A Reengineering Approach for Software Systems Complying with the Utilisation of Ubiquitous Computing Technologies

PhD Thesis

Mohammed Abdulrahman Alawairdhi

Software Technology Research Laboratory

De Montfort University

2009

Dedication

To my mother, my wife and my children Fatmah, Leen, and Abdulrahman

for their love, support and encouragement

during this time of challenges.

Declaration

I declare that the work described in this thesis is original work undertaken by me for the degree of Doctor of Philosophy, at the Software Technology Research Laboratory (STRL), Faculty of Technology, at De Montfort University, United Kingdom. No part of the material described in this thesis has been submitted for the award of any other degree or qualification in this or any other university or college of advanced education.

Acknowledgements

First and foremost, I am thankful to Almighty ALLAH for all his bounties and blessings, and for giving me the ability to complete this research

I was very fortunate to have Professor Hongji Yang as my first supervisor. His scientific support, insightful comments, ideas, visionary approach, support and guidance have greatly helped me in developing my thinking and technical writing, and widening my horizons.

My thanks also go to Professor Hussein Zedan, the director of the laboratory, for his support, encouragement and guidance through very exciting and enjoyable discussions. His scientific and moral support through the difficulties I faced during my research were crucial to my success.

I would like to thank all of my colleagues, especially Dr. Mousa AL-Akhras, Ruimin Liu, Sascha Westendorf, Feng Chen and Yang Xu, in the Software Technology Research Laboratory at De Montfort University for their valuable suggestions and discussions.

I would also like to thank the staff in the Research Office at De Montfort University for their outstanding management.

I am also deeply indebted to my friends Khamis Hussain Abdulbaset, Nasser Alahmed, Abdulaziz Aldhea'yan and Dr. Eisa Aleisa for their concern and encouragement through all my study years.

My special gratitude is due to my dearest brothers and to my lovely sisters and their families for their loving support, concern and encouragement.

I would like also to express my deepest gratitude to my loving mother, who gave her love and support, for everything she sacrificed in her life for me. Without her loving

care, prayers and support, it would have been very difficult for me to achieve my goals.

I would like to express my deepest love and gratitude for my wife, who stood by me in all these difficult years and has offered me her constant support, patience, encouragement, unconditional love and life.

Finally, my love goes to my children Fatmah, Leen, and Abdulrahman for the hope and encouragement they have given to me, without knowing it, to complete this work.

Abstract

The evident progression of ubiquitous technologies has put forward the introduction of new features which software systems can sustain. Several of the ubiquitous technologies available today are regarded as fundamental elements of many software applications in various domains. The utilisation of ubiquitous technologies has an apparent impact on business processes that can grant organisations a competitive advantage and improve their productivity. The change in the business processes in such organisations typically leads to a change in the underlying software systems.

In addressing the need for change in the underlying software systems, this research is focused on establishing a general framework and methodology to facilitate the reengineering of software systems in order to allow the incorporation of new features which are introduced by the employment of ubiquitous technologies. Although this thesis aims to be general and not limited to a specific programming language or software development approach, the focus is on Object-Oriented software. The reengineering framework follows a systematic step-based approach, with greater focus on the reverse engineering aspect. The four stages of the framework are: program understanding, additional-requirement engineering, integration, and finally the testing and operation stage.

In its first stage, the proposed reengineering framework regards the source code as the starting point to understand the system using a static-analysis based method. The second stage is concerned with the elicitation of the user functional requirements resulting from the introduction of ubiquitous technologies. In the third stage, the goal is to integrate the system's components and hardware handlers using a developed integration algorithm and available integration techniques. In the fourth and final stage, which is discussed in a general manner only in this thesis, the reengineered system is tested and put in the operation phase.

Abstract

The proposed approach is demonstrated using a case study in Java to show that the proposed approach is feasible and promising in its domain. Conclusions are drawn based on analysis and further research directions are discussed at the end of the study.

Table of Contents

Table of Contents

Table of Contents

List of Figures

List of Figures

List of Figures

List of Listings

List of Acronyms

ADL	Architecture Description Language
API	Application Programming Interface
AST	Abstract Syntax Tree
Auto-ID	Automatic Identification
AWT	Abstract Windowing Toolkit
BP	Business Process
BPD	Business Process Diagram
BPEL4WS	Business Process Execution Language for Web Services
BPMI	Business Process Management Initiative
BPML	Business Process Modelling Language
BPR	Business Process Reengineering
BPS	Business Process Simulation
CBSD	Component-Based Software Development
CFG	Control Flow Graph
CRM	Customer Relationship Management
CST	Concrete Syntax Tree
DEMO	Dynamic Essential Modelling of Organisations
ECA	Events, Conditions and Actions
ER	Entity-Relationship
ERP	Enterprise Resource Planning
EWSL	Extended Wide Spectrum Language
GP	Genetic Programming
GPS	Global Positioning System
GUI	Graphic User Interface
HTML	Hypertext Mark-up Language
I/O	Input/Output
IDE	Integrated Development Environment

IS	Information Systems
IT	Information Technology
J2EE	Java 2 Platform, Enterprise Edition
JDOM	Java Document Object Model
JDT	Java Development Toolkit
LCD	Liquid Crystal Display
LOC	Line Of Code
MDA	Model Driven Architecture
OFG	Object Flow Graph
OMG	Object Management Group
OMT	Object Modelling Technique
OOSE	Object-Oriented Software Engineering
PDG	Program Dependency Graph
RE	Requirements Engineering
RFID	Radio Frequency Identification
RIP	Remote Procedure Invocation
RTF	Rich Text Format
SCM	Supply Chain Management
SDBC	Software Derived from Business Components
SQL	Structured Query Language
STRL	Software Technology Research Laboratory
UI	User Interface
UML	Unified Modelling Language
WSL	Wide Spectrum Language
XMI	XML Meta-data Interchange
XML	eXtensible Mark-up Language
xUML	eXecutable UML

Chapter 1

Introduction

Objectives

- To present the motivation of proposing the reengineering framework.
- To explain the research method.
- To identify the research questions.
- To explain the research scope.
- To highlight the original contributions of the research.

1.1 Motivation

The noticeable advancement and maturity of ubiquitous technologies fuelled the introduction of new features which software systems can sustain. In the seminal article "The Computer for the 21st Century" by Mark Weiser [173] ubiquitous technologies are described thus:

"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it."

Ubiquitous technologies include worn computers, embedded systems, smart appliances, sensors and an assortment of wireless networking technologies. Such technologies introduced features to software systems like always-on service channel connection, awareness of user context and the ability to change the user environment through actuators and communication links. As in the case of wireless sensors and actuators are extending the reach of computing applications in ways that promise to transform our ability to communicate about and interact with things in the physical world. Weiser [173] described this as follows:

"a physical world that is richly and invisibly interlaced with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives, and connected through a continuous network"

At the current time, many of the aforementioned ubiquitous technologies have been utilised in traditional software applications in different domains. Healthcare, postal services, access control, transportation, as well as the military field are examples of the domains in which these pervasive technologies have been employed. Through automatic, real-time object tracking and the other features, ubiquitous technologies can furnish organisations with accurate data regarding their business operations in a more timely fashion, while automating the operations at the same time. These impacts of ubiquitous technologies on the business operations forced to some extent what is known as business process reengineering. Business process reengineering, as defined by Davenport [44], is a process that:

"...encompasses the envisioning of new work strategies, the actual process design activity, and the implementation of the change in all its complex technological, human, and organizational dimensions."

Although business process reengineering is not completely reliant on information technology, the reengineering of business processes most of the time is carried out simultaneously with the reengineering of the underlying software systems. The sixth law of Lehman's laws of software evolutions [92] states that:

"the functional content of a program must be continually increased to maintain user satisfaction over its lifetime."

Hence, software reengineering aims to position existing systems to take advantage of the newly introduced technologies. At the same time it enables new software development efforts to take advantage of reusing existing systems.

In an ideal situation, when developing a software system, the design of the software will be well-structured, the testing will be designed in conjunction with the code and the documentation will reflect the appropriate models and rationale. In addition, the changes in the requirements and the modifications of the software will be reflected in the documentation. However, the real world is not ideal. After the software system has been put in the operation phase, business rules may change, the original developers of the system may be replaced and the documentation of the system may become outdated. As a result of changing the development team and the out-of-dating of existing documentation, the source code turns out to be the only guide to the system static structure and dynamic behaviour. In addition, maintenance and reengineering activities are hindered by the lack of means to comprehend the targeted system.

Several research studies have been conducted to address various challenges in the field of ubiquitous technologies. Some these studies aim at developing what are known as smart appliances [41]. Other studies have devoted a significant effort in constructing smart or context-aware environments [41]. Unlike these research studies, this research addresses the need for reengineering software systems to obtain advantage of the new

features and capabilities acquired through the utilisation of ubiquitous computing technologies. Weiser said that

"[…]Like the personal computer, ubiquitous computing will enable nothing fundamentally new, but by making everything faster and easier to do, with less strain and mental gymnastics, it will transform what is apparently possible. […] But ease of use makes an enormous difference."[173]

This research proposes a unified software reengineering framework from a business logic perspective. Business logic as defined by Zue [185], refers to

"a set of interrelated tasks linked through a number of decision activities with a starting and ending point".

The reengineering framework comprises four stages, starting with program understanding and ending with testing and operation. This research regards three distinctive aspects which contributed to shaping the final form of the proposed framework. These aspects are: business logic and ubiquitous computing, in addition to the software reengineering process per se.

1.2 Research Method

The research approach used in this study falls in the category of a *formulative* approach based on Morrison and George's [109] categorization of research approaches in software engineering. Formulative research is concerned with "involving development and refinement of theories, models, or frameworks that govern research activities and support scientific progress through paradigm shifts."[109]. This research is also regarded as constructive research. The constructive research approach is "a research procedure for producing innovative constructions, intended to solve problems faced in the real world and, by that means, to make a contribution to the theory of the discipline in which it is applied." [31] A "construction" can be a new theory, algorithm, model, framework or method. Since there is a need to in this research to evaluate the

available tools and techniques empirical research is also required. Empirical research is a "research that is based on experimentation or observation, i.e. evidence. Such research is often conducted to answer a specific question or to test a hypothesis." [83]

Based on the research approach adopted, the present research was realised through the following method:

Step 1: Research Problem and Questions Identification.

Understanding of the problem in its full scope was obtained from a study of the relevant literature. Literature relating to the research problem was sought, studied and analysed to determine the algorithms and tools that were used with their potential for adoption. In order to narrow down the research problem, a set of research questions was proposed and formulated to tackle the underlying problem issues.

Step 2: Solution Construction.

The reengineering framework was constructed; it includes four stages: program understanding, additional requirements engineering, integration and finally testing and operation. Algorithms for code refinement and systems integration were also developed. This is in addition to the development of a method of business logic extraction from the source code of the software systems

Step 3: Validation.

A case study was used to validate and demonstrate the feasibility of the proposed approach, which showed that the methodology can work and produce impressive results. The case was carefully selected to be representative, and also to span the potential application space of the targeted application domain.

Step 4: Deriving Conclusions

Conclusions were drawn and the applicability of the methods was discussed based on the experiences from the evaluation. As research in nature is an iterative process, new relevant research questions were proposed and formed to motivate future research in the area.

1.3 Research Questions

This research is driven by issues which need to be addressed in order to incorporate the ubiquitous technologies within existing systems. These issues inform a rationale for this study. The overall research question this thesis attempts to answer is:

How to reengineer software systems to incorporate the new features introduced through the utilisation of ubiquitous technologies?

As the reengineering of software systems is a complex process which involves a large number of tasks and relates to various features of software artefacts, the study focused in three aspects that influence the regarded reengineering process. The aspects considered in this thesis are: business logic, ubiquitous technologies and the reengineering process itself.

From these determined aspects a set of research questions that address the problem in detail is defined in order to be able to answer the overall research question

RQ1. How to recover the business logic implemented in the existing system?

RQ1.1. What is business logic?

RQ1.2. How to extract business logic from source code?

RQ1.3 What other information needs to be recovered to facilitate the completion of the reengineering process?

RQ2. How do the features introduced by ubiquitous technologies impact the existing systems?

RQ2.1. What is a ubiquitous technology?

RQ2.2. What are the features introduced by ubiquitous technologies?

RQ2.3. How do the new features affect the business logic?

RQ2.4. What new requirements may result from effects of ubiquitous technologies?

RQ3. How can the new requirements resulted from effects of ubiquitous technologies be elicitated?

RQ3.1. What type of requirements to be elicitated?

RQ3.2 How the requirements can be elicitated?

RQ4. How will the reengineered system gets integrated?

RQ4.1. How to identify the software components need to be added modified or removed?

RQ4.2. How to integrate the new or modified components?

RQ4.3. How to integrate the hardware handlers of the ubiquitous technologies?

1.4 Scope of Research

In this thesis, the research is focused on establishing a general framework and methodology to facilitate the reengineering of software systems in order to allow the incorporation of new features introduced by the employment of ubiquitous technologies. The proposed reengineering framework considers the business logic as the corner stone for constructing its reengineering process stages. The framework follows a systematic step-based approach, the steps of which can be generally classified into the following four main stages: program understanding, additional-requirement engineering, integration, and finally the testing and operation stage.

In its first stage, the proposed reengineering framework addresses the problem of the possible lack of up-to-date documentation by regarding the source code as the starting point to understand the system using the reverse engineering techniques. The second stage of framework is concerned with the elicitation of the new user requirements resulting from the introduction of ubiquitous technologies. In the third stage, the goal is

to integrate the new or changed components using a devised integration algorithm. In the fourth and final stage, the reengineered system is tested and put in the operation state.

Although the proposed framework is for reengineering, the focus is on the reverse engineering rather than the forward engineering. In addition, in this research, the proposed framework for reengineering is aimed to be general and not limited to a specific programming language or software development approach. However, in this thesis the focus is on Object Oriented software.

In particular, the scope of this research is highlighted for each stage of the proposed reengineering framework as follows:

1.4.1 Program Understanding Stage

In the program understanding stage, which is the main focus of this research, several reverse engineering techniques and tools have been employed. A static-analysis based method was used in this stage. In order to achieve the first objective of the stage, which is the architecture recovery, only the logical view of the "4+1" model of Kruchten [88] will be considered. Hence, in this step of this stage, the UML class diagram and the class dependency diagram are recovered from the source code.

In order to accomplish the second objective of this stage, which is the business logic extraction, several tools and methods have been put in use. First the code is decomposed and grouped into three groups: boundary (interface) classes, control classes, and entity classes. It is assumed that this grouping can be accomplished using program slicing techniques. Each class in the control classes group is parsed into an abstract syntax tree (AST). The generated AST is traversed to filter out a non-business logic-related code using a devised abstraction algorithm. Although the developed algorithm is not limited to a specific programming language, Java is used in this thesis to describe the algorithm. Then UML activity diagrams are created from the refined code. Finally, the UML diagrams are transformed into Business Process Modelling Notation (BPMN) diagrams.

1.4.2 Additional-Requirements Engineering Stage

In the additional-requirements engineering stage, the new business logic resulting from the introduction of the ubiquitous technologies is elicited. The research focused on the elicitation of the user functional requirements which was done through a scenario-based elicitation method. The four steps followed in this thesis to carry out the additional-requirements elicitation process are: scenario description in plain text, scenario analysis using Carroll's model [33], scenario modelling in BPMN and finally functions pseudo code development.

1.4.3 Integration Stage

In the integration stage, an integration algorithm was developed. The devised integration algorithm makes use of the outcomes of stages one and two. The integration algorithm is at the BPMN diagram level. However, the low level integration considerations were addressed using the Component-Based Software Development (CBSD) techniques. In the same stage, the integration of the ubiquitous technologies hardware handlers is also maintained using the systems integration patterns.

1.4.4 Testing and Operation Stage

Since the testing and operation stage is not in the scope of this thesis, it has been discussed in a general manner only. The general discussion highlighted prominent factors which need to be taken into consideration when carrying out the testing and operation tasks. Such considerations need to address the three aspects of this research; these are: business logic, ubiquitous computing and software reengineering.

1.5 Contributions

This thesis aims to construct a reengineering framework to support software reengineering for software systems effectively, while complying with the utilisation of ubiquitous technologies. Compared with the previous published work, although many

aspects of software reengineering and ubiquitous computing have been studied, the combination of the three aspects - business logic, ubiquitous technologies and software reengineering - in one reengineering framework distinguishes this research. Concretely, the key contributions of this thesis are as follows:

1. **Unified Approach:** The framework proposed in this thesis starts with understanding the source code of the system to be reengineered as the only documentation available, and ends with explaining the integration of the desired reengineered system. The framework is limited to neither a specific programming language nor a specific software development process.

2. **Business Logic Extraction Method:** The business logic extraction method used in this research is novel. The outcome of the process is represented in a manner conceivable by both technical and non-technical professionals.

3. **Code Refinement Algorithm:** The research proposes a new code abstraction algorithm. The algorithm aims at filtering out the non-business-logic related code entities from the source code.

4. **New Uses for BPMN:** The normal use of BPMN is for business process modelling. In this research, BPMN is used in the reverse engineering tasks and also in the integration process.

5. **Integration Algorithm:** The devised integration algorithm presented in the thesis works in a high abstraction view of the system. The algorithm addresses the issues of what and where to integrate the new or modified system components.

1.6 Thesis Organisation

The remaining contents of the thesis are organised into 9 chapters, as follows:

Chapter 2. Provides an overview of the background information which influenced the proposed approach. The background information includes software engineering and software evolution, business logic concepts and ubiquitous computing concepts. The chapter also discusses the impact of ubiquitous computing and software reengineering,

Chapter 3. Gives a literature review covering the related research work. This includes software understanding and visualisation concepts, UML and BPMN in addition to recent studies in business logic recovery approaches and methods.

Chapter 4. Presents an overview of the framework for reengineering systems in utilising ubiquitous technologies. It also presents the key stages of the framework and challenges in the reengineering process.

Chapter 5. Describes the first stage of the reengineering framework which is the program understanding. The chapters go through the steps for comprehending the reengineered system via recovering the system architecture and extracting the implemented business logic.

Chapter 6. Explains the second stage of the proposed framework which is the requirements engineering. The chapter discusses the user functional requirements elicitation process using a scenario-based method.

Chapter 7. Discusses the third stage of the reengineering framework which is the integration stage. The integration algorithm proposed in this study is explained, along with the integration of required hardware handlers.

Chapter 8. Presents a case study to demonstrate the practical applicability of the proposed approach of reengineering software systems complying with the utilisation of ubiquitous technologies.

Chapter 9. Summarises the work presented in this thesis, highlights the significance of the proposed contributions and discusses directions for possible future directions.

Chapter 2

Background and Basic Concepts

Objectives

- To present an overview of software evolution and reengineering
- To discuss business logic and software reengineering
- To define basic concepts related to ubiquitous computing.
- To discuss the impact of ubiquitous computing and software reengineering.

2.1 Software Evolution and Software Reengineering

In this section, software evolution is discussed in Section 2.1.1. Section 2.1.2 describes the legacy system and software evolution. Section 2.1.3 introduces the software evolution process. In Section 2.1.4 software reengineering is presented in addition to its approaches in Section 2.1.5. Finally in Section 2.1.6 an elaboration of the reverse engineering tasks is provided.

2.1.1 Software Evolution

The term ‘evolution’ is used to explain a prevalent phenomenon across many domains which results from concurrent changes in the properties of the evolving entity. Ideas, concepts, cities, societies, and also natural species are all evolving, but each does so within its own context [102]. The term evolution in the context of software generally denotes progressive change in software characteristics or properties [94]. Software evolution is normally a lengthy process which includes execution, usage, enhancement, extension, and update of a software system. The purpose of the software evolution can be corrective actions to fix hidden defects, adaptive actions to handle changing environments, or perfective actions to adapt with new to necessities [12]. Software evolution can be conceived as a repeating software re-engineering cycle which includes forward engineering and reverse engineering tasks. In other words, the term evolution describes a situation in which reengineering and maintenance are frequently needed frequently [181]. The term software evolution is normally used in contrast to software maintenance, owing to the negative inferred meaning of the latter term. Maintenance gives the impression that the software itself is declining, which is not always the case. It is sometimes the changes in the software environment or user needs that make it essential to adapt the software [159].

There are two common viewpoints in software evolution studies, normally described as the ‘what’ and ‘why’ versus the ‘how’ perspectives. The what and why view deals with software evolution as a scientific discipline [93]. It investigates the nature of the software evolution phenomenon, and tries to comprehend its driving factors and its

impact. This point of view requires interdisciplinary research including non-technical sides such as human psychology, social interaction, complexity theory, and many more. On the other hand, the 'how' view looks at software evolution as an engineering discipline. It explores the pragmatic aspects that sustain the software developers or project managers in their daily tasks. Therefore, this view mainly pays more attention to technology, methods, tools and activities that provide the methods to direct, implement and control the software evolution [105].

After the software systems have been developed, they inevitably have to change if they are to remain useful. Keeping up with rapid change in the market and the demand for either new features or enhanced ones mean that such systems to tend to evolve continuously during their lifetime [144]. Generally, there are four different reasons for software change: perfective, corrective, adaptive, and preventive. First, perfective changes are changes made to improve the software, for example by adding new user functions, or enhancing performance such as response time. Second, corrective changes are those which aim to repair any defects in the system. Third, adaptive changes are made so that the user can keep up with the speed of the changing environment. Operating systems and compilers of programming languages are examples of such rapid change. Fourth, adaptive changes are changes for enhancing the system's future maintainability and reliability in order to simplify the future evolution [102, 181].

2.1.2 Legacy Software and Evolution

The term 'legacy system' is presently a well-consented upon and well-defined term within the software community. In other words, it should be said that there is really very little difference between the definition of legacy software in the commercial world and the scientific community when one considers computer software [181]. Legacy systems are informally defined as "large software systems that organisations do not know how to deal with but they are very critical to the organisation's main businesses" [13]. Brodie defined a legacy system as any information system that significantly oppose to change and evolution to meet new and continuously updating business requirements [25]. The legacy software problem used to be associated with old and large systems which were

normally coded in early versions of a third-generation language, such as COBOL or FORTRAN. The systems were built using relatively inflexible architectures and technology, and they had not been designed to adapt to changes over an extended period of time [102]. Yet today's software systems are larger and more complex with the rapid development of applications and requirements. Newly developed systems can quickly become legacy ones as they do not accommodate the new requirements of their users or they become unable to support business evolution fully [8].

One of the several reasons for newly developed software system being considered as 'legacy system' is often the lack of common language and communication channels between system developers and system users. Moreover, the absence of suitable and sufficient feedback may result in segregated evolutionary tracks [102]. On the other hand, Brooks claims that "problems associated with the software legacy crisis are caused by the characteristics of the software itself". This includes software complexity, conformity, change, and invisibility [26].

Normally cost plays an essential role when a decision is taken by an organisation for upgrading and maintaining their legacy systems in order to get the best return from their investment. This implies that organisations need to go through a critical assessment of their existing system in order to choose the proper strategy for evolving such systems. In general, there are four strategy options which are not exclusive and different options may be applied to different parts of the system [144]:

- Scrap the system completely. This strategy can be used when the system is not effectively contributing to the organisation's business processes.

- Keep the system and maintain it. This strategy can be used when the system is sufficiently stable and there are small numbers of requests from the system's users.

- Re-engineer the system. This strategy can be used when there is a declined in the system quality because of the regular change and there are more changes required to be carried out.

- Replace all or parts of the system. This strategy can be used when there are other factors that hinder the system from fulfilling its operation, such as hardware upgrade.

2.1.3 Software Evolution Process

The process of software evolution in an organisation differs considerably depending on the kind of software being evolved, the development process used, and the people engaged in the process. The evolution process normally includes essential activities of change analysis, release planning, system implementation, and finally delivering the system to the customer [144]. After more than thirty observations in many projects in the domain of software evolution, such as Feedback, Evolution And Software Technology (FEAST) projects, Leham, Ramil and their colleagues defined eight laws for software evolution. These laws were based on their definition of software evolution, which states that software evolution processes are feedback systems, and suggest feedback as a basis for direct relationships between the laws [94]. The laws, which are usually called Leham laws, are [92]:

- Law one: Continuing Change: An E-type program that is used must be continually adapted otherwise it becomes progressively less satisfactory.

- Law two: Increasing Complexity: As a program is evolved its complexity increases unless work is done to maintain or reduce it.

- Law three: Self Regulation: The program evolution process is self regulating with close to normal distribution of measures of product and process attributes.

- Law four: Conservation of Organisational Stability (Invariant Work Rate): The average effective global activity rate on an evolving system is invariant over the product lifetime.

- Law five: Conservation of Familiarity: During the active life of an evolving program, the content of successive releases is statistically invariant.

- Law six: Continuing Growth: The functional content of a program must be continually increased to maintain user satisfaction over its lifetime.

- Law seven: Declining Quality: E-type programs will be perceived to be of declining quality unless rigorously maintained and adapted to a changing operational environment. E-type program is a computer program that solves a problem or implements a computer application in the real world domain.

- Law eight: Feedback System: E-type programming processes constitute multi-loop, multi-level feedback systems and must be treated as such for them to be modified or improved successfully.

In the same sphere, Yang *et al*. also developed and experimented with a practical process for software evolution which takes the following stages [181]:

1. To translate the source code into EWSL (Extended Wide Spectrum Language);
2. To restructure (including clustering and visualising code);
3. To abstract;
4. To understand with the support of a cognitive tool;
5. To reuse components;
6. To retarget;
7. To measure evolution.

2.1.4 Software Reengineering Concepts

Software reengineering was defined by Chikofsky and Cross [35], as "the examination and alteration of a system to reconstitute it in a new form". Software reengineering is "a higher level and structural form of change that makes the software systems have easy and high quality maintenance" [181]. The aim of reengineering a

system is to understand the existing software and then to re-implement it in order to improve the system's functionality or performance [119]. There are four general re-engineering objectives in addition to the specific objectives of a re-engineering task decided by the goals of a certain organisation. They are: preparation for functional improvement, enhanced maintainability, migration, and enhanced reliability [11].

