, and all paths reaching and exiting each state are calculated. This would allow all conflicts among the states in the system to be revealed and subsequently corrected.

5.5 Conclusions

I have provided a formal-based financial audit system using the UML modeling language and Predicate logic formal specifications language. I also discussed how the formal specifications can be used to carry out a consistency check on propositions meant for the system. It is desirable to know what extent this method will suit a relatively large, and real-life database of a financial audit system. It is my hope that the approach outlined in this thesis will be utilized by financial auditors and other stakeholders to ensure financial systems maintain integrity in the operations they perform and the information they provide.

Bibliography

- [AE04] John A. Akinyemi and Sylvanus A. Ehikioya. A Predicate Logic Foundation for Financial Audit Systems. In *8th IASTED International Conference on Software Engineering and Applications (SEA 2004)*, pages 339–344, Cambridge, MA, USA, November 2004.
- [AEOA04] John A. Akinyemi, Sylvanus A. Ehikioya, Femi G. Olumofin, and Chima Adiele. An Ontology and Knowledge Representation of a Financial Audit System. In *IASTED International Conference on Knowledge Sharing and Collaborative Engineering (KSCE 2004)*, pages 207–212, St. Thomas, Virgin Islands, USA, November 2004.
- [ALLS00] Alvin J. Arens, James K. Loebbecke, W. Morley Lemon, and Ingrid B. Spletstoesser. *Auditing and Other Assurance Services*. Prentice Hall, 8th edition, 2000.
- [AAA73] American Accounting Association. *A Statement of Basic Auditing Concepts*. American Accounting Association, Sarasota, FL, USA, 1973.
- [Bal02] Francesco Balena. *Programming Microsoft Visual Basic .NET*. Microsoft Press, 2002.
- [CMR⁺99] H. Chalupsky, R. McGregor, T. Russ, D. Moriarty, and E. Melz. PowerLoomTM Knowledge Representation System. *Technical Report*, University of Southern California, 1999. <http://www.isi.edu/isd/LOOM/PowerLoom>.
- [Dal05] Chetan Dalal. Foiled by Nanoscience. *The Institute of Internal Auditors*, 8(1), April 2005. <http://www.theiia.org/itaudit/>.
- [DD97] C. J. Date and Hugh Darwen. *A Guide to SQL Standard, 4th Edition*. Addison-Wesley, 1997.

- [Dut02] M. Dutra. Ontologies for Web Services. *Technical Report*, Object Management Group, 2002. <http://www.omg.org>.
- [Fro86] Richard A Frost. *Introduction to Knowledge Base Systems*. Macmillan Publishing Co., Inc., Indianapolis, IN, USA, 1986.
- [HR02] H. Hamburger and D. Richards. *Logic Language Models for Computer Science*. Prentice Hall International, 2002.
- [Koc79] Harvey S. Koch. On-Line Computer Auditing. In *ACM/CSC-ER Proceedings of the 1979 Annual International Conference*, page 191, 1979.
- [Kos04] Eija Koskivaara. Artificial Neural Networks in Analytical Review Procedures. *Managerial Auditing Journal*, 19(2), 2004.
- [LAL87] W. Morley Lemon, Alvin A. Arens, and James K. Loabbecke. *Auditing: An Integrated Approach*. Prentice Hall Canada Inc., 4th edition, 1987.
- [Lam94] Leslie Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.
- [Lam02] Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.
- [LS98] George F. Luger and William A. Stubblefield. *Artificial Intelligence — Structures and Strategies for Complex Problem Solving*. Addison-Wesley, 3rd edition, 1998.
- [MBB⁺03] Brahim Medjahed, Boualem Benatallah, Athman Bouguettayal, Anne H. H. Ngu, and Ahmed K. Elmagarmid. Business-to-Business Interactions: Issues and Enabling Technologies. *The International Journal on Very Large Data Bases (VLDB)*, 12(1):59–85, May 2003.
- [Mer03] Rebecca T. Mercuri. On Auditing Audit Trails. *Communications of the ACM Journal*, 46(1):17–20, 2003.
- [CICA03] The Canadian Institute of Chartered Accountants. *Electronic Audit Evidence*. The Canadian Institute of Chartered Accountants, 2003.
- [Oes02] Bernd Oestereich. *Developing Software with UML — Object-Oriented Analysis and Design in Practice*. Addison-Wesley, 2nd edition, 2002.
- [RES01] Zabihollah Rezaee, Rick Elam, and Ahmad Sharbatoghlie. Continuous Auditing: The Audit of the Future. *Managerial Accounting Journal*, 16(3):150–158, March 2001.

- [Rez04] Zabihollah Rezaee. Restoring Public Trust in the Accounting Profession by Developing Anti-Fraud Education, Programs, and Auditing. *Managerial Auditing Journal*, 19(1), 2004.
- [RJB99] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.
- [RS84] Donald E. Ricketts and Horton L. Sorkin. *Quantitative Techniques for Internal Auditing*. Institute of Internal Auditors, Inc., Research Report Number 27, 1984.
- [SA66] R. M. Skinner and R. J. Anderson. *Analytical Auditing*. Sir Isaac Pitman (Canada) Limited, 1966.
- [TH83] C. William Thomas and Emerson O. Henke. *Auditing: Theory and Practice*. Kent Publishing Company, 1983.
- [vEK76] Maarten H. van Emden and Robert A. Kowalski. The Semantics of Predicate Logic as a Programming Language. *Communications of the ACM Journal*, 23(4):733–742, 1976.
- [Wil83] H. J. Will. ACL: A Language Specific for Auditors. *Communications of the ACM Journal*, 26(5):356–361, 1983.
- [TOG98] Distributed Audit Service (XDAS). Preliminary Specification. *Technical Report* ISBN: 1-85912-139-Document Number: P441, The Open Group, January 1998. www.opengroup.org/publications/catalog/p441.htm. Accessed on October 07, 2003.
- [YML99] Yuan Yu, Panagiotis Manolios, and Leslie Lamport. Model Checking TLA + Specifications. In *Conference on Correct Hardware Design and Verification Methods*, pages 54–66, 1999.
- [YYC00] Chien-Chih Yu, Hung-Chao Yu, and Chi-Chun Chou. The Impacts of Electronic Commerce on Auditing Practices: An Auditing Process Model for Evidence Collection and Validation. *International Journal on Intelligent Systems in Accounting, Finance and Management*, 9(3):195–216, September 2000.