There are two key advantages for reengineering over radical approaches to system evolution [144]:

1. Reduced risk. There is increased risk in re-developing business-critical software because of the errors that may be introduced in the system specifications or in the development process itself.

2. Reduced cost. The cost associated with reengineering process is noticeably lower than the cost of developing totally new software from scratch.

The following terms provide some of the classification of the domain of software reengineering as clearly explained by [181]:

- ***Forward engineering*** *is the traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system.*

- ***Reverse engineering*** *is the process of analysing a subject system to (1) identify the system's components and their interrelationships and (2) create representations of the system in another form or higher level of abstraction.*

- ***Restructuring or Refactoring*** *is the transformation from one representation form to another at the same relative abstraction level, while preserving the subject system's external behaviour (i.e., functionality and semantics) yet improves its internal structure. Refactoring makes reuse of both the domain knowledge and the source code.*

- ***Design recovery or Reverse design*** *is a subset of reverse engineering. Design recovery recreates design abstractions from a combination of code, existing design documentation (if available), personal experience, and general knowledge about problem and application domains.*

- ***Program Transformation*** *is the act of changing one program into another. The term program transformation is also used for a formal description of an algorithm that implements program transformation. The languages in which the program being transformed and the resulting program are written are called the source and target languages, respectively.*

- ***Program Understanding or Program Comprehension*** *is a term related to reverse engineering. Program understanding always implies that understanding begins with the source code, while reverse engineering can start at a binary and executable form of the system or at high-level descriptions of the design. Program understanding is comparable with design recovery, because both start at source code level.*

2.1.5 Approaches to Software Reengineering

Techniques of software reengineering can be categorised in two groups: formal methods and cognitive methods.

2.1.5.1 Formal Methods of Reengineering

The term ‘formal method’ refers to methods that have mathematical soundness. In other words, formal methods provide a mathematical ground for the software system design and verification against its specification. The formal methods enable the functionalities of software systems to be specified precisely. These methods are typically composed of some essential components, such as semantic model,

specification language, refinement calculus, verification system, development guideline, and other supporting tools [181].

Formal methods used in reverse engineering activities deal with the source code of certain software as an input in order to produce a formal specification as an output. Therefore, some researchers who support the use of formal methods in reengineering, such as Baumann, state that the use of non-formal methods causes the extraction of wrong information. He claims that a method which is not based on a sound foundation and does not require formal denotational semantics can lead to errors that affect the reengineering of a software system [17]. Other researchers also assert that the use of formal methods improves the understanding of a system through the exposure of undetected inconsistencies, ambiguities, and incompetence [37].

Formal methods can be classified into five types of methods: model-based, logic-based, algebraic, process algebra and net-based methods. In a model-based approach "the states and operations that transform a system from one state to another are defined. In this approach, there is no precise representation of concurrency and expression of non-functional requirements can be done in some cases" [181]. A logic-based approach uses logic to describe the required properties, including low-level specification, temporal, and probabilistic behaviours. With some concrete programming concepts, the logic can be amplified to acquire what is known as wide-spectrum formalism. In the algebraic approach, operations are defined implicitly through making a connection between the behaviour of different operations with no definition of the actual states. The process algebra approach "provides an explicit representation of concurrent processes and represents system behaviour by constraints on all communications between processes which are allowable and observable" [181]. The net-based approach provides particular advantages in system development and reengineering through the use of graphical languages with formal semantics. This approach is very popular, as it uses graphical notation in specifying systems which makes them easier to understand and more accessible to non software engineering specialists [181].

2.1.5.2 Cognitive Methods of Reengineering

Cognitive method of reengineering describes the mental process or faculty of knowing a software system [129]. The cognitive methods, which are also known as "structured methods are well-defined but do not have a sound mathematical basis to describe system functions" [60]. Cognitive methods depend heavily on domain knowledge, which includes activities such as browsing the code in delocalised plans and building abstractions. The problem in carrying out such actions of reverse engineering using a cognitive method is the need to throw away some information when jumping from a level to another to achieve the desired abstraction [147].

In the cognitive method of software reengineering there are two approaches commonly used in program understanding: functional and behavioural. The functional approach accords emphasis on what a system does, while the behavioural approach emphasises cognition by how a system performs [158].

- *The functional approach is bottom up and deductive. It depends on the knowledge of the implementation domain to produce a higher level of abstractions that may map to the application domain and the system's functional requirements.*

- *The behavioural approach is top down and inductive. It uses hypothesis assumption and refinement to match artefacts derived from the knowledge of the application domain onto the related software system.*

2.1.6 Reverse Engineering

Chikofsky and Cross [35] defined reverse engineering as "analysing a subject system to identify its current components and their dependencies, and to extract and create system abstractions and design information." Reverse engineering comprises actions to identify or recover program requirements and design specifications that assist in comprehending and changing the system [180] . Reverse engineering is "the process of transforming code into a model through mapping from a specific implementation

language" [19]. Reverse engineering is a development process based on the notion of taking something apart to see how it works and then putting it back together again [67]. The main aim of reverse engineering is "to discover the underlying software system's features including requirements, specification, design, and implementation" [181]. In other words, the process of reverse engineering is to obtain and record abstracted information about the system which includes the system's structure, dynamic behaviour, functionality, and construction [181].

There are numerous subareas of reverse engineering. The oldest and simplest two subareas that are broadly mentioned in the literature are software re-documentation and design recovery. Re-documentation is "a revised version of a semantically corresponding representation within the same relative abstraction level" [35]. The resulting forms of representation are regularly regarded as alternate views such as dataflow, data structure, and UML diagrams. On the other hand, within the subarea of design recovery, "domain knowledge, external information, and deduction are added to the observations of the subject system to identify significant higher level abstractions further than the abstractions accomplished by direct examination of the system" [35].

Today, there are several reverse engineering tools available in academic and industry environments. Although these tools facilitate many reverse engineering activities, they still have not yet become an effective and integral part of the standard toolset a software engineer needs in his daily activities. In spite of the good demand in the software market for code debugging tools, testing utilities and integrated development environments (IDE), the market for reverse engineering tools remains to some extent limited [110].

2.2 Business Logic

In this section the terms relating to the second concept regarded in this research, namely 'business logic', are discussed. The following section, Section 2.2.1, gives an overview about the definition of business process and process modelling. Section 2.2.2 covers business rules. Section 2.2.3 describes business process reengineering. Section

2.2.4 relates all these concepts to requirements engineering and user requirements.

2.2.1 Business Process

A business process is "a set of tasks that are performed in coordination in an organisational and technical environment. Together, these tasks accomplish a business goal" [71]. Hammer and Champy define the business process as a series of interrelated activities that take an input, add value to it, and produce an output that is of value to the customer [176]. In other words, a business process is characterised by a beginning and end that transform a certain input according to a set of business rules into a defined output. Well defined business processes have, according to Harrington [71], some important features: they contain a role, have well defined boundaries, have documented procedures, have known cycle times, and have well defined internal interfaces and responsibilities.

There is a substantial difference between tasks and processes. A process consists of a number of tasks that need to be performed based on a set of predefined business conditions that specify the order of the tasks. On the other hand, a task is a logical unit of work that is performed as a single whole by one resource. A resource can be a person or machine which is able to perform a specific task in the whole process [1].

There are many research studies working in the field of modelling business processes in order to map them to an information technology model. Nevertheless, how the mapping is going to be accomplished is still in the early stages of development [84]. Several of the existing business processes modelling tools that implement various approaches normally have many features in common. Every tool attempts to describe these four main issues: the business tasks to be automated, the location of the automating system to be used, the user of the system, and integration with other systems. In short, the following are conventional elements of a business process modelling language [51]:

- *The organisational model is a static view which depicts the roles and areas of responsibilities inside an organisation concerned with the tasks of a business*

process.

- *The control flow describes the execution order and dependencies among the various tasks of a business process.*
- *The data flow depicts how the business artefacts are affected by the various tasks in the process.*
- *Use cases describe the context of a business process and its externally visible behaviour.*
- *Collaboration diagrams document how business agents and artefacts work together to perform a certain task.*

2.2.2 Business Rules

From the earliest times of business activity, there have been rules that govern the activities of the business. In an ordinary architecture of a given business, the business rules are the defined rules under which a set of resources interact through a set of well defined processes to accomplish certain goals [70]. In fact, there exists no industry-standard definition for the term 'business rule'. However, in the literature, several definitions have been used to describe the term. One of the definitions stated in [68] is that "A business rule is a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behaviour of the business." Morgan, in his book "Business Rules and Information Systems" [108], defined the business rules as "a statement by which some aspect of the business are defined or constrained". It is designed to declare business structure or to produce an effect on the business behaviour.

Since there is not a universal scheme for the classification of business rules, business rules can be categorised based on different aspects. In the classification scheme suggested by the Business Rules Group, business rules are divided into the following [68]:

- *Derivation rule: a statement of knowledge that is derived from other knowledge in the business.*

- *Structural assertion rule: a defined concept or a statement of a fact that expresses some aspect of the structure of the enterprise. This encompasses both terms and the facts assembled from these terms.*

- *Action assertion rule: a statement of a constraint or condition that limits or controls the actions of the enterprise.*

Another example of business rules taxonomy is to classify the rules according to global rules, activity rule, and procedure rules. Global rules define or restrict organisational behaviour and relate to the business process as a whole. Behavioural rules are "detailed rules controlling specific business process activities. Procedure rules control the operations within the activities of a specific process" [86].

Business systems function based on the business rules in a particular business domain. The dynamic nature of business forces a business environment to change often according to internal and external factors such as changes in law, or new competition [165]. Businesses face many limitations of their existing systems owing to the frequent changes in their businesses. They are always trying to continue updating their existing software applications to keep up with rapid changes of business rules. In traditional business software applications, business rules are defined inside the implementation of the application itself. There exists a number of problems associated with this approach. It is costly and slow to change in accordance with the changing business environment. It is not easy to locate business rules that are hard coded inside the program. It is also difficult to comprehend the translation of business rules into the implementation code [7].

In an effort to solve the aforementioned problems, business rule systems were introduced. A business rules system is "an automated system that separates the rules of the business logically, perhaps physically, from other aspects of the system and shares them across data stores, user interfaces, and applications" [68]. The main goal of these

systems is to separate these three parts of a system: rules, data, and process aspects.

2.2.3 Business Process Reengineering

A business process, as defined by Hammer and Champy and mentioned earlier in this chapter, means a series of interrelated actions that take inputs, add value to them, and produce outputs that are of value to the customer [176]. In 1993, these authors also triggered the concept of business process reengineering when they published a book about the business revolution [69]. They defined business process reengineering thus: "Business Process Reengineering (BPR) is in essence a performance improvement philosophy that aims to achieve quantum improvements by primarily rethinking and redesigning the way that business processes are carried out." In the same period, another definition of BPR which introduced the term 'process innovation' was offered by a well-known researcher in the field of business improvement, Thomas H. Davenport [43]. He defined BPR as a process that "encompasses the envisioning of new work strategies, the actual process design activity, and the implementation of the change in all its complex technological, human, and organizational dimensions."

Although BPR is not completely reliant on information technology (IT), IT can contribute to the enhancement of a specific process by enabling processes to be executed in new ways. BPR is regarded as an essential enabler for new sorts of working together either inside or outside an organisation. The following are a few examples of the methods by which IT is able to change business processes [24]:

- Automating and expediting processes.
- Increasing interactivity and allowing instant feedback.
- Enabling intelligent and effective knowledge creation, sharing, and management.

Sometimes the redesigned business process needs to be implemented by appropriate information system support. This is the result of transforming the strategic requirements

emerging from the business strategy and formulating them into operational structures. It is highly important to consider software legacy systems when redesigning business processes, because these systems are significant assets containing valuable information about current activities and business rules [84].

BPR documentation tools are intended for use by process-oriented business professionals rather than technical information systems specialists. BPR software enables the capture of the key elements of a business process in a visual representation; these usually include activities, sequencing, resources, times, and rules. The software includes some aspects of project management in terms of allocating resources and costs to work activities. BPR software also has "what if" scenarios that enable process simulation and performance comparison of alternative process designs. While BPR software is used for business process design, workflow management tools are used to automate the managing of the execution of the business process when it is in production and being used on a daily basis [50].

BPR can largely differ based on organisations strategic direction. Cost, supply chain, production, and marketing do not need the same reengineering strategies, but have different tools with different outcomes. Therefore, it is always recommended that "after reengineering opportunities have been identified, steps should be taken to ensure they are in correspondence with the origination's strategy, goal, and objective" [63].

2.2.4 User Requirements

The term requirement has various definitions among the software industry professionals. In one direction of these definitions, requirements are basically defined as a high-level abstract of services that a certain system should sustain [144] while at the extreme, requirements are defined as a detailed, formal definition of a system function [44]. When designing a system identifying and eliciting the requirements are considered to be not only the first step but the most important step in the software development life cycle. This process is normally called requirement engineering (RE). Similarly, there are many definitions that describe this process. Zave [183] provides a clear definition of

RE: “Requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behaviour, and to their evolution over time and across software families.”

In the process of requirements engineering, there are different and clearly separated levels and types of requirements needed to extracted and analysed. There two main classifications to be considered when talking about software requirements: user requirements and system requirements. User requirements are statements which are normally depicted in natural language in addition to diagrams, and they are expressed in a way that the user easily understands. They focus on describing the services a system is anticipated to offer and the restrictions under which the system will normally function. On the other hand, system requirements describe precisely what is sometimes called the functional specification document, which is the systems functions that need to be implemented [90].

As mentioned earlier, user requirements define what the user requires of the system as a whole. A good set of stakeholder requirements can provide a brief non-technical description of a system at a level that is accessible for both system developers and senior management. For most of the time, these requirements are divided into two main categories: functional and non-functional. Functional requirements specify functions that a system component should do in particular situations, and explicitly state what the system should not do. By contrast, non-functional requirements state the characteristics of the system to be achieved that are not related to its functionality, i.e. its performance, security, reliability, maintainability, accuracy, availability, capacity, types of users, changes to be accommodated, level of training support, etc. [87].

The process of requirement engineering comprises several steps, such as elicitation, modelling, validation, and integration. Within each step of the process there exist various types of techniques which can be utilised. For example, there are several types of requirement elicitation techniques. The choice of a specific technique relies on the time constraints and resources available to the software engineer, and the sort of

information that needs to be extracted. Each of these requirement elicitation techniques has its strengths and weaknesses. Examples of such methods are: model-driven techniques, prototyping, and contextual techniques [117].

2.3 Ubiquitous Computing

In this section the terms related to the third concept considered in this research, namely ubiquitous computing, are discussed. The following section, Section 2.3.1, gives an overview about the definition of the terms ubiquitous and ubiquitous computing. Section 2.3.2 provides an overview of a related concept which is smart environments. Section 2.3.3 describes context awareness provided by ubiquitous technologies. Section 2.2.4 gives an overview of RFID as an example of widely-used ubiquitous technology. Finally Section 2.3.5 highlights the impacts of the introduction of ubiquitous technology to the business application.

2.3.1 Ubiquitous Computing

Since the publication of seminal paper of Mark Weiser on ubiquitous computing "The Computer for the 21st Century" [173], many research organisations have devoted a huge effort in order to make what he discussed in that paper a reality. Weiser described a ubiquitous environment thus "Hundreds of computers in a room could seem intimidating at first, just as hundreds of volts coursing through wires in the walls did at one time. But like the wires in the walls, these hundreds of computers will come to be invisible to common awareness. People will simply use them unconsciously to accomplish everyday tasks." Over the years from the vast number of research studies in the field of ubiquitous computing (pervasive computing), various definitions of the term have evolved. One of these definitions [66] describes ubiquitous computing as "saturating an environment with computing and communication capability, yet having those devices integrated in environment such that they disappear."

Ubiquitous computing aims to "enhance computer use by not only making many computers available through the physical environment, but making them effectively

invisible to the user. Hence, a pervasive computing system should be pervasive, embedded, nomadic, adaptable, powerful, yet efficient, intentional and eternal" [175]. In general, ubiquitous computing exhibits the following features:

- Context sensitive: services are triggered or provided based on physical conditions.
- User focus: service should be aware of real-world status and user situations.
- Automatic: services are automatically triggered by the system not by the intention of the user.
- Transparent: the user is not involved in underlying service operations.

A wide range of research topics relate to ubiquitous computing. Such topics include distributed computing, human-computer interaction, sensor networks, mobile computing, and artificial intelligence. In the last few years, ubiquitous computing has attracted a large amount of research activity in both academia and industry. Some of the well-known projects in the academic arena are Oxygen at MIT, project Aura at Carnegie Mellon University, and the Portolano Project as the University of Washigton. In industry there are several projects, including EasyLiving by Microsoft and Cooltown by Hewlett-Packard [41].

2.3.2 Smart Environments

Smart environments, also named intelligent environments are defined by Weiser [41] as "a physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives, and connected through a continuous network.". In another definition, a smart environment is defined as "a region of the real world that is extensively equipped with sensors, actuators and computing components" [115]. Smart environments aim to provide every environment user with tools to enable them to control their surroundings seamlessly.

A smart home, for instance, is a house or living environment that comprises the adequate technologies that are able to control devices and systems automatically without user intervention [164]. In a similar manner, smart offices aim to involve computers in all of the activities of the office- users in order to help them with everyday tasks and ease their communication and collaboration. Some of the systems interact with users through voice, gesture, or movement [61].

2.3.3 Context-Awareness

Context as a concept has been widely used in various fields of science, including linguistics, philosophy, artificial intelligence, and communication. In [136] the context been described in the software domain thus "context-aware software adapts according to the location of use, the collection of nearby people, hosts, and accessible devices, as well as to changes to such things over time. A system with these capabilities can examine the computing environment and react to changes to the environment." Another operational interpretation for the term was introduced by Dey [46] in which he defines context as "any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."

There are different types of classification for context. One way to classify context instances is based on their dimensions. These two dimensions are called 'internal' and 'external' or 'logical' and 'physical'. The external context is what can be measured by sensors such as light, sound and location. The internal context, on the other hand, is determined by the user himself or by the user's interactions. Examples of the internal dimension of the context are the user's tasks, goals, work context, and the emotional state of the user [126].

With regard to the types of applications that can utilise context information, a proposed classification introduces three categories of functions that an application may implement. The first category is related to the presentation of information. Applications in this category present context information to the user, or use context to suggest

options of selected actions to the user. The second category is related to the automatic execution of services. This category refers to applications that initiate a command, or reconfigure the system on behalf of the user based on changes in the context. The third category is related to the application used to capture attached context information storage for later retrieval [135].

2.3.4 Radio Frequency Identification

The term 'Radio Frequency Identification' (RFID) refers to "technologies that use radio waves to identify objects or entities automatically. There are several methods of identification, but the most commonly used method stores a serial number that identifies an entity or object, and possibly some other data, on a tiny chip that is attached to an antenna which is called an RFID tag" [55]. The antenna allows the chip to send the identification information to a reader. The reader transforms the reflected radio waves into digital data that can be transferred to a computer system which can make use of it in different applications. A typical RFID system is composed of two components [133]:

- Transponder: this is attached on the object to be identified;
- Reader: this is a separate device that can read data stored in the tags and also write on them depending in the type of technology used.

RFID tags are classified based on power source into three types [62]: active tags, passive, and semi-passive tags. Active tags have a transmitter and a battery. Passive tags have no battery. They are active only when a reader is nearby to power them, whereas semi-passive tags have a battery to operate their chip's circuitry, but communicate by puling power from a nearby reader. The read range for a passive tag is about 20 feet while the range for an active tag is approximately 100 feet.

Today there exists a vast number of applications that utilise RFID in many areas, including industrial applications, retail applications, transportation tracking systems, and healthcare systems. In addition, RFID enabled systems are used in schools and universities, museums, and libraries. Retail companies, for instance, use RFID to track

promotional displays, reduce out-of-stock and improve shipping and receive accuracy [106]. while in healthcare systems, hospitals and health-care providers use RFID technology to track patients and high-value assets, and also to ensure patient safety [171].

2.3.5 Ubiquitous Computing and Business Applications

In the past, the typical use of tiny devices and artificial intelligent applications was limited to laboratory experiments in research institutions. While at the present time there exist some ubiquitous technologies that are considered important building blocks for many applications in various domains including healthcare, libraries, and retail. The maturity of many of the available ubiquitous technologies also led to the adoption of these technologies in the business domain. Ubiquitous technologies can provide new kinds of services to businesses, because of the three primary capabilities they encompass. First, they provide a service channel for remote service providers via an "always on" connection. Second, they notify these services about the context of the user via sensors. Third, they allow these services to make a change in the user environment through actuators and communication links [148].

Currently, the three types of ubiquitous technologies having direct and effective impacts on business processes are automatic identification, localisation, and sensor technology. Automatic identification (Auto-ID), as is obvious from its name, is a technology that is used to identify products. There are normally two main tasks accomplished by any Auto-ID system: receiving an external signal from the item that needs to be identified and recognising that signal by retrieving the encoded information. Standard Auto-ID systems include RFID, barcode, smart card, and biometric systems [56]. Localisation, which is usually integrated with auto identification, has many different techniques to determine the location of an entity. Proximity, for instance, is a simple localisation technique by which an entity can be recognised within a cell. A localisation system determines the location of an entity via calculating the proximity of a known position of the monitoring device. Other techniques of localisation comprise lateration, angulation, and scene analysis. The Global Positioning System (GPS) that is

widely in use today uses the lateration technique [75]. The third technology that has an important influence on the business applications and attracts many researchers in academia is the sensor technology. At the present time, different sensor types, including thermal, visual, infrared, magnetic, seismic or radar sensors, are available. These sensors are used to monitor conditions such as "temperature, humidity, lightning conditions, pressure, noise levels, mechanical stress levels on attached objects, or current characteristics such as speed, direction, and size of an object" [59].

The information collected via ubiquitous technologies tools provide businesses with the capability to observe their clients, refine their marketing strategies, and continuously update their value plans. In a ubiquitous computing environment, clients can be uniquely identified and recognised by different types of sensors such as RFID. Then, according to the analysis of the client history data, businesses may offer to each client a customised service. For instance, based on factors such as previous purchase history, location, or environmental traits, a client may receive an immediate promotion or discount on a certain product or service [49]. A customer mobile can be utilised as a communication medium between the business and its client to transport promotions, distinguished content, or personalised information on the basis of the information collected through their presence in a network of sensors.

2.4 Summary

In this chapter, the background and definitions of the concepts that form the foundation of this research are reviewed. Such basic concepts related to software engineering include software evolution, legacy systems, software reengineering, reverse engineering and forward engineering. In addition, other concepts related to the proposed reengineering framework which are business logic and ubiquitous computing are also explored.

At the beginning of the chapter, the relationship between the legacy systems (which the author of this research prefers to call existing systems) and the software evolution is presented. Then the software evolution process has been described using the Leham's

software evolution laws and the suggested software evolution process stages in addition to the four strategies of evolving a system. After that, software reengineering is defined along with its objectives, namely the related classifications within the reengineering process. Such classifications include forward and reverse engineering, restructuring or refactoring, design recovery or reverse design, program understanding or program comprehension and program transformation.

The two categories of the software reengineering techniques were described in detail. The two groups are: formal methods and cognitive methods. The formal methods have a mathematical soundness, while the cognitive methods of reengineering describe the mental process of knowing a software system. Formal methods are classified into five types of methods: model-based, logic-based, algebraic, process algebra and net-based methods, whereas cognitive methods of software reengineering are classified into two approaches: functional and behavioural.

Before the last part of the chapter, the terms related to the second concept regarded in this research, which is the business logic, are discussed. Such terms include business process, business rule and business process reengineering. Then the relationship between these business concepts and the user requirement was highlighted.

In the last part of this background chapter, the concepts related to the third dimension of this research, which is the ubiquitous technology, were introduced. The term ubiquitous was defined. Concepts normally mentioned within the area of ubiquitous computing were also discussed. Such concepts included smart environments and context awareness. An overview of RFID as an example of widely used ubiquitous technology was given. Finally, the impacts of the introduction of ubiquitous technology to business applications were discussed.

Chapter 3

Related Research

Objectives

- To discuss the software understanding and visualisation concepts.
- To discuss UML as a business modelling tool.
- To discuss the role of UML in software life cycle and reverse engineering.
- To introduce BPMN and the relationship between UML and BPMN.
- To review the up-to-date approaches in extracting business logic from existing systems.

3.1 Program Understanding

This section reviews some of the basic concepts in program understanding. First, in Section 3.1.1, the terms are defined and the importance of stages in the reengineering process is highlighted. Section 3.1.2 lists the information that can be elicited from the source code, and the form that this information takes. Section 3.1.3 describes the concepts related to software visualisation and some of the available software visualisation tools. The last section, Section 3.1.4, discusses program slicing approaches and algorithms.

3.1.1 Program Understanding Concept

Since the 1980's, the topic of program understanding has attracted many software engineers. They introduced a number of theoretical and empirical approaches on analysis methods by which programmers can comprehend software that the have not developed themselves [96, 114]. Recent researchers also continued the investigation in the field, and have built various types of approaches and tools to ease and facilitate the process of understanding software [132, 138].

Program understanding is defined by Rugaber [38] as "The task of building mental models of the underlying software at various abstraction levels, ranging from models of the code itself to ones of the underlying application domain, for Software Maintenance, Software Evolution, and Reengineering purposes." Another definition for the term of program understanding was offered by Mueller [111] "Program comprehension is the process of acquiring knowledge about a computer program. Increased knowledge enables such activities as bug correction, enhancement, reuse, and documentation."

There are several related terminologies which are usually mentioned when exploring the literature about program understanding. These terms, as defined by Chikofsky and Cross [35], include reverse engineering, design recovery, and reverse specifications. Reverse engineering, as defined earlier, is the process of analysing a software system to produce a higher level representation of the system comprising its interrelated

components. In design recovery, design abstractions are regenerated from a union of existing design documentation, code, general knowledge and personal experience about a problem and its application domain. In design recovery external information and inference are augmented to the observations of a certain system to identify significant higher level abstractions further than those acquired by direct analysis of the system itself. In reverse specification, the term 'specification' refers to an abstract description of what the software aims to do [158]. Reverse specification as a process is a type of reverse engineering in which a specification is abstracted from source code or design description. In forward engineering, the specifications of software declare what the software is required to accomplish. However, the source code normally does not include this information. Only in rare instances is this information recovered from comments in the source code and from the people involved in the original forward engineering process.

Although program understanding is a closed term to reverse engineering, "program understanding begins with the source code while reverse engineering can start at a binary and executable form of the system or at high level descriptions of the design. Program understanding can be achieved in an *ad hoc* mode and no external representation has to arise, while reverse engineering is the systematic approach to develop an external representation of the subject system" [134]. It is very similar to design recovery, as both of these begin at source code level.

3.1.2 Elicitable Information from Software Code

There exist different abstract levels of understanding of a software source code. Therefore, there are different types of elicitable information, along with their supporting techniques that aim to provide program comprehension at any level of abstraction. The following are examples of the views that can be elicited from a source code:

- **Statistics view:** provides visual statistics associated with code that is divided hierarchically into subsystems, directories and files. Also, the relative sizes of the components, which components are stable and which are changing, and identified

error-prone codes can be viewed. Using animation, the historical evolution of the code can also be displayed [14]. The supporting techniques for generating statistics view are program metrics and visualisation.

- **Outline view:** allows software engineers to contract the amount of information necessary to comprehend programs, simplifying walking through them to localise given computations or to identify the role of a piece of code [15]. The supporting technique for generating outline view is flow analysis.

- **Logic view:** provides a mechanism of automated reasoning in response to a user's query. This is achieved by the production of the inter-modular data flow information acquired by the static analysis of code; this is put into a program dictionary which allows specific queries to be answered [30]. The supporting technique for generating logic view is flow analysis.

- **Algebraic view:** is a formal representation technique that describes a program algebraic specification. Whereas predicate logic describes the state of a computation as it proceeds, algebraic specification uses equations to denote the relationships between the operations that a domain provides [36]. The supporting technique for generating algebraic view is flow analysis.

3.1.3 Software Visualisation

In this section, the software visualisation concepts are reviewed in the following section, Section 3.1.3.1. Several of the software visualisation tools are surveyed in Section 3.1.3.2.

3.1.3.1 Software Visualisation Concepts

Software engineers require different tools and techniques to understand complex software elements throughout a set of related phases that spawn along the software lifecycle. During all these phases, it is important that software developers can understand the code that they are working on in a quick and clear manner, regardless of

whether they wrote the code themselves or not [107]. In this context, the graphic presentation of the system's artefacts offers significant help to facilitate the analysis and comprehension of such complex systems. Without full understanding of how the executing code behaves, the task of debugging a program, or examining its efficiency, can be very time-consuming and difficult to accomplish. Experience in software engineering and visualisation areas proves that a visual representation of a software can enhance its comprehension and decrease its development costs [162]. As a result, there is still an increasing demand for novel program understanding techniques and tools represent different aspects of software systems to graphically.

The term visualisation by itself has been defined by Card *et al.* [32] as "the use of computer supported, interactive, visual representations of data to amplify cognition." Visualisation is a powerful tool that may help users to perform different sorts of cognitive processes. These cognitive processes include: exploratory, analytical, and descriptive processes [28]. The exploratory or discovery process takes place when the user does not know what he is searching for. The analytical or decision-making process occurs when the user knows what he is looking for, while trying to determine if it exists in the data. The descriptive or explanation process takes place when the phenomenon represented in the data is known, but the user needs to have a clear visual verification of it.

In [127], software visualisation, which is also called 'program visualisation' in some of the literature, has been defined thus: "Software visualisation is the use of interactive computer graphics, typography, graphic design, animation and cinematography to enhance the interface between the software engineer or the computer science student and their programs." In other words, software visualisation creates an image of software via visual objects. These visual objects may represent components based on their structure, history or behaviour. Tools of software visualisation can assist software engineers to understand a particular software structure, logic, and behaviour. As a result, software development activities, such as analysis, modelling, testing, debugging, and maintenance, can be achieved rapidly and are less costly [151].

3.1.3.2 Software Visualisation Tools

Software visualisation approaches have concentrated on providing solutions for a variety of software engineering problems. These difficulties include range from algorithm animation, visual programming, and visualising recovered structural information from large scale and complex software systems [131]. Software visualisation tools are commonly applied to leverage the comprehension of inherently invisible and intangible software components and resources. In [18] visualisation tools have been classified based on data type taxonomy with seven data types 1- dimensional data, 2- dimensional data, 3- dimensional data , temporal and multi-dimensional data, and tree and network data. In the other hand, in the same paper, tools are classified based on tasks they seek to provide support to: overview, zoom, filter, details-on-demand, relate, history, and extract.

There are available many software visualisation tools such as CodeCrawler [89], SeeSoft [48], and Creole [98] that offer some understanding of software systems to software engineering. However, recent research in software visualisation tends to focus mostly on devising new methods, metaphors and utilities to undertake particular facets of the software understanding challenge [123]. In [74], for instance, a method that combines textures, blending, and scattered-data interpolation to visualise several metrics defined on overlapping areas-of-interest on UML class diagrams is presented. The method goal is to simplify the task of visually correlating the distribution and outlier values of a multivariate metric dataset with a system's structure. Another example of current research in the field of software visualisation is found in [113], in which a tool that creates a sequence diagram based visualisations is presented. The tool, Chrono, works in removing less related information, condensing diagram components, and providing a way for interactive exploration. The proposed tool aims to offer a solution for the problem which resulted from the diagrams created by traditional sequence diagramming tools becoming large and unmanageable when handling complicated code bases.

3.1.4 Program Slicing

Program slicing was originally introduced by Mark Weiser [174] more than twenty years ago as a "method for automatically decomposing programs by analysing their data flow and control flow. Starting from a subset of a program's behaviour, slicing reduces that program to a minimal form which still produces that behaviour. The reduced program, called a slice, is an independent program guaranteed to represent faithfully the original program within the domain of the specified subset of behaviour." Slicing complex software comprises thousands lines of code in small manageable segments to simplify their maintenance tasks. In software reverse engineering, slicing as a technique is normally utilised in analysing the legacy code to identify all parts of the code which have a certain effect on the value of a given variable. Slicing employment is not limited to the reverse engineering tasks. It is also used in other activities such as software testing, software measurement, program comprehension, and software debugging [178]. A program slice consists of all the statements of a program that may have a direct or indirect effect on the values of some variables in a set V at some point of interest p with respect to criterion C. A typical slicing criterion C has been computed as C = (p, V) [21].

Three main approaches of program slicing are available. The first approach, which was introduced originally by Weiser, is based on the iteration of dataflow equations. In this approach, slices are computed in an iterative process over a Control Flow Graph (CFG). The process computes consecutive sets of relevant variables for each node in the CFG using data dependencies [174]. The second approach is based on the information-flow relations of the programs [20]. After defining and computing several sorts of information-flow in a syntax-directed bottom-up fashion, a program slice can be easily obtained by applying relational calculus. The third and most popular type of slicing approach is based on a program dependency graph (PDG). In this approach, the dependence graphs, i.e. directed graphs, of a program are constructed, and then the algorithm produces slices by performing a graph reachability analysis from a node representing the slicing criteria. Different slices can be obtained by constructing different dependence graphs [54].

Numerous slicing algorithms have been proposed in the literature; these can be roughly classified into two main categories: static slicing [174] and dynamic slicing [3]. In static slicing, all dependence relations are statically analysed without any program execution and no assumption is made on input. On the other hand, in dynamic slicing, the dependence relations are dynamically analysed based on a given input. Since dynamic slicing focuses on the particular execution path using the provided input values, it can obtain the values of all referred and defined variables on each execution point. As a result of the slicing process, the analysis precision is better than that of static slicing, especially in software debugging and testing. There are also other types of slicing algorithms; these fall into different classifications such as quasi static slicing, simultaneous dynamic slicing, and conditioned slicing [100].

Program slicing approaches and algorithms are still an attractive topic for many researchers interested in software engineering. Many papers that have introduced new tools or methods have been published in the last couple of years. For instance, Ward and Zedan [172] introduced a program slicing unified mathematical framework. The framework places all slicing work for sequential programs on a sound theoretical foundation. This mathematical approach provides a sound basis for any specific representation. The framework uses Wide Spectrum Language (WSL) program transformation theory. The framework gives mathematical definitions for backwards slicing, conditioned slicing, static and dynamic slicing and semantic slicing as program transformations in the WSL transformation theory.

In [29], a tool for formalising the slicing of a program based on a specified scenario was presented. The slicing tool uses an XML tag set before all methods and class variables inserted by programmer during the development. The tool makes use of these tags to recognise the variables and methods linked with a particular scenario and to make a compatible program slice that implements only the scenario. Zanardini [182] , in a paper published in 2008, introduced a semantic basis for abstract slicing in which the slicing criteria are defined on properties of data, rather than concrete values. As a practical result, abstract slices seem to be smaller than standard slices. This is because commands which are relevant at the concrete level can be deleted if only some abstract

property is presumed to be preserved. Another example of recent publications in the field of program slicing is the work discussed in [128]. This paper presented a discussion on how scenarios can benefit slicing relevant program analysis, and proposed a general scenario specified program slicing method which can obtain smaller slices for program understanding.

3.2 Business Logic Modelling

This section discusses the UML concepts in the subsequent section, Section 3.2.1. In Section 3.2.2, more information is given about the UML activity diagram and its core elements. Section 3.2.2 shows the UML role in the software life cycle. In Section 3.2.4, the role of UML in reverse engineering tasks is identified.

3.2.1 UML Concepts

The Object Management Group (OMG) is an international non-profit computer industry consortium developing enterprise integration standards for a wide range of technologies serving various sorts of industries. The OMG state that "UML is a graphical language for visualising, specifying, constructing, and documenting the artefacts of a software-intensive system as well as for business modelling and other non-software systems. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components." The latest version of OMG UML, UML Version 2.1.2, was published in November 2007 [121].

As a modelling language, UML comprises three main elements: model, notation, and guidelines. Model elements are "the semantics and fundamental modelling concepts. The notation elements are the visual rendering of model elements. The guidelines are idioms of usage within the trade" [104]. It succeeds the concepts of Booch, Object Modeling Technique (OMT) and Object-Oriented Software Engineering (OOSE) by combining them into a single, common and widely usable modelling language [82].

Although UML offers a core set of notations, such as the Class diagram, for representing object oriented concepts, its use is not only limited to object oriented systems. UML provides a mechanism to extend UML to new concepts and notations beyond this core set. UML allows concepts, notations, and constraints to be customised for a particular domain [112]. The UML is intended to be a visual modelling language rather than a visual programming language. In spite of the fact that UML tries to standardise a modelling language, it is not dependent on any particular programming language nor it is reliant on a specific software development process[82].

Although diagrams are not the only part of UML, they are considered to be the most important part as they perform the graphical representation of a system. The UML diagrams can be classified into two main groups: structure diagrams and behaviour diagrams. The two groups of diagrams model the static and dynamic aspects of the software respectively. In one hand, structure diagrams define the static architecture of a model representing physical elements and conceptual elements. They are used to model classes, objects, interfaces and physical components in addition to the dependencies and relationships among these elements. Behaviour diagrams, on the other hand, describe the dynamic model, representing the interactions and the activities within the system or with the users or environment [22]. UML2.0 supports thirteen views, as compared with the nine views in the previous version. Since it is difficult to describe a system completely, it is meaningful to provide multiple views [88].

UML makes use of six different types of diagrams to describe the static system structure on six different views: Class diagram, Object diagram, Package diagram, Component diagram, Composite Structure diagram and Deployment diagram. For modelling behaviour aspects, UML uses the following behaviour diagrams: Use Case, Activity diagram, State Machine, Communication, Interaction Overview, Timing diagram, and Sequence Diagram [112].

3.2.2 Activity Diagram

Activity diagrams as defined by OMG are the diagrams derived from various

techniques to illustrate workflows visually [121]. An activity diagram depicts the behavioural (dynamic) point view of a system. It looks quite similar to a flowchart diagram showing flow of control from one activity to another.

An activity can be described as a state of performing an action: either a real world process like typing a report, or the execution of a software function, such as a method on a class. In other words, since activity diagrams are similar to state diagrams, an activity can be described as "an ongoing non-atomic execution within a state machine" [22]. Activities eventually bring about some action, which is "composed of executable atomic computations that result in a change in state of the system or the return of a value. Actions include calling another operation, sending a signal, creating or destroying an object, or triggering another pure computation" [58].

The activity diagram describes the sequencing of activities, with support for both conditional and parallel behaviour. Activity diagrams may be used to visualise, construct, specify, and document the behaviour of a group of objects, or they may be used to model an operation's flow of control [112]. To put it more simply, "an activity diagram in its basic form is a simple and intuitive illustration of what happens in a workflow, what activities can be done in parallel, and whether there are alternative paths through the workflow" [82].

Activity diagrams are read from top to bottom, and a basic activity diagram may include the following elements, as shown in Figure 3-1 [58]:

- **Initial node:** indicates where the workflow begins.
- **Control flow:** an arrow showing the direction of the workflow.
- **Activity:** indicates a step in the process.
- **Decision**: indicates a choice for a workflow to proceed along one of a number of possible paths, according to the guard conditions.
- **Merge:** when a number of flows lead to the same activity and the intention

is that any of the flows would lead to the activity. Rather than terminating them at the same activity, they terminate them at a merge, and draw a flow from the merge to the activity.

- **Guard condition**: A condition attached to a control flow. When the guard condition is true, workflow may flow along the control flow.

- **Fork and Join**: a fork indicates the point after which a number of activities may begin in any order. A join indicates that workflow may commence only once the parallel activities that flow into it have all been completed.

- **Final node**: indicates the end of the process.

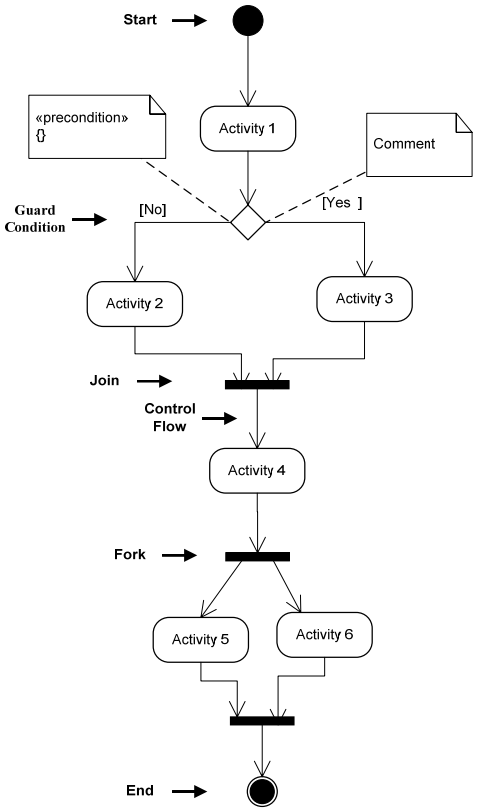

Figure 3-1: UML Activity Diagram Elements

3.2.3 UML and Software Life Cycle

UML is typically used during the design phase of the software lifecycle to depict graphically different aspects of the system's high-level architecture. In this respect, UML diagrams can serve as a universal communication medium used by all members of the software development team [82]. Because UML is normally used in software development at the very beginning in most software projects, UML can also be used in other phases of the software lifecycle. For instance, UML diagrams can be automatically recreated from a source code by way of reverse engineering techniques [180]. The recreated diagrams in this case are regarded as a type of graphical documentation suitable to be employed by software engineers in a system maintenance or reengineering project. In order to integrate disparate systems, developers must fully understand the systems being integrated. Not only must they understand the software structure of the system, but they must also be able to follow the data and control flow and the execution of events [137]. The system understanding is a mandate, because software engineers need to know where the changes should be made, and the impact of the changes are limited and predictable.

UML is considered to be a suitable tool for modelling legacy software because of the features it encompasses that satisfy the needs of legacy systems modelling. Such properties include - but are not limited to - the following:

- Simple navigation with minimum disorientation: UML is structured and includes features to aid the user in navigating the modelling.
- Well-structured presentation: UML diagrams each have their own characteristics as a theoretical foundation. They are used graphically in modelling.
- Varying levels of detail: Details, abstraction, information content and type of information change to suite the users' interests in the presentation of the UML diagrams.

- Desirable integration: UML is able to link the modelling and the original information it represents.

3.2.4 UML and Reverse Engineering

There are available a large number of reverse engineering approaches and tools, as discussed earlier, which serve specific stages of the overall reverse engineering process. In [110], a selected research agenda for code and data reverse engineering in addition to research strategies for tool development and evaluation has been discussed. In a similar manner, numerous software tools are available that provide various levels of support for and use of UML in different categories, including UML Drawing Tools, UML Code-Centric Tools and UML Framework Tools [141].

Research in exploiting UML in reverse engineering context resulted in the development of a number of reengineering tools and methods. CPP2XMI [85] is a reverse engineering tool which allows the extraction of UML class, sequence, and activity diagrams in XMI format from a C++ source code. The use of the utility is proposed as a part of the validation and verification chain of software systems. By using the tool, the static structure and the dynamic behaviour of a system are extracted from the source code and represented in XML Metadata Interchange (XMI) format. The derived model of the system is further analysed for properties such as complexity and soundness. In the same domain, Tunella [161] also proposed a technique for the automatic extraction of UML interaction diagrams from a C++ code. The extraction algorithm is established on a static and conservative flow analysis. The analysis approximates the system's behaviour at any execution and for any potential input.

Many researchers have contributed to the field of reverse engineering of software systems by introducing new tools, methods, and approaches of extracting UML diagrams from a source code. What follows is merely an example of such efforts in the last four years.

Green [170], during an ongoing project at the University of Buffalo, devised a UML class diagram plug-in for the Eclipse IDE, which includes features such as live

round-tripping and a customisable set of relationships. Round-tripping means that the tool supports both forward and reverse engineering. Green as a plug-in to Eclipse has the ability to utilise the integrated development environment. Moreover, it can maintain a real-time synchronisation between the recovered class diagrams and the Java source code. Briand *et al.* [91] proposed a methodology and instrumentation infrastructure toward the reverse engineering of UML sequence diagrams from dynamic analysis. The proposed approach is formally defined using meta-models and consistency rules. The instrumentation is based on aspect-oriented programming in order to reduce the effort overhead usually associated with source code instrumentation. It is a tool for automatically constructing UML models from the source code of existing web applications. Through analysing the source code and interacting with the web server, WebUml is able to generate class diagrams and state diagrams for the web application. The tool uses static analysis for the source code. In addition, it performs the dynamic analysis by applying mutational techniques using the server side execution engine.

3.2.5 BPMN and UML

In this section, an overview about BPMN concepts is provided in Section 3.2.5.1. Then in Section 3.2.5.2, the relationship between UML and BPMN is described. In addition, in the same section, several transformation techniques between UML and BPMN proposed in the literature have been described.

3.2.5.1 BPMN Concepts

Business Process Modelling Notation (BPMN) is a standard notation which is easily understandable by business users. The business users include “the business analysts that create the initial drafts of the processes, the technical developers responsible for implementing the technology that will perform those processes, in addition to the business people who will manage and monitor those processes” [120]. As a result, BPMN bridges the gap between the business process design and process implementation by providing a standard notation. In addition, a second goal of BPMN is to make certain that XML languages designed for the execution of business processes,

such as Business Process Execution Language for Web Services (BPEL4WS) and Business Process Modelling Language (BPML), can be expressed visually with a common standard notation [177].

BPMN was developed by the Business Process Management Initiative (BPMI), and has been maintained by the Object Management Group since the two organisations merged in 2005. The current version of BPMN is 1.1, and the last version released in 2007 is BPMN 1.2 Beta 3.

The OMG BPMN specification document BPMN [120]defines a Business Process Diagram (BPD), which is a flowcharting technique developed for creating graphical models of business process operations. The BPD was designed in such a way as to accomplish two conflicting goals. First, the diagram needs to create a simple mechanism for creating business process models easily understandable by non-technical users. Second, the diagram should be able to handle the complexity inherent to business processes, and can be naturally mapped to business execution languages. To manage these two contradictory requirements, the graphical aspects of the notation of BPD have been classified into specific basic categories. Additional variations and information can be added to the basic categories of elements in order to support the complexity requirements without the need to change the simple look and feel of the diagram. The four basic categories of elements as shown in Figure 3-2 are as follows:

- Flow Objects: the main graphical elements to define the behaviour of a Business Process. The three Flow Objects are: Events, Activities and Gateways.

- Connecting Objects: There are three ways of connecting the Flow Objects to each other or other information: Message Flow, Sequence Flow and Association.

- Swimlanes: there are two ways of grouping the primary modelling elements through Swimlanes: Pool and Lane.

- Artefacts: they are used to provide additional information about the Process. The current set of Artefacts include Data Objects, Group and Annotation.

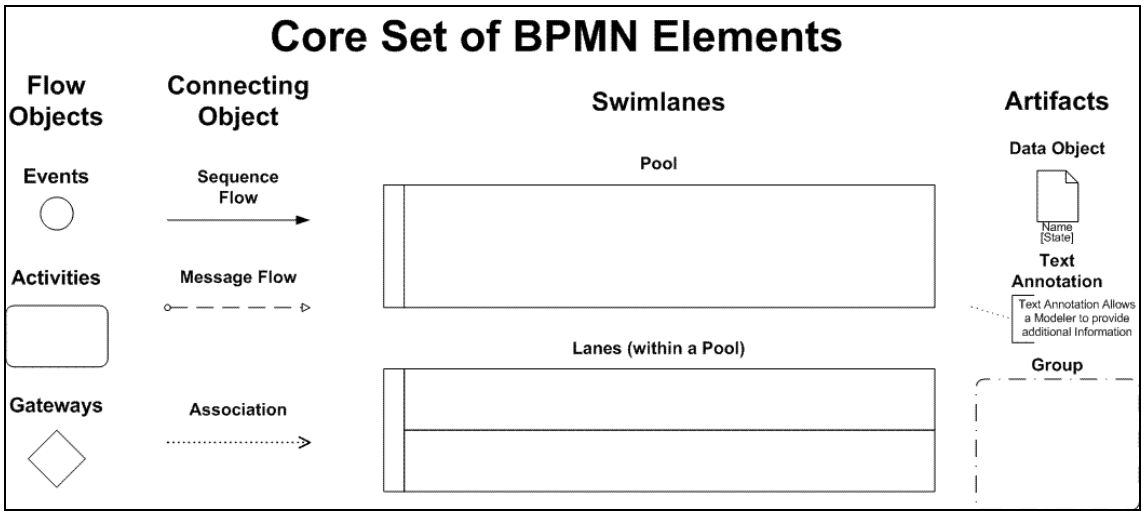

Figure 3-2: Core Set of BPMN Elements [120]

3.2.5.2 BPMN and UML

Liew *et al.* [97] presented a set of transformations to produce a particular set of UML artefacts automatically from the specifications of a business process. The generated UML models can be refined further to produce models sufficient for code generation based on the Model-Driven Architecture (MDA) approach. In order to achieve this objective, the business processes are captured, using BPMN first. Then UML diagrams are compared with BPMN at the level of meta-model to the meta-model. After that, the related information needed for comprehensive generation of models is assessed. In the case of the generated UML model not being provided via the existing workflow language, the model is extended through annotations to provide adequate semantic detail. The focus of proposed transformations was placed specifically on mappings between the BPMN and UML (v 1.5) activity, use case, collaboration, and deployment diagrams.

Okawa *et al.* [118] proposed a method for configuring information systems that links BPMN and eXecutableUML (xUML) techniques. The method proposed makes use of

BPMN in order to enhance the work process, transforms it to UML, and then expands it to xUML. In the method the transformation of model elements from BPMN to UML is determined through modelling guidelines. The xUML model created by the completed simulations was used to do automatic generation for the code. The model was shown to be effective by verifications of the dynamic operation of a system implemented from this code. A prototype information system for ticket reservations operating over a network was created to evaluate the proposed method.

IBM demonstrated in [64] how to transform UML activity diagrams automatically into WebSphere Business Modeler processes to be able to reuse the UML diagrams in a more business-oriented fashion. The transformation of UML activity diagrams to Modeler processes is performed using the Rational platform with the Model Transformation Framework tool from AlphaWorks. The Rational platform captures the UML models using XMI, which is an XML representation of the models. Modeler provides an XML schema that defines the structure of an XML file that one can import as a business process along with associated artefacts. Thus to convert from UML to Modeler, the user just has to convert XMI elements to their respective Modeller equivalent. The availability of this Rational to Modeler transformation facilitates round-trip business engineering between the Rational platform and WebSphere Business Modeler.

In the same domain, Cesare and Serrano [34] proposed a modelling framework that maps the constructs used in information systems (IS) models to those used in business process (BP) models so that both BP and IS analysts consider their organisational views in their designs. The mapping framework is based on the business process simulation (BPS) and UML. The information conveyed in UML models can be used to create BPS models and vice versa. In addition, Shishkov and Dietz [139] proposed SDBC is based on four fundamental elements. The first is an integrated view over business process modelling and software specification. The second is the Dynamic Essential Modelling of Organisations (DEMO) transaction theory. The third is the principle of component-based system development. The fourth is the re-use of requirements. The implementation of the proposed approach was carried out using UML.

3.3 Business Logic Extraction

In this review of recent approaches, the techniques and tool used in extracting the business logic from the source code of existing systems is provided. Section 3.3.1 reviews the static-analysis based approaches. Section 3.3.2 reviews the dynamic-analysis based approaches.

3.3.1 Static-Analysis Based Extraction

Sneed [142] presented an approach to recover the business logic embedded in a legacy COBOL application. The approach was composed of four steps. The first is the procedural code restructuring, used in order to facilitate the second step which is code slicing. This restructuring of the program code is carried out automatically. The second step of the approach is slicing the code into sub programs, each processing a discrete business rule. Unlike the first step, the code slicing is assumed to be done manually utilising human intelligence to recognise the beginning of the processing of a certain use case. In the third step, the business modules are analysed using a multi-view analysis. All of the extracted partial programs are presented to an automated documentation process. This automated process generates five views on each of the partial programs. The final step of the approach is the integration of the generated five disjointed views at the transaction, subsystem and system level into single unified business logic documentation. In this step all the views are analysed and stored in a relational database from which overall graphical documents describing the data decision and procedural flow of each program slice are created.

Zou [185] proposed a framework for model-driven business process recovery. The framework captures the system functional features representing a business process. In the framework, business logics are identified from the source code through a static tracing method and a number of heuristics. The approach aims at extracting as-implemented workflows from the source code to high-level business process entities. The approach sets up links between the business domain entities such as tasks and decisions and the implementation domain entities such as conditional constructs and

methods. Then in the framework, the business workflow entities and the implementation domain entities are analysed. This analysis permits the construction of an abstract business process model for e-commerce applications. The trace records were generated through static traces which were later refined via seven refinement rules; an example of such rules is filtering out the Java utility code. The heuristic rules that are used to determine whether code features can be extracted as task workflow entities were focused only on IBM e-commerce applications. The result is represented in an XML format; then it automatically converts the XML represented as implemented workflow into a graphical representation. Even though, in this work, an Eclipse plug-in tool was developed to do an automatic extraction for the as implemented workflows from Java-based Web applications, the role of software engineer remains essential for comparing both workflows and mapping them to each other.

As an extension to a previous work, Zou [184] proposed another automatic approach that captures business processes from the source code of ecommerce applications. The approach also refines the recovered business processes using control structure information in as-specified workflows. The approach runs a comparison between the structural features of both types of workflows, as-specified and as-implemented workflows. The comparison is done using an intermediate behavioural model. There are two main stages in the recovery process: information parsing and business logic refinement. First, in the information parsing stage, a code parsing is executed, and the as-implemented workflows are generated using code heuristics. Then, the recovered as-implemented workflows abstraction level is elevated through filtering out programming specific features. Second, in the business logic refinement stage, the behavioural models for as-specified and as-implemented workflows are compared. This comparison is achieved using a structural comparison algorithm. The algorithm is used to recognise the structural resemblance between both types of workflows. In addition, the algorithm is also used to associate code blocks with workflow tasks in the as-specified workflows.

Walkinshaw *et al.* [168] developed a tool-supported approach which extracts the source code that is relevant to a user-level function in an object-oriented system. Slicing

and call graph analysis are used in the tool to return a trail of the user-level function relevant methods. This is achieved based on identifying a set of landmark methods that must be executed in a given user-level function. The outcome of the tool is a graph that contains a reduced set of edges that are particularly relevant to the execution of this set of landmark methods.

Hung [78] introduced a method to recover business processes for the three-tier architecture systems. The automatic recovery is achieved by identifying the business data and business policies in the source code. The three-tier architecture has high maintainability because the components of three-tier are properly divided and the interface between these components is well formulated. The approach employs forward and backward tracing to identify the exact location of the business logics. To generate the business process, static tracing is used to determine the communications between the business logics and policies. The definition of business logic in this approach is a requirement on the manipulation of data expressed in terms of the business application domain, whereas a business policy is used to specify the rules and conditions on the time and location in which the business logic should be executed. In this method, the business data is identified from the database operations. On the other hand, the business policies are identified from the behaviours of the objects and the calculation of the outputs. Together the identified business data and policies can locate the business logics in a source code in an automatic manner.

In a recent work published in 2007, Hung [79] presented another technique to recover workflows from three-tier e-commerce software systems. The approach and the developed prototype tool identify the structure of workflows through the following process. First, beginning from an initial page in the user interface (UI) of the system, the process starts by tracing the navigation flow of the UI through the different UI pages. Second, the source code of the components that implement the functionality provided through the UI pages is analysed. Then, information from the controller code is recovered. The controller code integrates back-end components and databases in the three-tier architecture. The generated workflow is depicted using a hierarchical view. The view hides minor business processing steps. Furthermore, the generated workflows

can be imported into the IBM WebSphere Business Modeller to inspect its low-level processing tasks.

3.3.2 Dynamic-Analysis Based Extraction

In [149] Suenbuel and Shan proposed an approach and experimental results for the extraction of high-level business process models from running enterprise systems. The approach aims to inspect the process structures that underlie the tasks being performed throughout the execution of system components. The complicated business processes are extracted using just a simple business process scheme and a few transformation and refinement rules. The approach is organised into two phases: event-log analysis and data abstraction. In the first phase, events are recorded by the running system and written into log files. These log files are analysed later by the log-file analyser. The event-log analyser filters out the raw log messages in order to separate relevant from irrelevant information. In the second phase, event pattern are mapped to business process names using the process mining component. Then the runtime structures are recorded in terms of these business processes. Based on this approach, a business process visualisation tool has been implemented. The tool automatically generates a Petri-Net like diagram from the collected event data.

Similarly, Aalst et al. [2] developed techniques for the purpose of discovering workflow models by using workflow logs. Workflow logs contain real-time information about the workflow process as it is being carried out. A new algorithm is provided by this approach to derive a process model from the workflow log and then it is represented in the form of a Petri net. The α-algorithm is capable of mining a large and relevant class of workflow which is represented by what is referred to as a SWF-net. These WF-nets are a type of Petri net designed for the purpose of workflow processes. The algorithm takes advantage of the fact that for numerous WF-nets there are two tasks connected providing their causality is detectable upon inspection of the log. Results have demonstrated that α algorithm-based process mining, in addition to using tools such as Little Thumb and EMiT, is possible, at the least, for structured processes.

Walker *et al.* [167] provided a method for encoding dynamic trace information thus allowing it to become tractable and efficient to manipulate a trace from a diverse range of architecture-level perspectives. Traces are comprised of basic object-oriented execution events. Additionally, the suggested encoding scheme may be applied to encode events that are exchanged between components, remote process interactions, and other execution events that take place more than once. Two tools have been developed to implement the technique. The first tool is able to visualise dynamic information taken from the object-oriented system, thus the developer is able to analyse the execution of a system whilst it is off-line. There are two main parts in the visualisation, the first of which part is a series of pictures, which are put in order temporarily, and subsequently the pictures provide detailed information about the corresponding points found in the execution. The second part provides a summary of the execution to the corresponding point. The second tool allows the paths between the architectural components to be extracted from the trace data. The path query tool extracts all of the paths beginning in one architectural component and then finishing at the start or entry of a second architectural component within a provided trace and specified mapping.

Turner et al. [163] provided a different approach to business process mining by way of using a Genetic Programming (GP) technique in combination with a representation which is graph-based. Techniques that are genetic-based used for process mining are resilient to noisy data; additionally they have the ability to produce new sub-process combinations derived from a provided set of data. The graph-based representation provides flexibility when analysing process flowchart structure as well as making the mining of complex business processes from event logs, that are incomplete, possible. Their approach has five stages, applied in the operation of a GP process mining algorithm, which are explained hereafter. The first stage is to read the event log containing data related to the process execution. The second stage involves a calculation of the dependency relations between the activities found within the process log, which are based on a set of heuristics. In the third stage individuals are built from the relationships of the event log found in the second stage. The fourth stage consists of measuring an individual's fitness. Finally the fifth stage which involves parsing each

process trace found within the event log against each individual which has been generated for the process mining algorithm by task fashion.

Foo *et al.* [57] introduced a technique for recovering business processes from e-commerce applications and for verifying the recovered processes using dynamic analysis which trace the execution of processes. The approach recovers usage scenarios from the execution logs and uses them to verify the business processes recovered using static analysis. User interface (UI) design patterns are utilised to identify tasks with appropriate granularity, and to separate different business processes. To detect each business process, the traces produced from all users' requests made during a session are analysed, particularly those that are executed to react to user's requests. To exclude non-business logic from the usage scenario, a criterion to guide the insertion of instrumentation code into the different tiers has been defined. In addition, in order to create a complete usage scenario of a business process, the recorded information from each tier is merged and sorted by access time. Depending on the results of matching the tasks identified from both techniques (static and dynamic) in sequential order, it is possible to determine whether a recovered business process is complete or incomplete.

As an extension to the work described earlier, the same authors, Foo *et al.*, developed a tool named Process Explorer. By employing static and dynamic analysis techniques the process explorer tool [65] automatically recovers business processes found in e-commerce systems comprising of three tiers. Three main features are offered by the process explorer tool. The first feature is the traceability between the code and the recovered processes. If the lines in the code that correspond to a process element are tracked, it makes it possible for developers to locate the code that corresponds to a specific task, business data or a control flow element. Additionally, a developer may also refine the recovered process by editing the highlighted blocks of code manually. The second feature of this tool is that it provides consistency when displaying the recovered processes of the sources. This consistent display is in the form of tree diagram which allows the business processes to be represented hierarchically. Nodes are assigned icons in the tree representation which denote the control flow elements, business data and tasks. Finally, the third feature, this feature provides integration with

the IBM WebSphere Business Modeler (WBM) business process modelling tool. The WBM allows viewing of the working processes so that a business analyst can search through the recovered processes in order to provide more appropriate names for the tasks.

3.4 Summary

In this chapter, the research studies related to the proposed reengineering framework are reviewed. Since the proposed reengineering framework focuses mostly on the program understanding stage, most of the concepts and studies covered in this chapter are related to program understanding and reverse engineering methods and concepts.

The first section starts by providing the definition of program understanding mentioned in the software engineering literature. In addition, the section points out the differences between program understanding and other reengineering concepts, such as reverse engineering and reverse specification. Program understanding begins with the source code, while reverse engineering can start at a binary and executable form of the system or at high level descriptions of the design.

Then the section provides a synopsis about the different types of elicitable information along with their supporting techniques that aim to provide program comprehension at any level of abstraction. Examples of these views are statistics view, outline view, logic view and algebraic view.

After that, the software visualisation concepts along with the available supporting tools are discussed. The tools reviewed included CodeCrawle and SeeSoft.

The last part of the program understanding section discusses program slicing. The different program slicing approaches, algorithms, and tools are presented. Several of the recent research studies in the field of program slicing have been also reviewed.

The second section of the chapter focuses on the business logic modelling. The section starts with identifying the importance of UML and the role of UML in the

software life cycle, and, more specifically, in the reverse engineering tasks.

In the last part of the second section, the BPMN concepts are discussed. In addition, the relationship between UML and BPMN is described, along with the available transformation approaches proposed in recent research studies in the field.

The third and last section of the chapter provides a review of the up-to-date approaches in extracting business logic from the source code of existing software systems. The reviewed studies are divided into two categories: static-analysis based approaches and dynamic-analysis based approaches.

Chapter 4

Proposed Approach

Objectives

- To introduce the four stages of the reengineering framework.
- To give an overview of the steps within each stage.
- To highlight the focus of this research in each stage of the framework.

Contents

4.1 Framework Overview

This section presents an overview of the framework‘s four stages as depicted in Figure 4-1

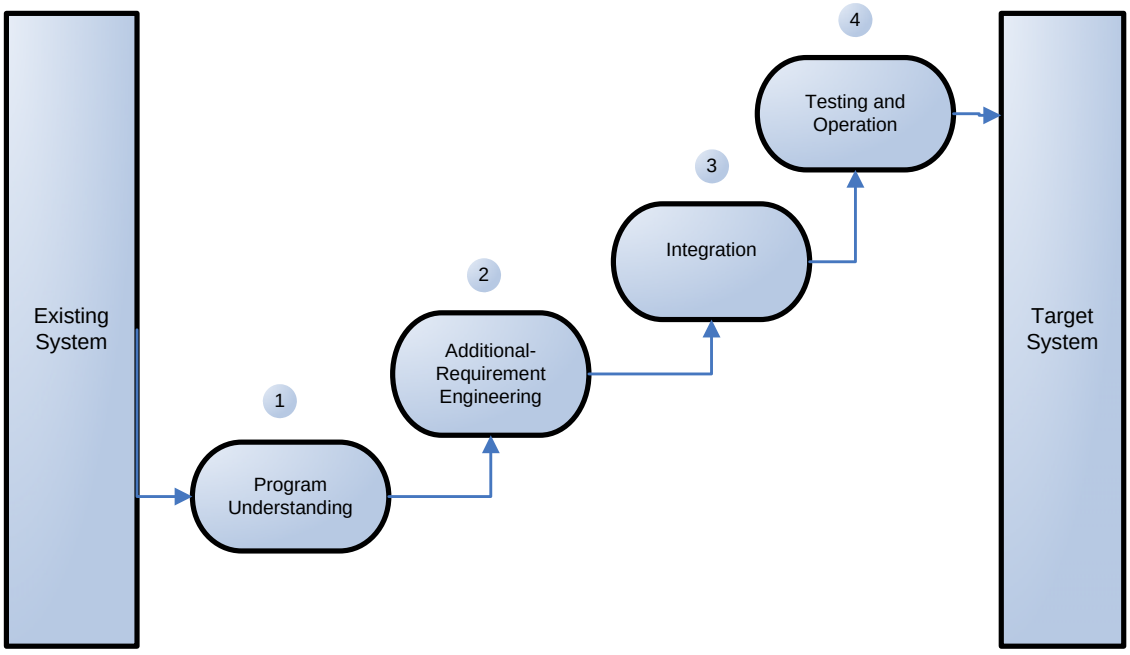

Figure 4-1: The Four Stages of the Reengineering Framework

The term framework is widely used with very different meanings. The following definitions from dictionaries, as well as the synonyms, provide a basic understanding of the meaning of framework in English.

***Framework:* 1** *a frame or structure composed of parts fitted and joined together.* **2** *a skeletal structure designed to support or enclose something.[13]*

***Framework*: 1** *a set of assumptions, concepts, values, and practices that constitutes a way of viewing reality.* **2** *a particular set of beliefs, ideas, or rules referred to in order to solve a problem.[52]*

The framework proposed in this research can be regarded as a roadmap for software engineers aiming to incorporate the new features introduced by ubiquitous technologies into an existing system as explained in the first dictionary definition "…composed of parts fitted and joined together." The framework consists of stages with internal steps fitted and joined to accomplish the goal of the evolution process. The overall steps begin with understanding the existing system using the reverse engineering techniques and end with the operation and testing of the target reengineered system through forward engineering methods.

The framework follows a systematic step-based approach in which its steps can be generally classified into the following four main stages. The four stages of the framework as shown in Figure 4-1 are:

1. **Program Understanding Stage:** in this stage, code reverse engineering techniques are used for program comprehension and architecture recovery. First, the business logic buried in the source code is extracted and represented in BPMN. Then, the software system architecture is recovered. Both steps are performed to facilitate the third proceeding stage in the framework which is the integration stage.

2. **Additional-Requirement Engineering Stage:** in this stage, the new business logic which resulted from the new features offered by the introduction of ubiquitous technologies will be determined and modelled. A scenario-based requirements engineering technique is used in this stage. The new business logic is modelled using BPMN.

3. **Integration Stage:** in this stage, integration is carried out based on a developed integration algorithm. The integration algorithm relies on a comparison technique at a model level. The extracted business logic and the new determined business logic are used to identify 'what' and 'where' the needed changes in the system should be made. In addition, the required hardware handlers' components are also integrated with the system in this stage.

4. **Testing and Operation Stage:** in this stage, the integrated system is tested in order for it to be put in the operation phase. Since this stage is out of the focus of this thesis, only general guidelines are discussed. Different testing strategies can be developed to test the new system's features provided after the introduction of the ubiquitous technologies, and before it can be put into operation.

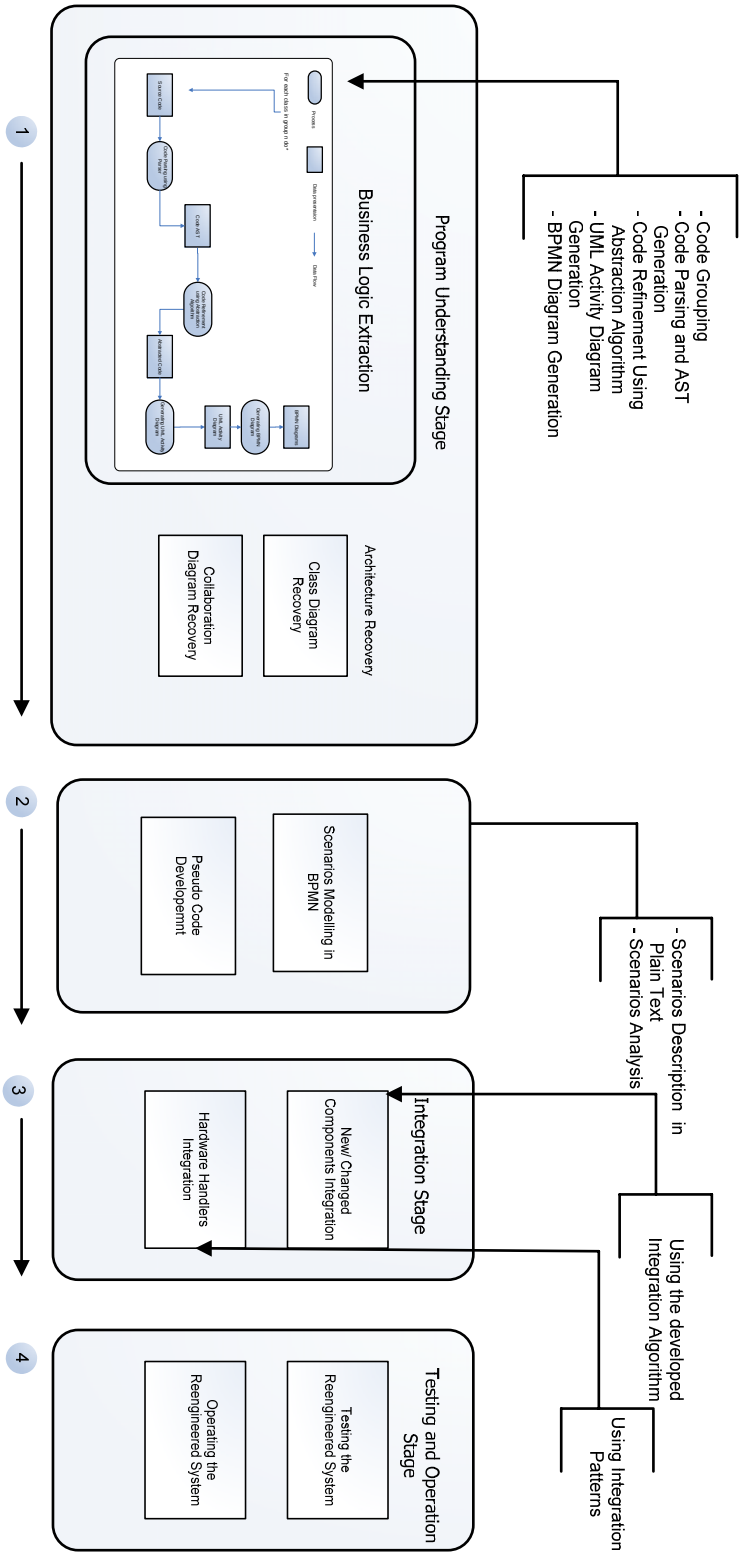

Figure 4-2: The Reengineering Framework Stages and Steps

4.2 Program Understanding Stage

The greatest part of the software reengineering process is devoted to understanding the system being maintained. Program understanding has the ultimate goal of enabling the comprehension of the underlying functional and data concept of software. Program understanding as defined by Rugaber [38], is "..the process of acquiring knowledge about a computer program. Increased knowledge enables such activities as bug correction, enhancement, reuse, and documentation."

Software visualisation can help software engineers to cope with the complexity of program comprehension by displaying programs, program artefacts, and program behaviour.

Reverse engineering available techniques assist in software comprehension through exploiting the source code as the main source of information about certain software. The application of such techniques allows the extraction of a set of useful views provided to software engineers in the form of diagrams.

As shown in Figure 4-2, the stage of program understanding is represented by a big box. This box will be decomposed further in the next chapter, Chapter 5, into a number of steps. In general, this stage has two objectives.

- The first objective is to comprehend the functional aspect of the existing software via extracting the business logic imbedded in the source code and representing it as a diagrammatical view, BPMN.
- The second objective is to recover the structure of software system using available software architecture recovery methods and tools in order to recover the logical view of the software.

4.2.1.1 Business Logic Extraction

The employment of ubiquitous technologies in certain organisations affected how

their business processes are carried out was explained. This change of business process is known as business process reengineering. As a result of the business processes reengineering, the business logic implemented in the employed software applications also needs to be changed accordingly. In order to facilitate the process of re-engineering the existing software to comply with the new business logic, there is a necessity to understand the business logic already implemented in the existing software systems.

In this step, a new approach has been proposed which makes use of various reverse engineering tools and approaches. The extraction approach considers the source code to be the only available documentation for the system. Furthermore, the extracted business logic is represented in a manner not only conceivable by software engineers, but also understandable by business-oriented professionals. The result of this step is a business logic represented in BPMN.

4.2.1.2 Architecture Recovery

Software architecture provides higher level views than components, which aim at designing systems with coarser-grained elements and their overall interconnection structure. It is not only the starting point for system design and development, but the desired outcome of the reverse engineering process [16]. Architecture recovery aims to produce the highest level of abstraction of a legacy system, which is built on the understanding of individual procedures, modules and components. In this step, the well-known “4+1” view model of software architecture introduced by Kruchten [88] has been adopted. From the five views defined in the “4+1” model, the logical view is recovered.

A static analysis is carried out on the source code of the existing system with the aim of extracting information about the whole system and its overall structure. The design and implementation plans of classes and class hierarchies along with their methods and attributes are recovered using available approaches and tools. In addition, the relationships between the classes, represented in a class diagram which can be generated automatically using available tools, are also recovered at this stage.

4.3 Additional-Requirements Engineering Stage

Requirements engineering as explained by Thayer [152] provides: ".. the appropriate mechanism for understanding what the customer wants, analysing need, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification, and managing the requirements as they are transformed into an operational system."

From the above statement we can understand the steps which need to be executed in the process of requirement engineering. These steps are the following:

- Elicitation
- Analysis and negotiation
- Specification
- System modelling
- Validation
- Management

Sommerville [144] classifies software requirements into functional requirements and non-functional requirements. The functional requirements type specifies something that the delivered system must be able to do. Non-functional requirements, on the other hand, specify how well the system performs its functions; such requirements include availability, testability, maintainability, and ease-of-use. There exist other classifications of software requirements, such as the classification introduced by Buede [27].

Sommerville [144] also classifies requirements, based on the level of details and type of language and notation used in the description of software requirements into two types: user requirements and system requirements.

- User requirements which are sometimes called stakeholders requirements, mean the high-level abstract requirements.

- System requirements on the contrary, mean a detailed description of the system's function, services, and operation.

As show in Figure 4-3, the stage of requirements engineering is composed of four sub-steps: scenarios description, scenarios analysis, scenarios modelling and pseudo code development. This stage will be elaborated upon in Chapter 6, describing the internal steps within this stage. In the additional-requirements engineering stage, the focus is on the elicitation of the functional requirements in a user requirement form. In other words, the new or modified functions of the system which can be introduced as a result from the use of ubiquitous computing technologies are determined and modelled.

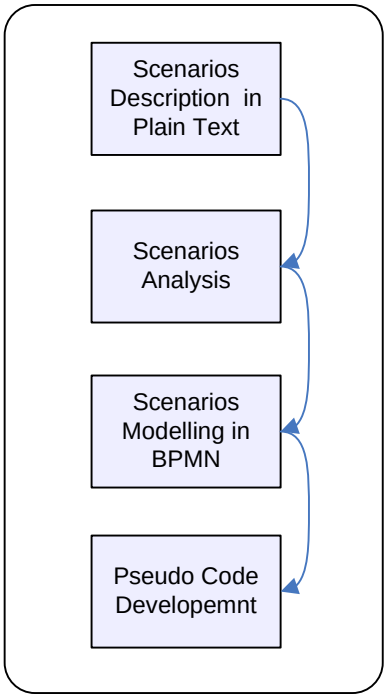

Figure 4-3: Additional-Requirements Engineering Stage Steps

In the proposed framework, the scenario-based method is selected for carrying out the requirement elicitation process. Scenarios are examples of interaction sessions, and consist of descriptions of sequential actions which relate to real-life examples rather

than abstract descriptions of the functions [150]. The reason for selecting a scenario-based approach as a requirement elicitation method is explained in detail in Chapter 6. The function requirements are modelled using BPMN which is currently a *de facto* standard notation that depicts the steps in a business process. The BPMN notation is understandable by all system stakeholders - including the business analysts who create and refine the processes - and the technical developers responsible for implementing the processes.

In this stage, only the functional requirements are considered. Other forms of requirements such as data requirements and quality requirements are beyond the scope of this research. Moreover, only the elicitation step of the requirements engineering process is described. Other steps in the requirement engineering process are also not discussed. Therefore, Chapter 6 will describe both the elicitation of functional requirements using a scenario-based method and the modelling of these requirements using BPMN.

The aim of this stage is to determine and model the new functions or business logic which resulted from the new features offered by the introduction of ubiquitous technologies. The result of this stage, which is the functions or business logic modelled in BPMN serves as the first input for the integration stage alongside the extracted business logic acquired from the first stage of the framework.

4.4 Integration Stage

In the software sphere, integration is part of the software development lifecycle. The integration process aims to combine the developed software components and ensure that these components can work together [144].

Much work has been done by researchers in the area of software integration. Different methods and also tools have been developed to carry out the integration process. These integration methods vary in their purposes and targets, depending on the software types, software development methodologies and software architectures. For

instance, in the object-oriented software development, many approaches and tools have been introduced in the literature; the approach described in [103] is one example. Likewise, integration in the component-based software development (CBSD) approach has attracted a significant number of researchers [45, 95, 139].

The integration stage is composed of two steps, as depicted in Figure 4-4. These two steps are:

- **First:** Integrating the new/changed software components.
- **Second:** Integrating the hardware handlers of the ubiquitous computing technologies

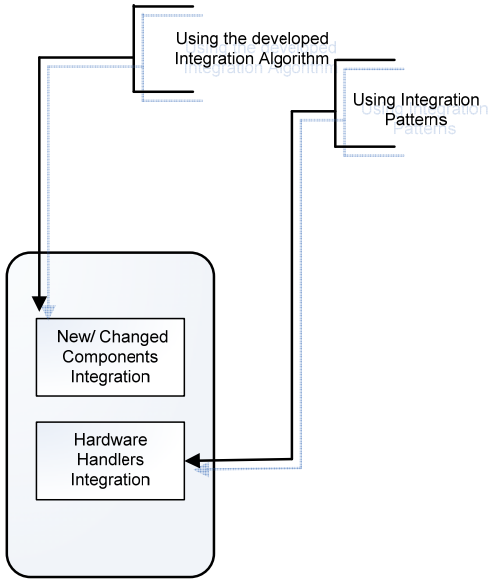

Figure 4-4: Integration Stage Steps

4.4.1 Integrating New/Changed Components

The integration of new/changed system components is accomplished using a devised integration algorithm. The developed integration algorithm relies on comparing the extracted business logic and the new determined business logic, both of which are modelled in BPMN. The comparison can be done automatically or manually. Automatic

comparison can be achieved through the use of available graph matching techniques, whereas manual comparison can be performed by a software engineer with sufficient experience in the system domain.

The result of this comparison leads to either adding or modifying a component in the existing system. This addition and modification have a reflection on the existing software architecture. Hence, the main system architecture is adjusted accordingly. The integration algorithm is discussed in Chapter 7, Section 7.2. The flow of the integration process is shown in Figure 4-5

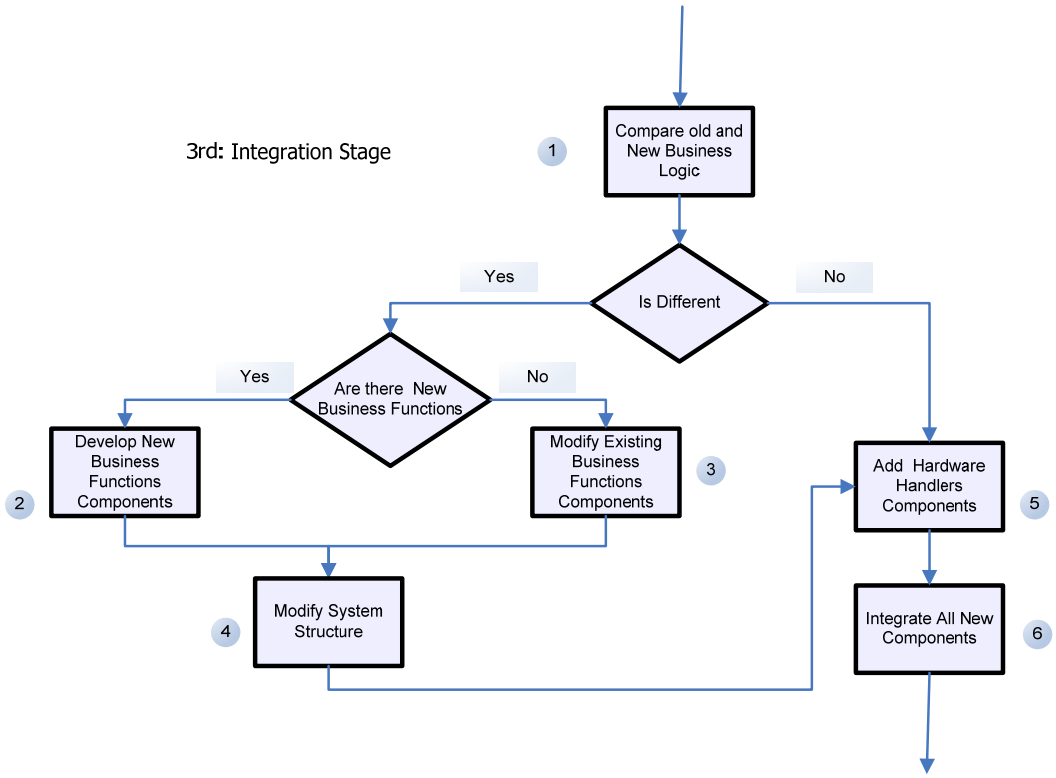

Figure 4-5: Flow of Integration Process

4.4.2 Integrating Hardware Handlers

The second objective of this stage is to add the required hardware handlers' components to the system. The hardware in this case can be a sensor or an RFID. These technologies can be integrated to the reengineered system either through using

integration patterns as described in Chapter 7, Section 7.3.

4.5 Testing and Operation Stage

Regardless of the software process model used, testing is an essential activity which must be implemented in any software project in order to reveal defects in software and to ensure that the software requirements have been met. Over the years, a rich variety of testing methods, approaches and tools has evolved. In software engineering, - similar to any engineering field - the engineered product, which is the software in this case, is tested using one or both of these strategies. The first is black box testing, which is sometimes called functional testing. In white box testing, the software is tested using validation techniques to ensure that the software performs the required functions that it has been designed to perform with no knowledge of the internal logic. The second is white box testing, which is sometimes called structural testing. In this type of testing, the internal logic of the software is tested using verification techniques. To put it more simply, the software testing process in general aims to achieve two goals: validation and verification. Validation means that the software has met its requirements, while verification means that the system had no faults or defects.

Testing is still a hot topic in software engineering research. As a result, there exist in the literature many classifications and levels for the testing process. Figure 4-6 shows an example of these classifications and levelling. For instance, in some literature, testing has been divided to integration testing and unit testing. Furthermore, within each type there are several testing activities. In other literature, system testing and component testing are referred to as white box testing and black box testing. Here, the classification of testing as integration testing and unit testing has been adopted, and will be discussed in the following sections.

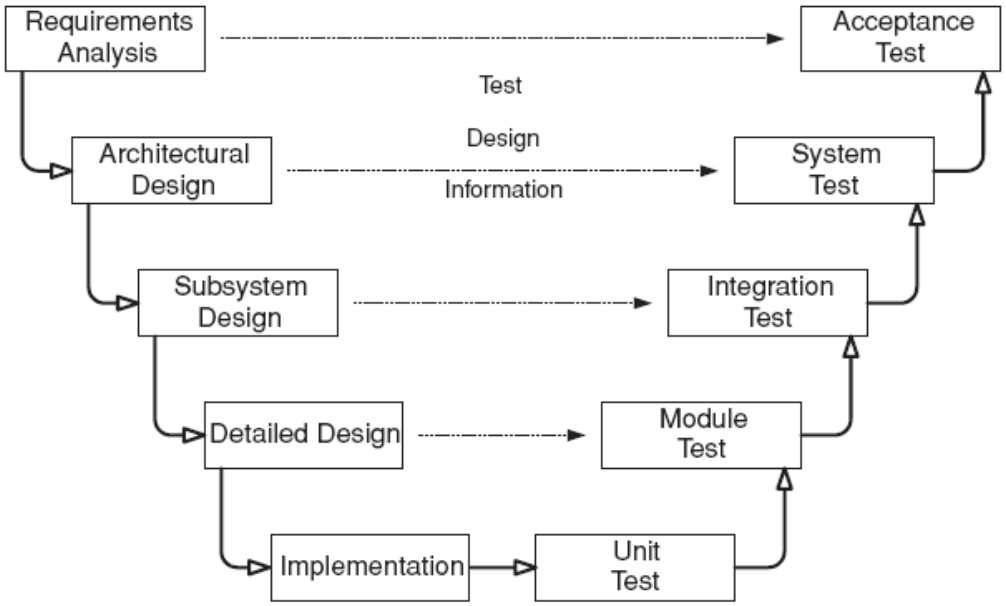

Figure 4-6: Software Development Activities and Testing Levels [9]

4.5.1 Integration Testing

Integration testing which in some testing literature is known as system testing can be conceived from two different viewpoints. From one perspective, integration testing is the task of assessing the interface between working software components. The aim of this assessment is to integrate the subcomponents to form a bigger working component as specified in the design document. In this case, the testing is usually conducted by the software development team. From the other perspective, integration is seen as the process of testing a system whose components have already been assembled or integrated. In this case, the testing activities are normally assigned to an independent team. The common assumption in both types of viewpoint about integration testing is that the individual components testing, which is known as unit testing, has been already exercised.

4.5.2 Unit Testing

Unit or component testing is regarded as most detailed form of testing carried out on

certain software. Depending on the programming language used and type of software, a unit can be a class, a page or applet. With respect to the software development process, unit testing assesses software at the implementation stage. Unit testing normally commences after the code has been developed, reviewed and verified. Good unit testing should uncover implementation defects, including functional description defects and algorithmic defects in addition to control logic defects. In unit testing, white box testing is used to ensure that the tested component complies with the design specifications. In addition, the component is tested using test data to examine the presence or absence of required features.

Although unit testing is normally performed by the software development team, some of the literature on testing has suggested the involvement of independent testers at this stage. Cost-wise, in order to reduce the overall test costs, it is always recommended that the unit testing be completed, making sure that a unit has no defects before submitting units to the integration level of testing.

4.5.3 Framework Testing and Operation Stage

In this study, the testing and operation stage is beyond the focus of this research. Therefore, as in any software project, both unit testing and integration testing will be performed. This includes designing appropriate test cases for each level and type testing planned. Much work has been done in the domain of testing techniques and approaches related to the utilisation of ubiquitous computing. Therefore, implementing one of these proposed techniques can be an option. Furthermore, this stage is considered as possible future work for this research which is worth further investigation in order to select the most appropriate testing strategy for the proposed reengineering framework. The testing strategy needs to cover the three aspects of the framework, which are business logic, ubiquitous computing and software artefacts.

4.6 Summary

In this chapter, an overview of the proposed reengineering framework is provided. The framework aims to address the need for reengineering software systems in order to comply with the utilisation of ubiquitous computing technologies.

The proposed reengineering framework is composed of the following four main stages: program understanding, additional-requirements engineering, integration, and finally testing and operation.

In the first stage of the framework, code reverse engineering techniques are used for program comprehension and architecture recovery. The business logic buried in the source code is extracted to understand the system, and the system architecture is recovered. In the second stage, scenario-based requirements engineering is used and the new business logic is represented using BPMN. In the third stage, the integration is carried out based on a devised integration algorithm which relies on a comparison technique at a diagram level. In the fourth stage, the integrated system is tested using a suitable testing technique prior to its transfer to the operation phase.

The first stage of the framework will be discussed in detail in Chapter 5. The elicitation and modelling parts are the only steps from the requirement engineering stage that will be described in detail in Chapter 6. In a similar manner, in Chapter 7only the integration algorithm will be expanded. The testing and operation stage is not in the focus of this research; hence in the last chapter of this thesis it will be proposed as future work.

Chapter 5

Program Understanding Stage

Objectives

- To highlight the importance of the program understanding stage.
- To define the related terms used in this stage.
- To describe the steps of the program understanding.
- To discuss the steps of the architecture recovery.
- To highlight the focus of this research in each step of the process.

Contents

5.1 Importance of the Stage

The initial step of a software reengineering project is the understanding of the software system which is to be reengineered. Software understanding, or as it is widely called program understanding, is "the task of recapturing the abstract design of a system, in part or in full, from its source code." [134] Very often in software development, the initial documentation of a specific software system suffers from an ageing problem. As the software evolves and many of the software components are updated and modified, the documentation of the software is scarcely being synchronised with these updates. As a consequence of having an obsolete and out-dated documentation, the source code of the software becomes the most reliable and up-to-date documentation. Browsing lines of code in order to understand a software system is certainly an unattainable task with a huge system comprising thousands of lines of code. Hence, much work has been done by researchers in order to devise theories, methods and tools to facilitate software understanding, starting with the source code.

There are different types of approaches aiming to automate the process of program understanding. Such program understanding approaches use various types of source code analysis techniques. These techniques include lexical analysis, syntactic analysis, control flow analysis and data flow analysis. Although there are available many tools of automating the process of program understanding, human intervention is still necessary. Program understanding certainly requires from a software engineer the knowledge not only of programming language, but also of the application domain.

In order to support human program comprehension there is a need for generating different views for the source code. The popular views are in the forms of integrated browsing framework, compact system architecture, or other notations that can be readily processed by external information systems.

Program visualisation is the application of graphical transformations to an executing program in order to enhance the reader's understanding of that program [131]. Reverse

engineering tools usually encompass visualisation capabilities. Software visualisation techniques and tools have become available to support activities such as analysis, modelling, testing, debugging, and maintenance. Visually representing software components and architecture is an effective means of communication between the software stakeholders: end-user, developers, systems engineers, project managers, etc. Today different sorts of graphical notations covering a wide range of abstraction and detail are available. These tools attempt to fulfil the need for different types of information required to understand fully a software system.

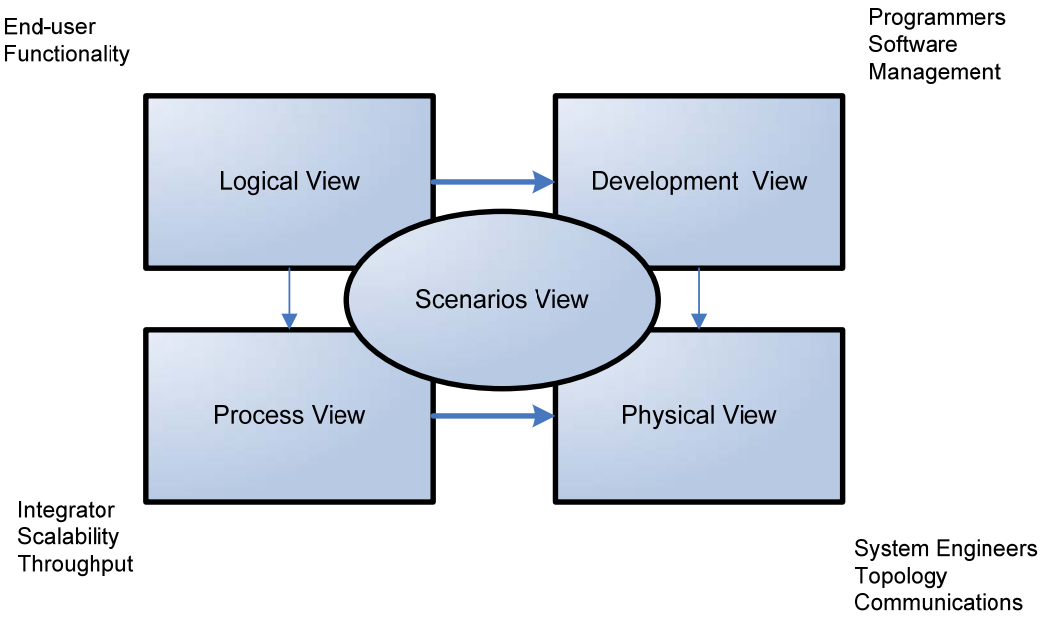

Figure 5-1: The "4+1" View Model [88]

No single view exists by which one can fully understand all aspects of a software system. Instead there are several views which together provide a big picture covering various perspectives of a system. The "4+1 Views" model introduced in [88], for example, describes a system architecture using multiple views or perspectives. The model is composed of five main views as shown in Figure 5-1:

- **Logical view:** the system is decomposed into a set of key abstractions in the form of objects or object classes.

- **Process view:** captures the concurrency and synchronisation aspects of the design.

- **Physical view:** describes the mapping and distribution of the software elements onto the hardware.

- **Development view:** describes the actual software module organisation on the software development.

- **Scenarios:** instances of one or more use cases used to integrate, illustrate and validate the above four views.

In this program understanding stage of the proposed reengineering framework, the following have been targeted:

- First: understanding the functional aspect of the existing software system by means of extracting the business logic embedded in the source code. The extraction of business logic is carried out using reverse engineering tools, with intervention by the software engineers at the refinement stage.

- Second: referring to the "4+1 Views" model, the logical view is also recovered from the source code using available reverse engineering tools and techniques.

5.2 Architecture Recovery

There is no single universally agreed definition of software architecture. Kruchten [88], for example, states that software architecture "deals with abstraction, with decomposition and composition, with style and aesthetics." Perry and Wolfe [124] define software architecture thus "architecture is concerned with the selection of architectural elements, their interactions, and the constraints on those elements and their interactions necessary to provide a framework in which to satisfy the requirements and serve as a basis for the design." Using analogy to building architecture, they proposed the following formula of software architecture: Software Architecture = {Elements,

Form, Rationale}. Nevertheless, common to all proposed definitions in the literature of software architecture is the notion that architecture deals with large scale constructional techniques and ideas, constraints, a high level organisation of a system, architectural, styles rationale, interconnections, collaborations and of system artefacts. Software architecture allows software engineers to overcome the complexity of a system by offering the ability to organise, understand, present, and manage a software system through the different view of the system. This is certainly accomplished in a much more natural and easier manner than by merely using programming language features.

As noted in the definitions above, software architecture is represented through multiple different views or perspectives, such as the famous 4+1 views of Kruchten. The views are shown in Figure 5-1 and described in Section 5.1. Similarly, Booch *et al.* [22] used the same five different views introduced by Kruchten, but with slightly different naming to classify the use of UML diagrams. Each view illustrated is a projection into the organisation and structure of the software system, and focused on a particular aspect of that system. The views of Booch which match those of Kruchten respectively are: design view, process view, implementation view, deployment view and use case view.

In this research, only the logical view of the "4+1" model which is the design view in Booch's model, will be considered. This is because the logical view, unlike the other views, is related to the core functions (business logic) of the system [88].

5.2.1 Architecture Recovery

The goal of architecture recovery is to create a high level of abstraction of a software system. This abstraction is based on the ability to perceive individual procedures, modules and components. A recovered architecture can be utilised to update architecture documentation, support maintenance activities, provide different views of architecture and migrate to other platforms. System architecture serves as the vehicle for communication among the system stakeholders. It acts as the manifestation of the earliest design decisions and as a reusable abstraction that can be transferred to new

systems. For presenting such high level abstraction which is not only for developers, but also for managers and other system stakeholders, an appropriate description language is required to represent the recovered architecture.

In this research only the logical (design) view is considered. Therefore, UML will be used to represent the logical view of the system structure. Available reverse engineering tools will be used for the visualisation of the static structure of the system and the interconnections between its components.

5.2.2 Architecture Visualisation

Much work has been done by researchers to automate the recovery of software architecture and to present this information visually. This effort was undertaken by researchers because one of the main uses of the software architecture is to help in understanding a system that is to be reengineered. Tools that have been designed for the purposes of architecture recovery have mostly been used only to allow the user to visualise the static structure of a system and how the components are interconnected. The most valuable attribute of these tools is that they clearly present important aspects of the architecture, at the same time not showing decisions that have been made in the lower levels of the design.

Rigi [111], as an example, is one of the well-known interactive visual tools used to discover abstractions in large software systems. Rigi allows software engineers to analyse a software system at different levels of abstraction. It also allows the analysis of the dependencies between different subsystems, and the propagation of dependencies through function calls, variable references, etc. Doxygen [72], a documentation system, creates documentation from documented source files and is able to provide a visualisation the relationships between the elements by employing dependency graphs, collaboration diagrams and inheritance diagrams, all of which are created automatically. Additionally, because the documentation is extracted from the sources directly, it therefore proves more consistent with the source code.

In this research, the proposed framework for reengineering is aimed to be general and

not limited to a specific programming language or software development approach. However, in this thesis the focus is on Object Oriented software. Therefore, only a static analysis is carried out on the source code of the existing system, aiming to extract information about the overall structure of classes and class hierarchies along with their methods and attributes represented in UML. In addition, the relationships between the classes in class dependency diagrams, which can be generated automatically using available tools such as Doxygen [72], are also recovered at this stage.

5.2.3 UML and Architecture

UML is considered to be the standard graphical language used to represent systems' architectures in diagrammatic form. UML encompasses a number of advantages as a graphical modelling notation. An advantage of UML is its ability to extend to new notations and concepts beyond its core set. Furthermore, UML makes it possible for concepts, constraints and notations to be specialised for specific domains. UML has the ability to cope with recurring complexity in architecture complexity by employing component technology, patterns and frameworks and visual programming [112]. In addition, UML is independent from any particular programming language or any particular software development process. UML also has been adopted by the software development community, and its theoretical aspects are the subject of several research studies. For these reasons, UML was chosen in this step as the visual representation that is produced as the output of the reverse engineering of the software static structure. Nevertheless, the information reverse engineered from the code can be represented in other forms either graphical or non graphical - with UML being replaced by some other Architecture Description Language (ADL).

In this step of the program understanding stage the class diagram extracted from the source code to present the static structure of the system. Class Diagram views the static structure of the system. The reverse engineered class diagram assists in understanding the system's organisation in general. Also, it helps in identifying the kind of interclass connections that exist in the system.

5.2.4 Class Diagram Recovery

A class diagram depicts the static structure of the core classes that are used to build an Object Oriented system. The attributes and methods of each class together with the optional indication of some of their properties such as visibility and type are provided in the class diagram. Furthermore, the relationships amongst the classes are also presented in the class diagram.

Relationships between classes in the class diagram indicate either the possibility of accessing features of another class or the presence of abstraction mechanisms. Aggregation, association and dependency relationships are shown in a class diagram to indicate that a class has access to attributes or operations of other classes. An aggregation relationship shows that a class is related to another class if the latter is a part of the former. In the association relationship, two classes are connected by a bidirectional association if it is possible to navigate from an object instantiating the first class to an object instantiating the second class. A dependency relationship holds between two classes if any change in one class might affect the dependent class. On the other hand, generalisation and realisation relationships are examples of abstraction mechanisms that can be shown in a class diagram. A generalisation relationship connects two classes when one class inherits features from the other class. A realisation relationship links a class to an interface in the case of the class implementing all methods declared in the interface.

A basic algorithm for class diagram recovery [160] can be achieved by a syntactic analysis of the source code, provided that an accurate definition of the interclass relationships is known. For instance, an association can be inferred when a class attribute stores a reference to another class. From the implementation perspective, there is no clear difference between aggregation and association. Both relationships are usually implemented as a class attribute referencing other objects. Similarly, implementations of the composition and aggregation relationships have a large overlap. Dependency relationships, on the other hand, can be distinguished from the association and aggregation relationships. This distinguishing is possible because the reference to

the accessed object is not stable by being stored in temporary variable and at the same time any change in the target class will possibly affect the user class. Generalisation and realisation relationships can be determined by looking for the keywords ‘extends’ and ‘implements’ respectively in the class declaration. This is Dependent on the programming language.

Available tools for Object Oriented design make it possible to recover class diagrams from the code, which has this kind of syntactic information. One of the tools, Omondo EclipseUML, discussed later in Chapter 8, Section 8.3.2.2 was used to extract the class diagram from the source code of the case study. As pointed out before, the class diagram gives a static view of the structure of the system which provides an informative summary of many design decisions about the system’s organisation. Other tools such as Doxygen, also discussed later in Chapter 8, Section 8.3.2.4 was used to extract the class dependency diagram from the source code of the case study. A class dependency diagram that can be extracted using the Doxygen tool is shown in Figure 5-2.

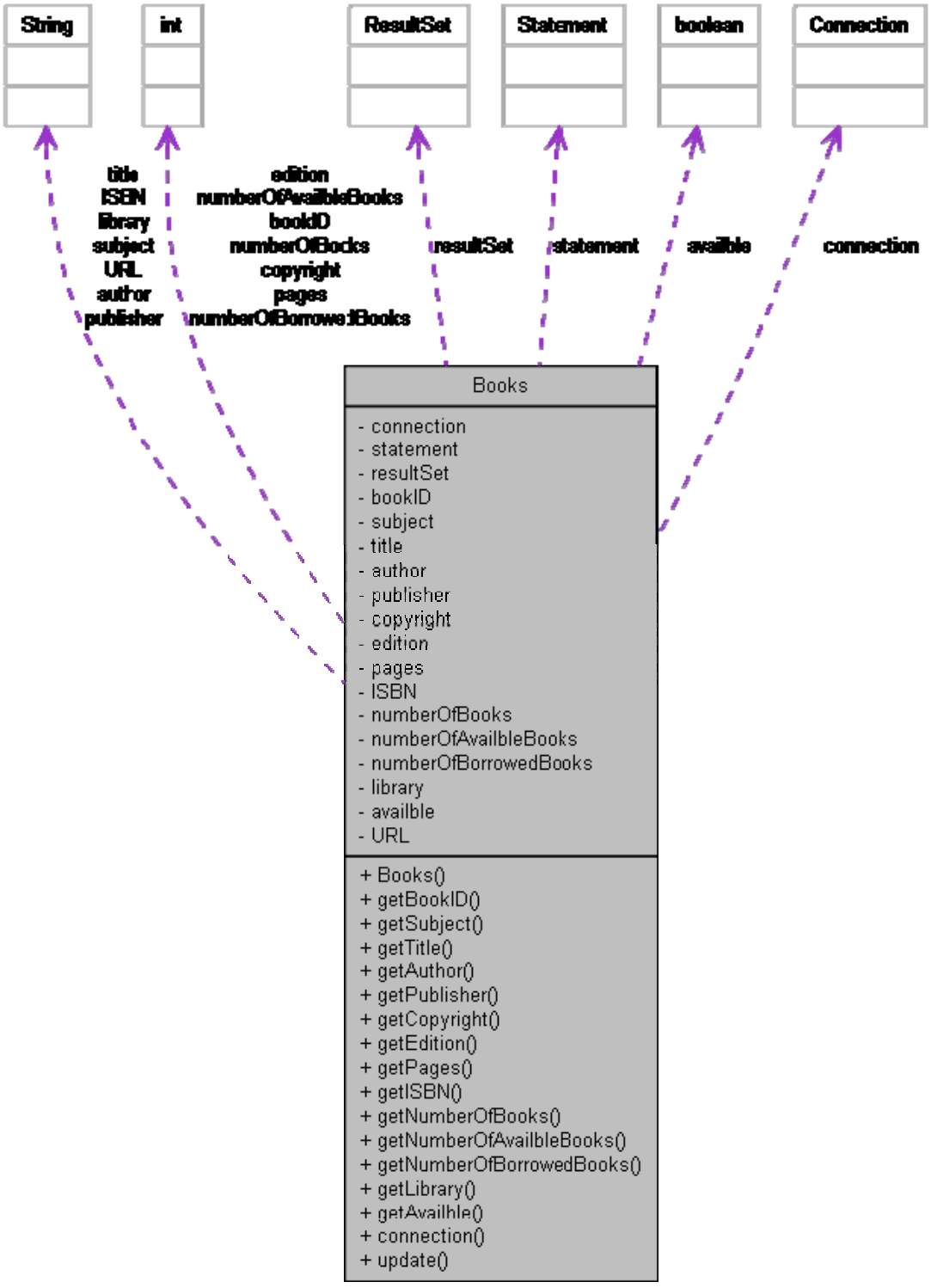

Figure 5-2: Example of Class Dependency Diagram Recovered by Doxygen

5.3 Business Logic Extraction

In this section, related terms are defined in Section 5.3.1. An overview about the business logic extraction methods are provided in Section 5.3.2. The details of the developed business logic extraction method are given in Section 5.3.3. The code grouping technique is discussed in Section 5.3.4. Code parsing, code refinement, activity diagram generation and BPMN diagram generation steps are described in the sub sections of Section 5.3.5.

5.3.1 Introduction

Prior to explaining the details of the two steps of the program understanding and architecture recovery stage, it is worth defining the term 'business' used so often in this thesis. The term 'business' is used extensively with extremely different meanings. The following definitions from dictionaries provide a basic understanding of the meaning of the term in English.

***Business:* 1** *a person's regular occupation or trade.* **2** *work to be done or matters to be attended to.* **3** *a person's concern.* **4** *commercial activity.* **5** *a commercial organization.[13]*

Business:* 1** *the occupation, work, or trade in which a person is engaged.* **2** *a specific occupation or pursuit.3** *serious work or endeavour.* **4** c*ommercial, industrial, or professional dealings.[52]*

In this research the term 'business' which is frequently mentioned in this thesis means "work *to be done or matters to be attended to"* as stated in the number 2 meaning from the Dictionary.com. The number 3 meaning from the FreeDictionary.com "*serious work or endeavour"* also refers to a similar meaning of the term that has been adopted in this thesis. In other words, the term 'business' as used in this research does not strictly refer to commercial or trade activities. Therefore, the types of software discussed in this research are not limited to commercial applications, but also include other applications which do not have a commercial side. In short, the focus is on *"the*

work to be done" by an application, and not on the domain of the application

Mentioning the term business leads to a brief discussion about other terms related to the term 'business' which are normally discussed in the software engineering literature. These terms are 'business process', 'business rule' and 'business logic' which have been explained previously in detail in Chapter 2, Section 2.2. In the following sections, a brief discussion about the three terms has been introduced, along with the adopted definition of business logic.

5.3.1.1 Business Process

A business process "consists of a set of activities that are performed in coordination in an organisational and technical environment. These activities jointly realize a business goal. Each business process is enacted by a single organization, but it may interact with business processes performed by other organizations." [71] The activities in a business process are commonly called tasks; a task can be regarded as a process which cannot be decomposed further. As an example of a process, we shall examine how a public library deals with borrowing a book. We can identify the following tasks:

1. Checking that the potential borrower is a member of the library
2. Checking that the member does not already have the maximum permitted number of books on loan.
3. Recording that this library member has this copy of the book on loan.
4. Rejecting, if task 1or 2 has a negative result.

5.3.1.2 Business Rule

A business rule as defined in [108] "is a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behaviour of the business." A rule is typically composed of two parts: a condition and an action. When the condition is satisfied, the action is initiated. The following is an example of a business rule expressed in Events, Conditions and Actions (ECA) notation:

Business rule: **“INVOICE _REGISTRATION”:**

ON	**(invoice)**
IF	**(related order exists) AND (receipt exists)**
THEN	**begin invoice registration**
	raise event “INVOICE_ACCEPTED”
ELSE	**reject invoice**
	raise event “INVOICE_REJECTED”

5.3.1.3 Business Logic

The term ‘business logic’ has been used to refer to different concepts. Sneed to some extent refers to business logic as the business rules which govern how the manipulation of data in a software system is carried out. He adopted the definition of business rules thus “a requirement on the conditions or manipulation of data expressed in terms of the business enterprise or application domain.” [143]. Zuo in [184, 185] had a different set of definitions for the terms related to business logic. First, she used the same definition of a business rule used by Sneed to define business logic. Second, she defined a business policy thus: “a business policy specifies the rules and conditions on when and where the business logic should be executed”. Third, she defined a business process as “communication of the knowledge of business policies and business logics.” Fourth, she defined a workflow thus: “A workflow consists of a sequence of tasks that implement business logics (rules), control or data flows that link through tasks, participants, and resources required by tasks.” Zou also defined a business process in [185] as “a set of interrelated tasks linked through a number of decision activities. Business processes have starting points and ending points, and they are repeatable.” She also stated that “Typically, a workflow represents a business process.” Similarly Hung and Foo share the same definitions for the former terms in their efforts for extracting business logic from the source code [57, 78].

5.3.1.4 Definition of Business Logic in this Research

In this research, when the term 'business logic' is mentioned it refers to "a set of interrelated tasks linked through a number of decision activities" with a starting and ending point as defined by Zou [185]. Moreover, business logic is concerned merely with the functional requirements of a software system. Functional requirements describe what the system should do from the user's perspective. For example, in a university library system, the possible function requirements can include the following:

- Borrowing a book from the library.
- Recording patron information.
- Searching the library catalogue.

This definition of the term business logic will be the foundation on which the extraction process discussed in the following sections will be constructed.

5.3.2 Business Logic Extraction Methods

Initially, the link between business logic as defined in Section 5.3.1.4 and the underlying implemented software applications is established through the requirement specification and the design documentation. Normally, as time goes on, business logic is constantly improved or changed in order to provide enhanced services or to reduce cost. As a result, the software applications employed are continuously updated to comply with the changes in their functional features. However, the update of the initial documentation of the system often does not keep pace with the gradual update of the software. Consequently, the available documentation becomes obsolete or outdated.

Outdated documentation leaves no option for software engineers other than to rely on the source code in order to understand the already implemented business logic. Of course, it is a challenging task for software engineers to spot manually the code portions that implement certain business logic. The advantage of identifying the segments of the source code that implements business logic is twofold. First, it facilitates comprehending the existing business logic. Second, it facilitates the implementation of the new business logic through determining the locations where the changes must be

made. Since extracting business logic cannot be achieved manually, - especially with large systems - automatic and semi-automatic methods have been proposed. These extraction methods can be generally classified into static-analysis based methods and dynamic-analysis based methods.

5.3.2.1 Static-Analysis Based Methods

The static-analysis based methods consider looking only at the source code without actually executing the software. The methods discussed in Chapter 3, Section 3.3.1 statically analyse the source code in order to extract the implemented business logic. For instance, Zou *et al.* [185] utilised static tracing techniques and a number of heuristics to map source code entities to business logic entities. Similarly, the methods in [78, 79, 184] used only source code static analysis techniques to extract user level functions and business logic from certain types of software systems.

5.3.2.2 Dynamic-Analysis Based Methods

Dynamic-analysis based methods, on the other hand, are done by running software systems on a real or virtual processor. The execution of software systems is carried out upon providing various input values. The extraction of business logic using dynamic-based analysis usually follows one of these two approaches. First, the dynamic analysis is used to verify business logic extracted through a static-based analysis. Second, the execution logs are analysed to extract the business logic. The methods discussed in Chapter 3, Section 3.3.2 demonstrate examples of these two approaches. Foo *et al*. in [57], as an example, utilised a dynamic analysis method to study a system's behaviour at run-time and used the result of the dynamic analysis for verifying the business logic recovered from static analysis.

5.3.2.3 Analysis Method Used in This Research

In this research, a static-analysis based method has been devised. Static based analysis provides a complete record of the possible execution paths of a system. Moreover, static-based analysis does not require an execution of the system. Running a

piece of software requires more professionals to install and configure the software in addition to acquiring the computational resources needed to run the system. Furthermore, for dynamic-based analysis to be effective, the target system must be executed with sufficient input values and usage scenarios to produce interesting behaviour. For the reasons expressed earlier, in this research a static-based analysis approach has been utilised in order to extract the implemented business logic in an existing system.

In this research, the term ‘existing system’ is used instead of the term ‘legacy system’. Although the term ‘legacy system’ can refer to a system that significantly opposes change and evolution to meet new and continuously updating business requirements [25], the term so often used to be associated with old and large systems which were normally coded in an early version of a third-generation language such as COBOL or FORTRAN. Therefore, it is preferred to use the term ‘existing system’ in this research.

5.3.3 The Proposed Business Logic Extraction Method

The method developed for extracting business logic in an existing system follows a static-analysis based approach. The method relies on analysing the source code of the system independently from any input which requires an execution of the system. The devised method is composed of three steps as depicted in Figure 5-3.

- The first step of the method is grouping the code classes into three groups based on the three major types of classes defined by Rumbaugh, Jacobson and Booch [81] in their Analysis Model of an object-oriented design: boundary (interface) classes, control classes, and entity classes. This grouping of the source code entities attempts to facilitate the extraction of business logic from the source code of the target system. The grouping in this method is assumed to be accomplished using the basic program slicing techniques. The details of how the grouping can be done using code slicing is out of the scope of this research.

- The second step of the method is extracting the business logic in each class

which belongs to the control classes group and representing the extracted business logic at the end of the process in a BPMN. The second step is decomposed further into four more internal steps which are code parsing, code refinement, UML activity diagram generation and finally BPMN diagram generation.

- The third step of the method is combining the extracted business logic which is represented in BPMN into more meaningful BPMN representation. The objective of this step is to make the extracted business logic more readable and understandable by the non-technical professionals involved in the reengineering process.

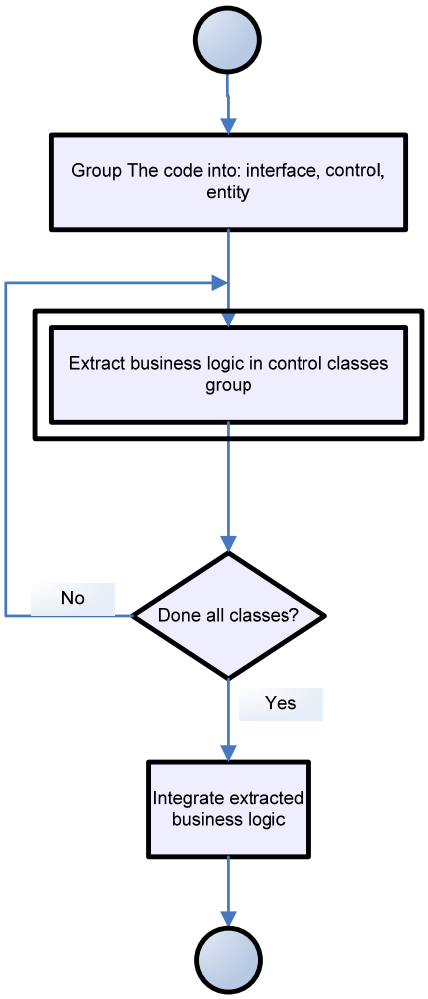

Figure 5-3: Business Logic Extraction Steps

5.3.4 Code Grouping

Grouping as a general term means grouping entities or objects according to their relationships or similarities. In other words, clustering is the classification of similar objects into different groups. Grouping algorithms "partition data objects (patterns, entities, instances, observances, units) into a certain number of clusters (groups, subsets, or categories)" [179]. The concept of grouping has been applied in many engineering disciplines, including mechanical engineering and manufacturing. Although software engineering is relatively new compared with other well-established engineering disciplines, clustering has also been intensively studied by researchers in software engineering.

In software engineering, grouping techniques have been applied in many areas. For instance, it has been used to capture reusable legacy code segments in a legacy system. It is also applied in the program understanding area through decomposing large software systems into more manageable components. Not only is clustering limited to software maintenance and evolution, but it is also used in the stage of designing a software system.

In software clustering literatures, several clustering algorithms focus on the utilisations of the software structure. The algorithms use structural dependences and relationships to decompose large software systems into a set of meaningful modular clusters. A meaningful cluster normally refers to a cluster with maximum cohesion within a module and minimum coupling between modules. Nevertheless, approaches that used attributes other than the software structure have also demonstrated merit. Such approaches include grouping of software entities based on file names, ownership or functionality. For example, in [101] a clustering technique has been applied to support software architecture decomposition based on attributes described in the requirements document. Another example of clustering techniques which use attributes other than the software structure is presented in [140]. The approach aimed at identifying decompositions of a software system based on the participation of software entities in accomplishing the system's functionality. The approach utilised a data mining technique

known as ‘transaction clustering’ to extract and cluster software transactions which represent a unit of functionality.

5.3.4.1 Code Grouping in this Research

Jacobson *et al.* [81] defined three major types of classes in their Analysis Model of an object-oriented design. The three types as shown in Figure 5-4 :

- Boundary classes which are called Interface classes,
- Control classes.
- Entity classes.

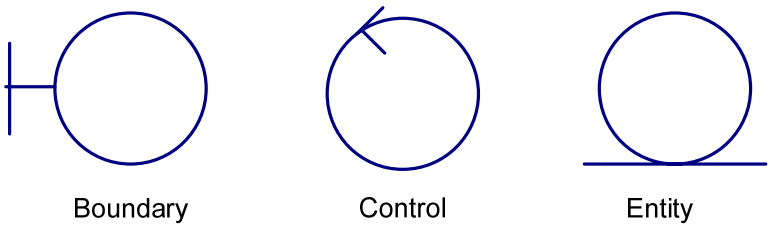

Figure 5-4: Three Types of Classes [81]

The following are the definitions of the three types as stated by Jacobson *et al.* [81]:

- **Boundary (Interface) Classes:** *Are used to model interactions between the system and its actors (i.e., users and external systems). The Interaction often involves receiving (and presenting) information and requests from (and to) users and external systems.... Boundary classes often represent abstractions of windows, forms, panes, communication interfaces, printer interfaces, sensors, terminals, and other APIs. Each boundary class should be related to one actor.*
- **Entity Classes:** *Are used to model information that is long-lived and often persistent. Entity classes model information and associated behaviour of some*

phenomenon or concept such as an individual, a real-life object, or a real-life event.

- **Control Classes:** *Are used to coordinate sequence and generally control the interaction between objects of other classes. Control classes typically encapsulate the business logic or processing flow of a single use case.*

The aim of this grouping is to assist in extracting the business logic from the source code which will be discussed in detail in the next section; Section 5.3.5. In the process of extracting the business logic, the focus will be on the source code classes which are part of the control classes group. The control classes group is more likely to hold the classes that are mostly related to the core business functions of the application. In this research, it is assumed that the grouping of the code classes can be attained by taking advantage of the basic program slicing techniques. Code slicing has been discussed in Chapter 3, Section 3.1.4.

An example of a control classes' group in a library management system which performs the basic books loan functions can be something similar to what is shown in Figure 5-5.

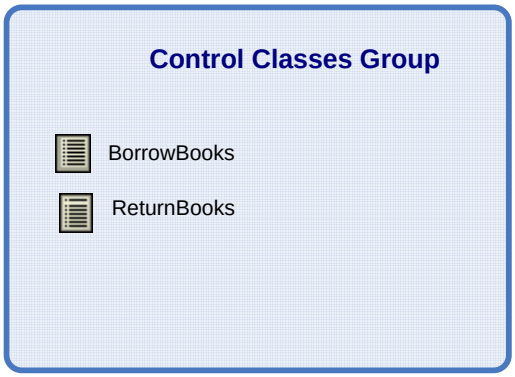

Figure 5-5: An Example of Control Classes Group

5.3.5 Extracting Business Logic in a Code Group

In order to extract the business logic in a code group, in this research a source code

static-analysis based approach has been adopted. A static-analysis based approach is independent of the input and does not require the setup and execution of a software system, as explained in Section 5.3.2.1. The overall business logic extraction process steps are illustrated in Figure 5-6. As depicted in Figure 5-6 these four steps are executed on each source code of classes which belongs to the control classes group.

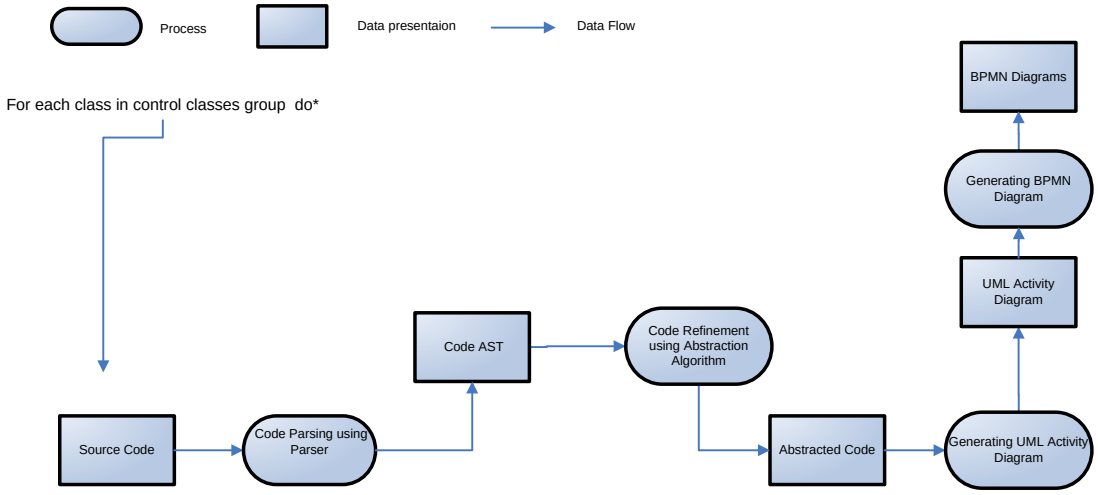

Figure 5-6: Extracting Business Logic in a Code Group

Although the focus will be on the control classes group, as stated in the previous section, Section 5.3.5, other code groups may also be considered, depending on the type of the software system. The determination of including the source entities from a group amongst the other groups is granted to a software engineer. The software engineer handling this task is assumed to have experience not only in the programming language, but also in the software system application domain. The four steps performed in the process of extracting business logic in a code group are: code parsing, code refinement, UML activity diagram generation, and finally the BPMN diagram creation.

1. **Code Parsing**: Code parsing is the first step of the business logic extraction process performed on the source code of each code entity in a code group. Depending on the programming language used, a suitable parser will be employed. For example, for an application coded in Java, a Java parser will be used. The role of the parser is to parse the input code in order to generate an

abstract syntax tree (AST). The generated AST can be traversed by other tools to analyse the source code as a tree of nodes, where each node represents a part of the source code. The purpose of traversing the generated AST is to facilitate the execution of the preceding step of the business logic extraction process which is the code refinement.

2. **Code Refinement:** The second step of the extraction of business logic process is the code refinement. In the code refinement operation, we aim to raise the abstraction level of the source code by filtering out programming specific features. The generated AST of a source code will be traversed to analyse the source code as a tree of nodes, where each node represents a part of the source code. The aim of traversing the AST is filtering out the non-potential candidates for business logic entities based on the characteristics of the code and the programming language used. The elimination of non-business related code such as supporting and error handling code is carried out based on a developed abstraction algorithm.

3. **UML Activity Diagram Generation:** Subsequent to the code refinement step, UML activity diagrams will be automatically generated from the abstracted source code entities. There exist a number of techniques to extract a UML activity diagram automatically from the source code. Likewise, there are now several automatic code flow chart and activity diagram generator software tools that can reverse engineer a program with a code analyser and create a programming flowchart from the code. One of these available tools can be used to create activity diagrams from the source code of the abstracted classes. Since the non-business logic code has been filtered out, the activity diagram will view an abstracted level of the business logic embedded in the source code.

4. **BPMN Diagram Creation:** The proposed approach aims to present the extracted business logic in a way that non-technical professionals are familiar with. In the fourth and last step of the extraction of business logic process, software engineers with adequate knowledge in the domain of the application and the programming

language used are needed. Since the BPMN is currently widely used by business professionals, it has been adopted in this research. The software engineer's job in this step is to convert the generated activity diagrams into BPMN notation and replace the code inside the diagrams with more human readable expressions. Although the available tools claim to be capable of automatically transforming UML activity diagrams into a more business-oriented fashion notation, human intervention is still required to complete the final transformation.

5.3.5.1 Code Parsing

The first step in the process of extracting business logic buried in the source code is parsing the source code and generating an Abstract Syntax Tree (AST) of the source code. Parsing is the process of analysing a sequence of tokens in order to determine its grammatical structure with respect to a formal grammar. Parsing, or syntactic analysis as it is sometimes called, as a concept is not only limited to computer programming languages, but can also be applied to natural languages [156]. In computing, the component that handles this analysis is called a parser. A parser is a component in a programming language interpreter or compiler. The parser examines correct syntax and builds some form of internal representation of the source code [4].

In order to create the data structure found within the language token, token generation or lexical analysis must take place, here the input character stream is divided into symbols, and these symbols are defined by regular expressions. After this step the tokens are checked to see if they form a permissible expression, this checking is performed by referring to a grammar that is free of context. This context-free grammar repeatedly defines the components that can be used to create an expression and also their required order of appearance. The parsing process creates different data structure in forms such as abstract syntax, parse tree or any other possible hierarchical structure.

A parse tree or Concrete Syntax Tree (CST) is defined by [4] thus "A parse tree pictorially shows how the start symbol of a grammar derives a string in the language."

AST, on other hand, is a simplified syntactic representation of the source code, and is most often expressed by the data structures of the language used for implementation [116]. AST is thus different from CST, as AST hides the whole syntactic clutter, but only shows the parsed string in a structured and compact way. This compact way of representation for the code structure facilitates convenient analysis and further processing. The example presented in Figure 5-7 makes the explanation of the difference between CST and AST easy to understand.

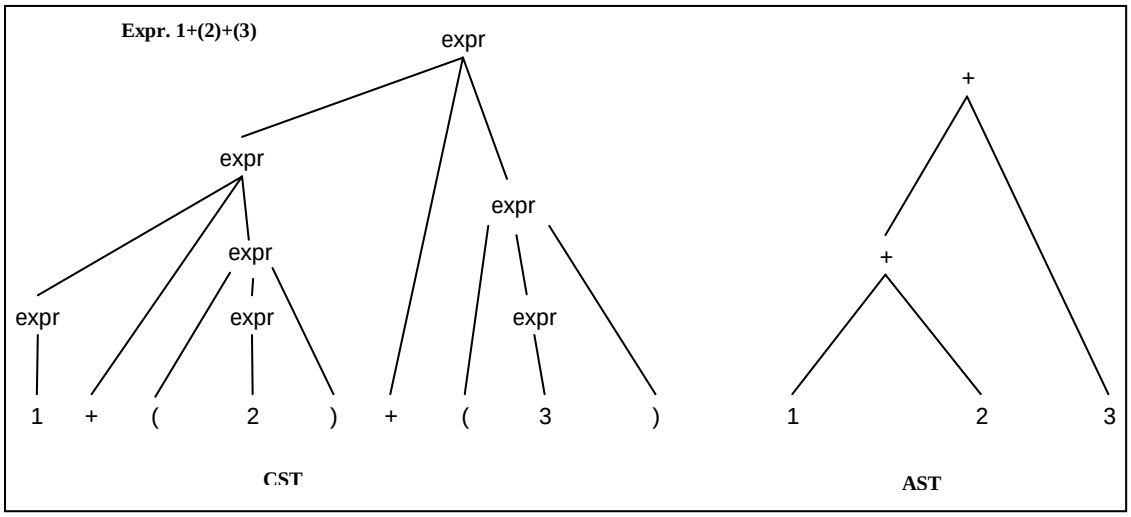

Figure 5-7: CST vs. AST

In this research, the generated AST of the source code will be traversed by other tools to analyse the source code as a tree of nodes. Each node of the generated AST represents a part of the source code. The purpose of the traversing process is to facilitate the execution of the proceeding step of the business logic extraction process which is the code refinement. An example of AST that will be generated for the source code is shown in Figure 5-8

Figure 5-8: AST Example

5.3.5.2 Code Refinement

In the code refinement step, once the AST of the source code has been created by the parser, it can be analysed or manipulated by a visitor. The AST is traversed to analyse the source code as a tree of nodes, where each node represents a part of the source code. A 'visitor' can be implemented in different ways, depending on the programming language used and the purpose of traversing the AST. For example, in Eclipse [154] which is an open source tool integration platform used for a Java programming development environment, every subclass of AST node contains specific information for the Java element it represents. In Chapter 8, the case study, Eclipse tools and

plug-ins will be discussed in more detail in Chapter 8. An example of Eclipse ASTVisitor usage is shown in Listing 5-1.

```
ASTNode node = ...;

PositionSearchVisitor visitor = new
PositionSearchVisitor(position);

node.accept(visitor);

visitor.getNearestNode(); //PositionSearchVisitor's method
```

Listing 5-1: Example of Eclipse ASTVisitor Usage [156]

Another type of visitor transcribes each AST node into an XML representation. This transformation allows the AST to be examined conveniently using standard XML capable tools. An example of AST rendered as XML is shown in Listing 5-2.

```
?xml version="1.0" encoding="UTF-8"?>
<CompilationUnit beginColumn="1" beginLine="12" endColumn="1"
endLine="308">
    <ImportDeclaration beginColumn="1" beginLine="12"
endColumn="21" endLine="12" importOnDemand="true"
importedName="javax.swing" importedNameNode="Name"
packageName="javax.swing" static="false">
        <Name beginColumn="8" beginLine="12" endColumn="18"
endLine="12" image="javax.swing"/>
    </ImportDeclaration>
    <ImportDeclaration beginColumn="1" beginLine="13"
endColumn="18" endLine="13" importOnDemand="true"
```

Listing 5-2: Example of AST Rendered as XML

The purpose of traversing the generated AST is to filter out the non- business-logic entities based on the characteristics of the code and the programming language used. The result of this operation is an abstracted source code which will be used in the next step which is the generation of the UML activity diagram. The elimination of the non-business-logic code is performed using a developed abstraction algorithm. Although the developed algorithm is not limited to a specific programming language, Java will be used in this thesis to describe the algorithm.

```
For Each Class in Group n Filter out
     1: Import Statements
     2: Getters and Setter Methods
     3: GUI components Creation Methods
     4: GUI layout Management methods
         For each candidate method m Filter out
             1: Variables Declaration
             2: Exception Handling Statements
             3: Utility Code Statements
             4: Type Objects Statements
          End For
End For
```

Figure 5-9: Abstraction Algorithm

The abstraction algorithm depicted in Figure 5-9 contains two levels of abstractions. The first level contains the code that is opted out from the class as a whole. The second level contains the code that is opted out for each method within a class. The following are examples and a description of the non-business logic candidates code entities which are filtered out from the source code of a Java class part of a basic library management system.

5.3.5.2.1. Class Code Level

1. **Import Statements:** are used when a program refers to a class the compiler needs to determine which package contains that class. Hence the following code in class will be eliminated. Listing 5-3 shows an example of such code.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
```

Listing 5-3: Import Statements

2. **Getters and Setters**: a getter is a method that gets the value of a specific property. A setter is a method that sets the value of a specific property. These methods are

trivial and have no role in implementing the business logic. Therefore they are eliminated from the class code. Listing 5-4 shows an example of such code.

```
public int getBookName() {
        return Book.Name;
    }
public void setBookName(String anyName) {
Book.Name = anyName;
```

Listing 5-4: Getters and Setters

3. **Graphical User Interface (GUI) Creation and Management methods:** the methods of the two sets of classes, Abstract Windowing Toolkit (AWT) and Swing, for creating managing the layout of a program's user interface are also filtered out. Listing 5-5 shows an example of such code.

```
private JPanel northPanel = new JPanel();
private JPanel informationTextFieldPanel = new JPanel();
private JTextField[] informationTextField = new JTextField[10];
```

Listing 5-5: GUI Creation and Management

5.3.5.2.2. Method Code Level

1. **Variables Declaration**: the purpose of variable declaration is to warn the compiler that the variable exists and to notify the compiler of the type for that variable. Therefore, the variable deceleration block is eliminated from the method source code. Listing 5-6 shows an example of such code.

```
    private AddBooks addBooks;
    private ListAvailbleBooks listAvailble;
    private ListBorrowedBooks listBorrowed;
    private EditBooks editBooks;
```

Listing 5-6: Variables Declaration

2. **Exception Handling Statements:** an exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions. An exception object can be identified in try, catch and finally blocks in throw statements. Cases in which exception used to report error cases in the business logic handling are not handled in this algorithm. Listing 5-7 shows an example of such code.

```
    try {
            listAvailble.setSelected(true);
                    }
    catch (java.beans.PropertyVetoException e) }
```

Listing 5-7: Exception Handling Statements

3. **Utility Code Statements:** act as helpers in providing internal services that facilitate the completion of a business logic code. For instance, a utility object performs transaction initialisation, rollback, commitment, tracing or logging.
4. **Type Objects Statements:** a Java type class, such as Enumeration, String and Vector, provide primitive building blocks to construct a Java program. Hence, code fragments of the former basic types are not regarded to implement business logic.

Listing 5-8 shows a part of a class before applying the abstraction algorithm.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.SimpleDateFormat;
import java.util.Locale;

public class BorrowBooks extends JInternalFrame {
              private JPanel northPanel = new JPanel();
       private JLabel title = new JLabel("BOOK INFORMATION");

       private JPanel centerPanel = new JPanel();
       private JPanel informationPanel = new JPanel();
       private JLabel[] informationLabel = new JLabel[4];
       private String[] informationString = {" Write the Book ID:",
" Write the Member ID:",
                                          " The Current Data:", " The
Return Date:"};
       private JTextField[] informationTextField = new
JTextField[4];
       private String date = new SimpleDateFormat("dd-MM-yy",
Locale.getDefault()).format(new java.util.Date());
       private String[] data;

       private JPanel borrowButtonPanel = new JPanel();
```

Listing 5-8: Example of a Class Source Code before Refinement

Listing 5-9 shows part of the same class after applying the abstraction algorithm which filters-out the non-business-logic-related code entities from the source code of the class.

```
    public class BorrowBooks extends JInternalFrame {

      borrowButton.addActionListener(new ActionListener() {
           public void actionPerformed(ActionEvent ae) {
                    if (isCorrect()) {
                       Thread runner = new Thread() {
                        public void run() {
                        book = new Books();
                        member = new Members();
                        borrow = new Borrow();
book.connection("SELECT * FROM Books WHERE BookID = " + data[0]);
member.connection("SELECT * FROM Members WHERE MemberID = "+data[1]);
int numberOfAvailbleBooks = book.getNumberOfAvailbleBooks();
int numberOfBorrowedBooks = 1 + book.getNumberOfBorrowedBooks();
int numberOfBooks = 1 + member.getNumberOfBooks();
if (numberOfAvailbleBooks == 1)
{                                                          numberOfAv
ailbleBooks -= 1;
book.update("UPDATE Books SET NumberOfAvailbleBooks =" +
numberOfAvailbleBooks +
",NumberOfBorrowedBooks =" + numberOfBorrowedBooks + ",Availble =
false WHERE BookID =" + data[0]);
member.update("UPDATE Members SET NumberOfBooks = " + numberOfBooks
+ " WHERE MemberID = " + data[1]);
borrow.update("INSERT INTO Borrow (BookID, MemberID, DayOfBorrowed,
DayOfReturn) VALUES (" +
data[0] + "," + data[1] + ",'" + data[2] + "','" + data[3] +
"')");
                                                  }
else if (numberOfAvailbleBooks > 1)
{                                                            numberOfA
vailbleBooks -= 1;
book.update("UPDATE Books SET NumberOfAvailbleBooks =" +
numberOfAvailbleBooks +
",NumberOfBorrowedBooks =" + numberOfBorrowedBooks + " WHERE BookID
=" + data[0]);
```

Listing 5-9: Example of a Class Source Code after Refinement

5.3.5.3 UML Activity Diagram Generation

Subsequent to the code refinement step, UML activity diagrams will be generated automatically from the abstracted source code entities. In this research, the UML activity diagram has been selected from the five UML diagrams for modelling the dynamic aspects of systems. This activity diagram has been selected because it is the most suitable diagram among the UML diagrams for describing and modelling a workflow and an operation. A business logic diagram based on a UML activity diagram can exhibit the generation and handling of business events, or can be received between

different processes. Although in this research the generation of UML activity diagram is not the last step of the process of extracting business logic, the activity diagram by itself is readily understood by non-technical people.

The generated activity diagram in this step follows the basic standard notation as described in Chapter 3, Section 3.2. A typical activity diagram can be partitioned into swimlanes that determine where an activity is placed in the swimlanes of the object where the activity occurs. Since the diagrams extracted from the source in this step will be simplified further using BPMN, swainlanes and other activity diagram advanced notation elements are not used.

There exist a number of techniques for extracting UML diagrams automatically, including activity diagrams from source codes. Research in exploiting UML in reverse engineering context resulted in the development of a number of reengineering tools and methods. Some of these research efforts were discussed in Chapter 3, Section 3.2.4.

In the same manner, there are several automatic code flowchart and activity diagram generator tools. These tools can create code flowchart and activity diagrams directly from the source code using a code analyser engine. Examples of such tools are Code Visual to Flow Chart [53] and Flowchart4j [40] which provide flowcharts for Java methods generated from the source code. For example, the Code Visual to Flow chart can automatically generate a programming flow chart from the source code to help programmers to document, visualise and understand the code. The tool works with a large number of programming languages, such as C, C++, Visual Basic, and Java. More information about these tools will be presented in the Case Study chapter, Chapter 8.

In order to generate the activity diagrams from the abstracted code, one of the aforementioned tools and techniques can be utilised. The role of the selected tool will be to create activity diagrams from the abstracted source code entities. Since the non-business logic code has been filtered out in the code refinement step the activity diagram will show the refined code will most likely represent the business logic embedded in the source code. Figure 5-10 depicts part of an activity diagram which can be generated from the refined class shown in Listing 5-9.

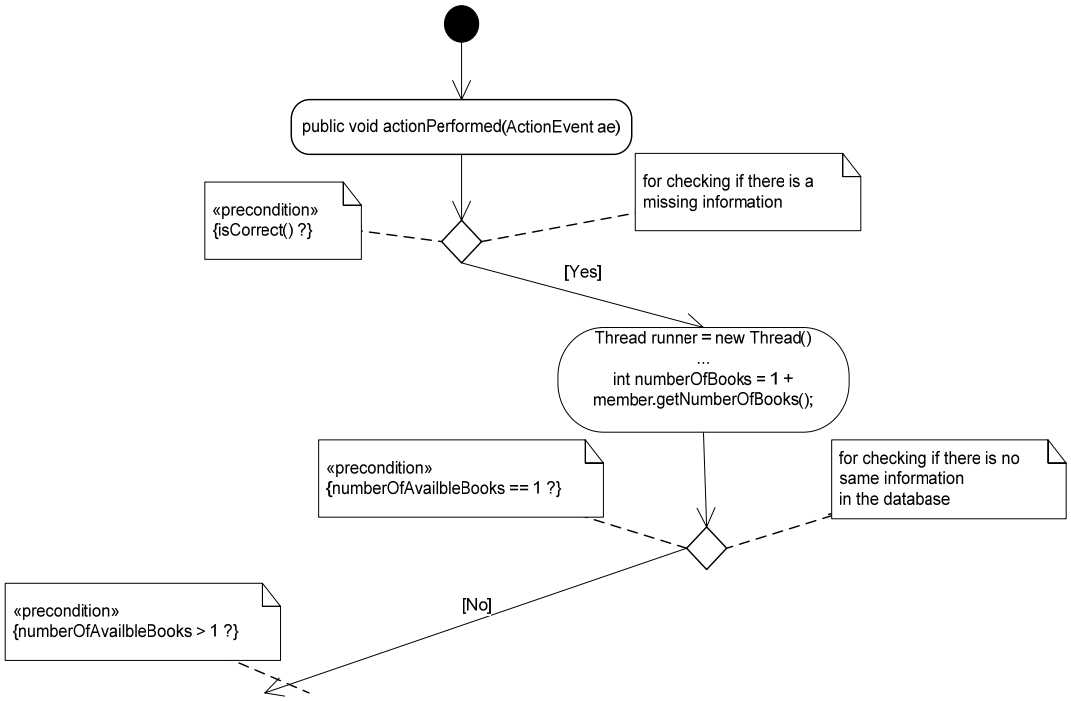

Figure 5-10: Part of a Refined Class Activity Diagram

5.3.5.4 BPMN Diagram Creation

As explained earlier, as it is important to extract the business logic from the source code, it is also important to represent this extracted logic in an understandable manner. Understandable here means for both technical and non technical professionals. In this last step of the process of extracting the business logic, the generated activity diagram is transformed into BPMN.

BPMN as described by OMG [120] “provide(s) a notation that is readily understandable by all business users, from the businesses s that create the initial drafts of the processes, to the technical developers responsible for implementing he technology that will perform those processes, and finally, to the business people who will manage and monitor those processes. Thus, BPMN creates a standardised bridge for the gap between the business process design and process implementation." The Business Process Diagram (BPD) as specified in BPMN was designed to model business processes in a way that makes it easy to use and understand. In spite of the

simplicity of the BPD diagram, it offers expressiveness to model very complex business processes. More about BPD elements has been explained in Chapter 3, Section 3.2.5. Despite some practitioners advocating the use of UML activity diagrams for depicting workflow for business oriented people, activity diagrams are not easily understood by non-technical people, compared with BPD.

BPMN provides a number of advantages for business analysts to modelling business processes over the UML activity diagram. The following are examples of these advantages:

- **First**: BPMN solid mathematical foundation is expressly designed to map to business execution languages.
- **Second**: OMG specification for an activity diagram does not talk clearly about business process modelling. It generally states that activities in activity diagrams may be applied to organisational modelling for business process engineering and workflow.
- **Third**: BPMN takes a process-centric approach to the modelling of applications, while UML offers an object-oriented approach.
- **Fourth**: a BPMN business process diagram can be simulated. A simulated model imitates the operations of the business process, by walking through the events in compressed time while displaying an animated picture of the flow.

To transform the generated UML activity diagram resulting from the previous step, in this research a software engineer with adequate knowledge in the domain of the application and the programming language used will be required, unlike the last three steps of the process, which were carried out without the need for human intervention. Code parsing, code refinement, and UML activity diagram generation are executed automatically using the appropriate software tools. As pointed out in the beginning of the chapter, the approach we propose aims to present the extracted business logic in a

manner that non-technical professionals are familiar with. Therefore, the job of the software engineer in this step is to simplify the extracted activity diagram by replacing the code inside the diagram with more human readable expressions using BPMN.

As presented in Chapter 3, Section 3.2.5, there are some approaches and techniques that can be utilised for automatically or semi-automatically transforming UML activity diagrams to BPMN. For example, IBM demonstrated in [64] how to transform UML activity diagrams automatically into WebSphere Business Modeler processes so that the UML diagrams can be reused in a more business-oriented fashion. The transformation of UML activity diagrams to Modeler processes is performed using the Rational platform with the Model Transformation Framework tool from AlphaWorks. Despite the existence of such techniques and tools for transforming UML activity diagrams to BPMN diagrams, human intervention is still essential to complete the final transformation. Figure 5-11 depicts part of an BPMN diagram which can be generated from the activity diagram shown in Figure 5-10 .

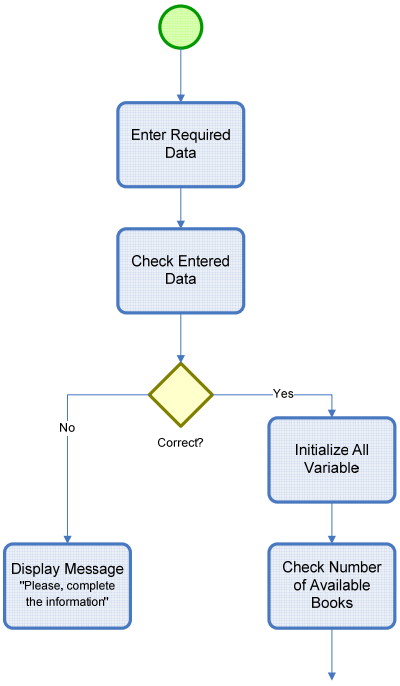

Figure 5-11: Part of a Refined Class BPMN Diagram

5.4 Summary

At the beginning of this chapter, an overview about the importance of the program understanding stage has been discussed. The focus of this research in this stage has been highlighted. This stage of the framework has two objectives. The first objective is to understand the functional aspect of the existing software system by means of extracting the business logic imbedded in the source code. The second objective is to recover the logical view of the "4+1 Views" model using available reverse engineering methods.

After that, an overview of definitions of software architecture was given. The purpose and advantages of the architecture recovery in software reengineering process were described. For the purposes of this research the UML class diagram. The class diagram is used to show the static structure of the system.

Then in the chapter, the terms 'business', 'business rules', 'business process' and 'business logic' were defined. Moreover, the definitions of these terms adopted by this research were identified.

An overview of the business logic extraction methods, which are the static- analysis based method and the dynamic-analysis based method, was provided. After that, the selected method of extracting the business logic along with the rationale behind the selection was presented.

The devised method is composed of three steps: the first step of the method is grouping the code entities or classes into three groups: boundary (interface) classes, control classes, and entity classes. It is assumed that this grouping can be accomplished using program slicing techniques. The second step of the method is extracting the business logic in each class which belongs to the control classes group, and then representing the extracted business logic in a BPMN. The third step of the method is combining the extracted business logic into a more meaningful BPMN diagram.

Chapter 6
Additional-Requirements Engineering Stage

Objectives

- To discuss the impact of ubiquitous computing on business logic.
- To define the scenario-based requirements elicitation.
- To discuss the steps of the additional-requirements engineering stage.
- To highlight the focus of this research in each step of the stage.

Contents

6.1 Introduction

In this introduction two main points are discussed. First, the impact of ubiquitous technologies on business logic is identified. Second, the focus of this research in this stage is specified.

In the first part of the introduction, Section 6.1.1, as pointed out at the beginning of this thesis, the presumed principal cause of the new user functional requirements to the existing system is the introduction of ubiquitous technologies. Therefore, this introduction starts with stating the impacts of defining the business process and the role of IT in the various levels of BPR. Then the impacts of ubiquitous technologies on the business logic are presented.

The second part of the introduction, Section 6.1.2, shows the focus of this research in the stage of requirements engineering, and the method chosen to tackle this task. This research focuses on the requirements engineering tasks using a scenario-based technique.

6.1.1 Impact of Ubiquitous Technologies on Business Logic

The reasons for business process reengineering along with the possible levels of engineering were discussed in Section 6.1.1. Overviews of the role of information technology in BPR and the five levels of information-technology-based transformations were provided in Section 6.1.2. Finally, in Section 6.1.3, the impact of the utilisation of ubiquitous computing technologies on BPR was given.

6.1.1.1 BPR

BPR, as defined by Hammer and Champy [69], refers to “the fundamental rethinking and the radical redesign of business to achieve dramatic improvements in critical contemporary measures of performance, such as cost, quality, service, and speed." There are several reasons that may lead an organisation to reengineer its current business process, depending on the role of an organisation. For instance, in the

commercial domain, the reasons for reengineering include economic shift, social trends, change in the scope of the firm, and, most important, competition. In general and regardless of the type of the organisation, the three reasons commonly suggested for organisations finding it necessary to choose to reengineer are:

- Customer satisfaction,
- to be more efficient or
- to save on costs.

Reengineering as a response to the above reasons can be implemented at different levels within an organisation. As depicted in Figure 6-1, these levels are:

- Process reengineering: to re-conceptualise the overall business processes from beginning to the end, i.e. “door to door”.
- Process simplification: to improve significantly already existing work flows and quality measures.
- Process improvement: to make minor changes at the operational task level.

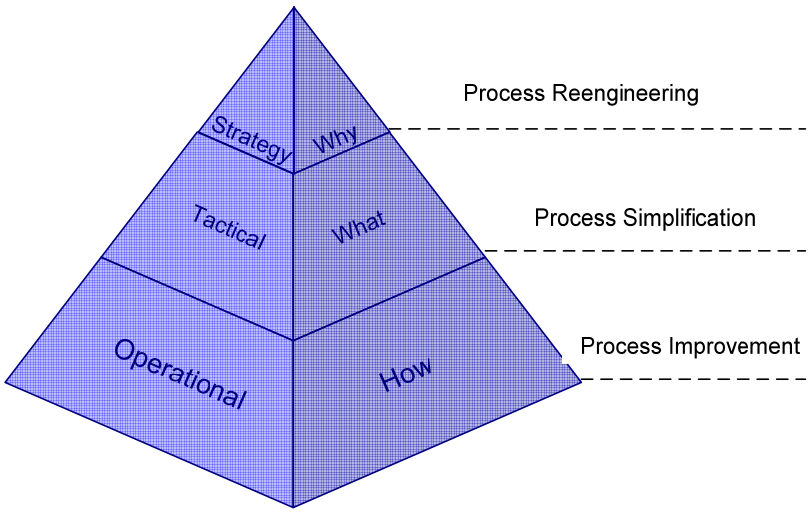

Figure 6-1: Business Process Reengineering Levels [146]

6.1.1.2 IT Role in BPR

In spite of the other factors that need to taken into consideration in any business reengineering project, IT is undoubtedly an essential enabler to the BPR endeavour. With reference to the levels of reengineering illustrated in Figure 6-1, IT can play an important role at all the three levels. Davenport [43] states that IT is fundamental in BPR as it enables organisations

- to break the old rules and conventions that dictated the design of business processes and
- to achieve the aggressive improvement targets of reengineering.

Venkatraman [166] agrees with the argument that considers the role of IT within organisations has leveraged from a main focus on efficiency enhancements to that of fundamental enabler. He also suggests five levels of information-technology-based transformations:

- ***First level****: localised exploitation of technology to existing business processes.*
- ***Second level****: internal integration of IT capabilities across an entire existing business process.*
- ***Third level****: use of IT as a lever for designing an organisation's core business process.*
- ***Fourth level****: exploitation of IT to redesign process extending beyond one organisation to a network of organisations.*
- ***Fifth level****: use of IT to redefine the organisation's business scope.*

6.1.1.3 Impact of Ubiquitous Computing on BPR

Although there have been significant advancement of IT systems, such as e-business systems or Enterprise Resource Planning (ERP), there still remains a number of

problems that these systems are unable to solve. The main reason for the inefficiency in information flow of an enterprise has been human errors, media breaks, and delayed information [53]. Ubiquitous technologies enable many new products and services for different kinds of enterprises that can grant them a competitive advantage and improve their productivity. Ubiquitous technologies such as automatic identification, localisation, and sensor technology are potential candidates to conquer these difficulties [74].

For example, in supply chain management, companies encounter problems related to controlling their inventory. Such problems are normally caused by the lack of coordination between material flow and information flow that leads to what is known in the business as the 'bull-whip' effect. Excess production or stock outs are expected outcomes of such problems which can waste twenty five percent of a company's operation costs [145]. Ubiquitous technologies such as RFID systems can cut cost and lead to enhanced supply chain management. Through real-time information about products in the supply chain, production planning will be enhanced and instability in the supply will be reduced.

Errors in production caused by insufficient monitoring and documentation of the production process can also be very costly if they are not recognised at the right time. Moreover, defective items cannot be identified because individual information on an item basis is missing. The avoidance of media breaks that can be accomplished by using ubiquitous computing technologies improves the efficiency of business processes through automation. A high standard of process automation which results in less human interventions leads to reduced cost by eliminating human errors [23]. In addition, ubiquitous technology can enable objects to become self-aware. Self-awareness means that objects can express their current and past context, and can make decisions that affect the object itself [46].

Another area in which the ubiquitous technologies have an obvious impact on a business process is the domain of Customer Relationship Management (CRM). After the thriving of e-CRM, organisations began to use the Internet to change the ways in which their business reached out to their customers through Web sites which people

could access from PCs at home or at work. The concept of mobile CRM, which uses mobile media such as mobile phones or PDA for managing customer relationships, was presented. Today the trend of CRM systems is to work in a proactive approach to provide the customers with the right service in the right time and in the right location; in some of the literature this is called ubiquitous CRM systems, and in others [6] it is called Proactive CRM systems.

In [148] , a ubiquitous computing process model which identifies tasks and activities that can be supported by ubiquitous technology is proposed. This process model is illustrated in Figure 6-2, and provides a presentation of how the possibilities of the technology can be connect to the business processes. The technology functions are able to help the tasks involved in the business process. To explain further the computing process model employs four functions that change physical objects into smart objects. Depending on their physical context, these functions provide identification, localisation, and physical status information. The four main macro business processes that the model focuses on are customer relationship management (CRM), supply chain management (SCM), and the innovation and supportive processes. These macro business processes can be described in terms of a task or detailed activities.

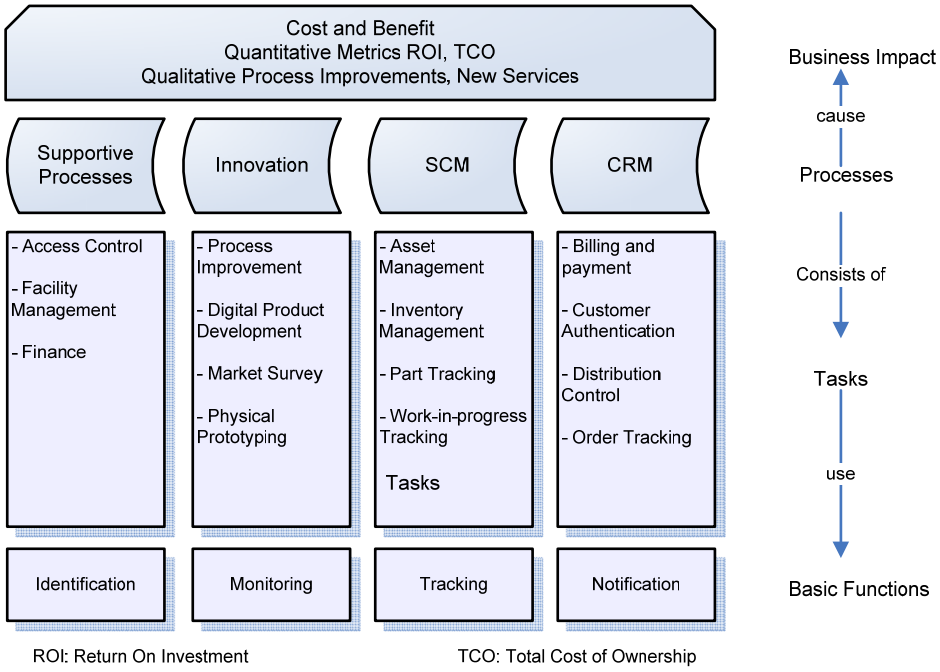

Figure 6-2: Ubiquitous Process Model [148]

6.1.2 Additional-Requirements Engineering Stage

Requirements Engineering (RE) process as described by Sommerville [144] consists of the following distinct steps:

- First: Requirements Elicitation.
- Second: Requirements Analysis
- Third: Requirements Specification
- Fourth: Requirements Validation

In some of the literature, two additional steps also have been added to the RE process: requirements modelling and requirements management. The following sections, Section 6.5.1.1 to Section 6.1.1.4, provide brief descriptions of the four steps of the requirements engineering process. In this stage, the research is concerned with the task of requirements elicitation which resulted from the new feature introduced by the

ubiquitous technologies. For this reason, requirements elicitation techniques are elaborated upon in Section 5.1.1.5.

6.1.2.1 Requirements Elicitation

In the requirement elicitation step, the system requirements are discovered through consulting the stakeholders, from system documentation, domain knowledge and sometimes market studies. The term 'system's stakeholders' refers to any person or group who has either direct or indirect effect on the system requirements. A typical system's stakeholders can include managers, end-users and software engineers. In addition, other parts of an organisation that may be affected by the system - and in some cases external organisations - can be part of the system stakeholders. Requirement elicitation is also called requirement collection, capture or discovery. Since this research focuses on this step of RE only, more details about this activity will be discussed in a subsequent section, Section 6.1.2.5.

6.1.2.2 Requirements Analysis

In the requirements analysis and negotiation step, the requirements are analysed in detail in order for them to be accepted by the system's stakeholders. After gathering the requirements, the analysis step starts by classifying requirements and organising them into related and coherent groups. This activity takes the unstructured collection of gathered requirements and explores each requirement in relationship to other requirements. Moreover, requirements are examined for consistency, omissions and ambiguity. Since usually multiple stakeholders are involved, requirement conflict is anticipated. Thus in this step, conflict is resolved through negotiation.

6.1.2.3 Requirements Specification

The product produced by the system and requirements engineers is the system specification. The system specification describes the function and performance of a system. It also shows the constraints that will control its development. Moreover, the system specification also describes the information that is input to and output from the

system. A system specification can be delivered in different forms. It can be a written document, a graphical model, a formal mathematical model, usage scenarios, a prototype, or any combination of these.

6.1.2.4 Requirements Validation

The role of the validation step is to assess the quality of the products produced as a consequence of requirements engineering. In other words, it is concerned with showing that the collected requirements actually define the system that the stakeholder needs.

A primary requirements validation mechanism is the formal technical review, which includes the system's stakeholders. The review examines the system specification for errors in content or interpretation, missing information, inconsistencies, conflicting requirements, or unrealistic requirements. In order to ensure that a requirements document does define the stakeholders' requirements, it needs to be checked against desirable features such as comprehensibility, verifiability, traceability and adaptability.

6.1.2.5 Requirements Elicitation Methods

When conducting a requirements-engineering process the elicitation of requirements is regarded as the first step in the process. Although this step is sometimes called requirements capturing, the term "elicitation" is preferred to "capture", to avoid the assumption that requirements are out there to be collected simply by asking the right questions. This suggestion is certainly not accurate; the elicitation of requirement can be rather a hard, resource-consuming and time-consuming job.

In general, requirements elicitation activities aim at accomplishing an important goal that identifies the system boundaries. Identification of the system boundaries affects all the following elicitation efforts. For instance, the identification of stakeholders and user classes, of scenarios and use cases, and of goals and tasks all depend on the boundary determination. Basic practices of the elicitation activity to find out what problems need to be solved include assessing the business and technical feasibility, identifying the people who can help in specifying requirements and defining the

technical environment in which the system or product will operate. This is accomplished by exploiting different types of elicitation techniques.

The selection of a specific elicitation technique is dependent on a number of factors, including the time and resources available to the requirements engineer, and the kind of information that needs to be elicited. Examples of classes of elicitation technique are:

- **Interviews**: planning, recording and analysing interviews with the system's stakeholders. Interviews can be structured, with pre-planned questions, unstructured or hybrid, i.e. a mixture of both.
- **Inspection**: studying existing records, statistical sampling. Inspection is valuable where there are large volumes of data.
- **Observation**: also called ethnography, watching, participating and video recording of activities, processes and workflows.
- **Scenarios**: talking through different scenarios with users to understand normal and exceptional processes.

Each elicitation method has its strengths and weaknesses, and is usually best fit for use in particular application domains. Hence, the requirements engineer needs to select the technique or even techniques most appropriate for the elicitation process. In this research, the scenario-based requirements election is adopted. The next section, Section 6.2 gives more information about the method adopted.

6.2 Scenario-Based Requirements Elicitation

A scenario as defined in [157] is "an instantiation of a generic task type or a series of generic tasks linked by transitions. It specifies the characteristics of the group that should carry out, and the social protocols which should be in place. It describes what the users should do at the requirements level, but not how they should do it at any of the lower levels of the framework."

In software engineering, the term ‘scenario’ also has a number of definitions. For instance, in [150] the ‘scenario’ has been defined thus: “Scenarios are examples of interaction sessions, and consist of descriptions of sequential actions which relate to real-life examples rather than abstract descriptions of the functions.” Another definition was introduced by Sutcliffe, who described a scenario as “"facts describing an existing system and its environment including the behaviour of agents and sufficient context information to allow discovery and validation of system requirements.” An agent here means an actor within a specific scenario setting. In a general manner, scenarios have been used in requirements engineering to help elicit requirements, analyse requirements, detect ambiguities, uncover missing features and inconsistencies among specified features, and verify and validate requirements.

6.2.1.1 Scenarios in Requirement Elicitation

In requirements elicitation, scenarios are considered as practical and useful medium. Systems stakeholders can easily relate to the scenarios more than an abstract statement of what they require from a system. In addition, scenarios are very helpful for adding extra information to an outline requirement description. As presented by Potts [125] in scenario-based requirements elicitation, the requirement elaboration is an iterative process, as shown in Figure 6-3, composed of three stages: document requirements, discuss requirements, and evolve requirements.

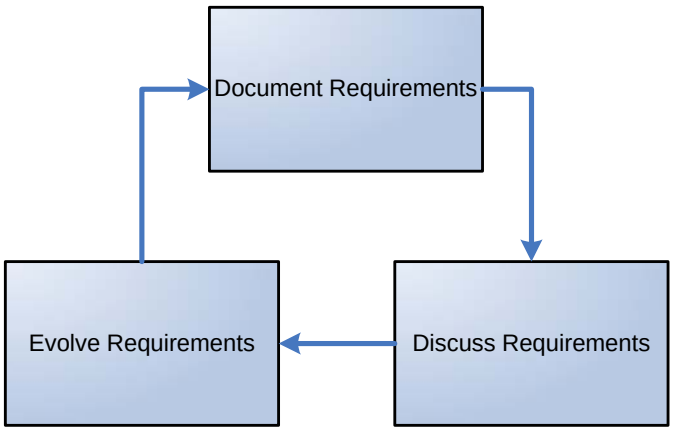

Figure 6-3: Iterative Requirement Elaboration Process [125]

The documenting requirements stage comprises three tasks: first, collecting information from the system's stakeholders, second, studying existing documents and third, drafting requirement documentation. The drafted requirement document usually includes domain knowledge of the system to be developed, the system constraints, background information and existing documentation.

The discussing requirements stage also comprises four tasks: presenting the collected requirements to stakeholders, gathering opinions and suggestions via questions, answers, and reasons, and finally, obtaining the agreement. In this stage, the results of the requirements discussion need to be documented for future reference and further refinement.

The last stage of requirements elaboration process is evolving requirements. In this stage, based on the discussion results, evolving requirements is done by freezing a requirement or changing it or adding more information into the requirements.

Holbrook [77] introduced a scenario-based requirement elicitation called Scenario Based Requirements Elicitation (SBRE). The SBRE creates scenarios in a repetitive process, whereby feedback from the stake holders helps to achieve refinement. Using these created scenarios, the stakeholders provide feedback and the functionality of the system is reviewed. This review of the scenario is essential because it may reveal unstated requirements and furthermore, may improve the entireness of the requirements.

In this research, either of the two methodologies explained above can be used to complete the requirement elicitation step for the user functional requirements in the new additional-requirements engineering stage. The definition of user requirements and functional requirements is explained in Chapter 2, Section 2.2.4.

6.2.1.2 Why Scenarios

In general, a scenario holds information not only about the operation of the current system but also about its surrounding environment. A scenario may contain descriptions of actions in manual systems, descriptions of actors, roles, and also their organisation

settings. In a system that aims to exploit ubiquitous technologies in its operations, it is quite vital that the user be aware of the environment of the system in which it will be employed. This awareness is important, because ubiquitous technologies can be embedded within the environment itself.

More specifically, in the requirement elicitation task, the use of a scenario-based method offers a number of advantages, as follows:

- Scenarios ground argument and reasoning in specific detail or examples. Thus scenarios fit into the process of requirements elicitation which collects examples and stories from stakeholders and then looks for the generalisation.
- Scenarios situate examples with existing memory. Thus scenarios help in understanding requirements problems.
- Scenarios concrete examples, likened with the general abstract model of requirements, help in requirements comprehension.

For all of the above reasons, the scenario-based requirements elicitation approach was adopted in this research.

6.3 Steps of Additional-Requirements Engineering Stage

The proposed four steps in the requirements elicitation process are based on a scenario-based method. The four steps as shown in Figure 6-4 are:

1. Scenarios description in plain text.
2. Scenarios analysis.
3. Scenarios modelling in BPMN.
4. Pseudo code developing for new functions.

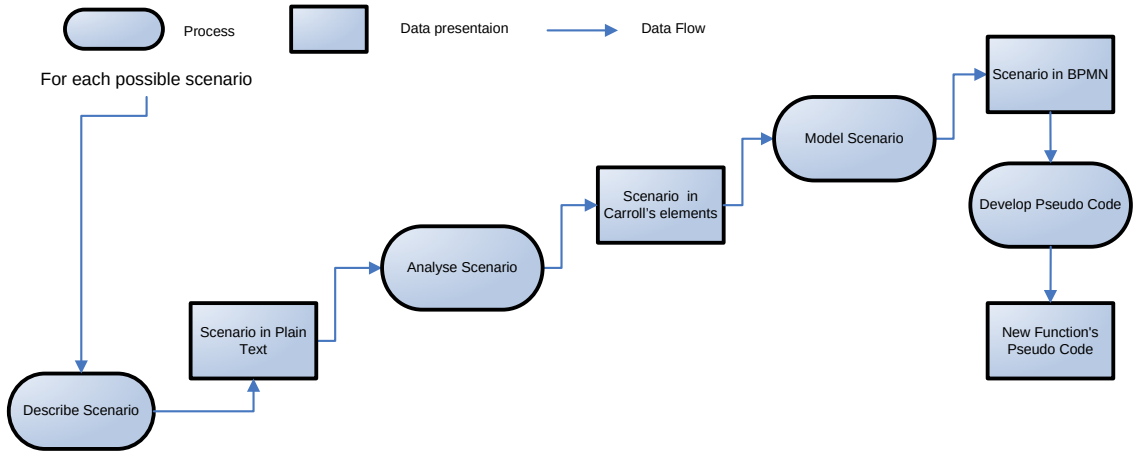

Figure 6-4: Steps of Requirements Elicitation Stage

6.3.1.1 Scenarios Description

Scenarios description can be done using a number of methods. These methods allow scenarios to be captured in different kinds of formats, such as text narratives, sketches, screen shots and informal media. As the analysis of the collected usage scenarios progresses, the former informal representations are replaced by appropriate models. In these more formal models, a more structured approach - such as event scenarios or use cases - can be utilised. Modelling of scenarios is normally restricted to the essential set of facts that are useful in shaping the new system. Also, the modelling effort is limited by time and available resources.

In the effort to capture and model scenarios, several techniques and tools have been introduced in the requirements engineering literature in order to address these issues. For instance, ScenIC [150] introduced a schema of objectives, tasks, obstacles and actors. In this schema, scenarios are scenario-related knowledge made up of goals, episodes and actions performed by actors. Actors are generally people, but can be machines too. Another example of tools that aim to support the scenario-based requirements engineering is ScenarioPlus [157]. Scenario Plus captures, verifies, animates and plays back scenarios. The tool allows stakeholders to describe their requirements to software developers. Scenario Plus is composed of a set of add-on tools, such as the use case toolkit and diagrams toolkit to be used for scenario-based

requirement elicitation and analysis.

For an example of a plain text scenario, one might consider a library where an RFID-based system has been implemented for document tracking, charging and discharging of documents, security of materials, inventorying, stock verification and shelf handling. In this type of libraries, consider the scenario shown in Figure 6-5. The scenario demonstrates the new method of promoting a library's services after the employment of an RFID system. RFID were discussed in Chapter 2, Section 2.3.4.

The library has decided to change its current promotion method of the new services and the new book collection. In the current method, the service and new collection are announced for all of the library patrons through emails or through the LCD screens positioned near the setting and study areas. The services are prompted in a generic fashion without considering the different types of the library patrons and their variety of interests. After utilising RFID, the library decided to install a number of RFID readers next to the LCD screens. This was done in addition to replacing the current library patrons' cards with RFIF enabled ones. Having done this, by using the RFID system the library will be capable of tracking the different locations in which a library patron may have spent some time, inside the library premises. The analysis of these locations is loggged into the patron's book interest profile, which is updated accordingly. As a result of this operation, promotion of a new service or new book will be customised for a specific patron if he is sitting by himself, for a group of patrons through looking at a possible common interests, or a general promotion otherwise.

Figure 6-5: Example of a Scenario in Plain Text

6.3.1.2 Scenarios Analysis

According to Carroll [157], scenarios have the following characteristic elements:

- Setting (context of the environment)
- Agents or actors
- Sequences of actions and events
- Goals

A scenario includes at least one actor and at least one goal. The actor (human)

completes a sequence of tasks (observable behaviour) to accomplish a specific goal in the circumstances of the context of the environment (setting). Events are external actions done by the computer system or other features of the setting.

For the scenario presented in the previous section, Section 6.3.1.1, the scenario elements based on Carroll method are:

- **Setting**: is a library setting and study areas. The scenario implies further setting elements by identifying the person as a patron or a group of patrons. The library uses an RFID system in its operations and LCD screens as bulletin boards.
- **Actor** or agent: the patron or group of patrons are the actors in this example.
- **Actions**: sitting or reading a book in the study or reading areas.
- **Events**: retrieving the patron or group of patrons' profiles, displaying the appropriate promotion of the LCD screen.
- **Goal**: displaying a promotion on the LCD screen.

6.3.1.3 Scenarios Modelling in BPMN

The business logic is modelled in BPMN. BPMN, as a *de facto* standard notation depicts the steps in a business process which is conceivable by all the system's stakeholders. Only the user function requirements elicitation is considered in this stage. User requirements were discussed previously in Chapter 2, Section 2.2.4. The modelled new requirements serve two purposes. The first purpose is to determine the new business logic which represents the additional requirements after the introduction of the ubiquitous technology. The second purpose is for it to be the second input in the integration stage, explained in Chapter 7.

In order to view the scenarios in BPMN notation, the technique proposed and presented in [99] was adopted. The technique starts by writing the scenarios in a use

case form. A use case is a "description of the possible sequences of interactions between the system under discussion and its external actors, related to a particular goal." [39] The technique restores an overview of the use cases, and visualizes the control flow of the resulting business logic in BPMN notation. The approach assumes use cases to be in a tabular, semi-structured form and uses pre-conditions, post-conditions, and triggers use cases to assemble business logic automatically.

For the example scenario, discussed in section 6.3.1.3, the technique adopted will write the scenario in a tabular form similar to that depicted in Figure 6-6. Then the algorithm of the technique shown in Figure 6-7 will be employed to produce the BPMN diagram shown in Figure 6-8.

Use case No.x	LCD Promotion
Precondition	(none)
Trigger	Patron(s) is sitting near the LCD
Post-condition	Promotion is displayed
Main Scenario	1. Check for patron(s) 2. Check for patron profile 3. If one patron with profile then display patron's interest. 4. If more than one patron with profiles then display common interest promotion. 5. If no profiles then display generic promotion.
Extensions	1a if no patrons near LCD then continue check for patrons

Figure 6-6: Scenario in Use Case Tabular Form

```
1: Function ConvertScenario(Scenario, LastElement):
2: for all   Step IN Scenario.Steps do
3:     if   Step.IsJumpTarget then
4:           XORGateWay := P.add(new XORGateWay());
5:          LastElement.connectTo(XORGateWay);
6:          LastElement := XORGateWay;
7:    end if
P.add(new Activity(Step));
8:    if   Step.isExtended then
9:          XORGateWay := P.add(new XORGateWay());
10:        LastElement.connectTo(XORGateWay);
11:        LastElement := XORGateWay;
12:       for all Extension IN Step.Extensions do
13:          ConvertScenario(Extension, LastElement);
14:       end for
15:    end if
16: end for
17: if Scenario.JumpsBack then
18:     LastElement.connectTo(
            GetXORGateWayFor(Scenario.JumpTarget));
19: end if
```

Figure 6-7: Conversion of Scenarios to BPMN Algorithm [99]

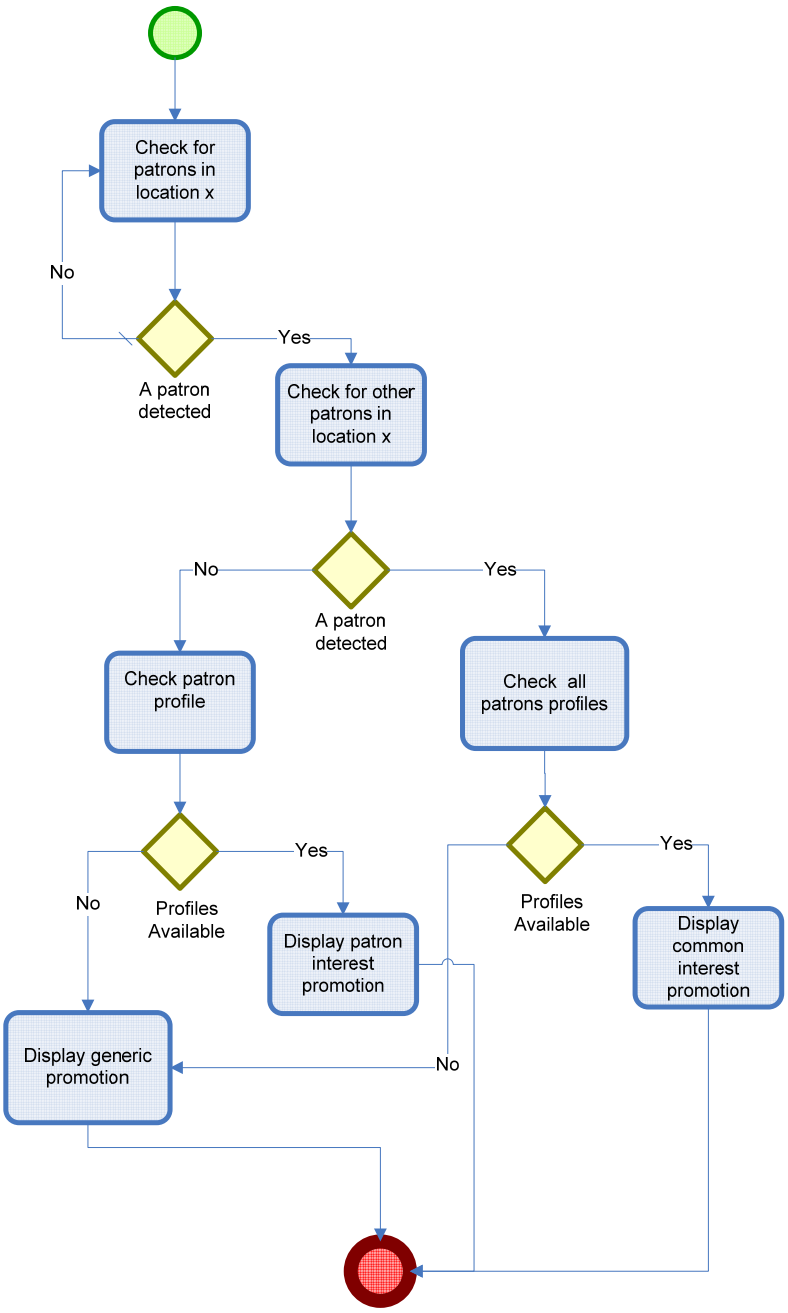

Figure 6-8: Example of Scenario Modeled in BPMN

6.3.1.4 Pseudo Code Development

The last step in the proposed additional-requirements engineering stage is the development of the code at the pseudo code level for the new requirements. From the modelled scenarios the new functions or the functions which need to be modified are identified. For example for the scenario illustrated in Figure 6-8 a new function needs to be added to the system to implement the LCD screen promotion, based on the detected library's patrons. The pseudo code for the new function can be something similar to what is shown in Listing 6-1

```
public void promoteAService(String readerIp,List tags)
{   //get the LCD using the reader IP
   currentLocationLCD=this.getLCDByReaderIp(readerIp);
   //if there was only one tag (patron) in the range of the reader
   if(tags.size()==1)
   {      //use the first and only tag in the list
      patronTag=tags.get(0);
      //get the patron profile using his tag id
      patron=this.getPatronByTagId(patronTag.getId());
      //using the method getNewPromotedService which is assumed to do
some data minning and promote a new suitable service for the patron
      promotedService=patron.getNewPromotedService();
      //update patron's history
      patron.updatePromotedServices(promotedService);
   }
   // else it's more than one patron in the range
   else
   {      //get a general promotion service
      promotedService=this.getGeneralPromotionService();
      //get an iterator to iterate the tags list
      tagsIterator=tagList.iterator();
      //while there;s still tags in the list
      while(tagsIterator.hasNext())
      {         // get the tag on turn
         patronTag=(Tag)tagsIterator.next();
         //get the patron profile using his tag id
         patron=this.getPatronByTagId(patronTag.getId());
```

```
            //update patron's history
            patron.updatePromotedServices(promotedService);
        }    }    //display the service message on the LCD
    currentLocationLCD.display(promotedService.getMessage());
```

Listing 6-1: Example of Function Pseudo Code

6.4 Summary

In the introduction of the chapter, the general factors that can force an organisation to opt for the BPR process were discussed. In addition, three levels of reengineering that can be carried out on a certain process have been provided. These three levels are: process reengineering at the highest, process simplification and process improvement at the lowest. Moreover, the role of IT in general was described in five levels of information-technology-based transformations. Then the possible impacts of the utilisation of ubiquitous computing technologies were discussed.

Ubiquitous technologies, such as automatic identification, localisation, and sensor technology, can give many new products and services that can grant different kinds of enterprises a competitive advantage and improve their productivity. Examples presented in this chapter show how ubiquitous technologies have influenced a variety of fields.

Later in the introduction, an overview of the RE process which includes requirements elicitation, analysis, specification and validation was given. Since the focus of this research is on requirements elicitation, the term was defined. Moreover, the methods used in requirements elicitation such as interviews, inspection, observation and finally scenarios, were briefly discussed.

Since in this research a scenario-based approach was adopted for requirements elicitation, the rationale behind this selection was presented. In addition, more explanations were given about the use of scenarios in the requirements elicitation tasks.

Thereafter, the steps of the additional-requirements engineering stage were presented. These steps are: scenario description in plain text, scenario analysis using Carroll’s

method, scenario modelling using BPMN, and pseudo code development. All the steps of the stage were explained with supporting examples from a library management system.

Chapter 7

Integration Stage

Objectives

- To discuss the role of the integration stage within the reengineering framework
- To illustrate the developed integration algorithm.
- To highlight the considerations of the integration process.
- To discuss the integration pattern that can be with the hardware handlers.

7.1 Introduction

In the reengineering framework, the integration stage comes before the last stage, testing and operation stage. As pointed out earlier, the proposed framework is not attached to a specific software development approach or a specific programming language. Nevertheless, the discussion of this stage will use the Component-Based Software Development (CBSD) to discuss the integration algorithms and techniques proposed in this stage. CSBD was selected because the software systems targeted by the framework are composed of components. In addition the Object Oriented software systems are also considered component based systems [169].

In CBSD, integration in general has been discussed widely in the literature. Prior to mentioning the integration algorithms and techniques devised in this stage, it is worth revising the basic concepts which will be used later in this chapter. As pointed out earlier, CBSD is used merely to illustrate the purpose of this stage. Hence, the definitions of the term were selected without making reference to the other definitions available in the literature. A component is defined in [144] as "a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard." An interface is defined in [42] thus "An interface specifies the access points to a component, and thus helps clients to understand the functionality and usage of a component. The interface is clearly separated from the implementation of a component." In the literature, there several definitions for the term integration, in [42] it was defined thus "Component integration is the mechanical task of wiring components together by matching the needs and services of one component with the services and needs of others."

Sommerville [144] described this process as component composition which has a number of types: sequential, hierarchical and additive. Sequential composition occurs when the constituents are executed in sequence in which extra code is needed to link the components. Hierarchical composition happens when a component directly calls on the services offered by another component, while in additive composition, the interfaces of two or more components are added to construct a new component.

The aim of this stage in the reengineering framework is the process of assembling the developed or changed components to create a system as a whole.

This stage as depicted in Figure 7-1, has two objectives:

1. Integrating New/Changed Components: to achieve this objective an integration algorithm has been developed. The devised integration algorithm makes use of both the extracted business logic and the new determined business logic, both of which are represented in BPMN. The integration process is at a higher abstract layer, which in our approach is at the diagram level. The algorithm runs a comparison between the two BPMN diagrams; this can be done either manually or automatically. The result of this comparison leads to either adding or modifying a component in the existing system.

2. Integrating Hardware Handlers: to achieve this objective one of the recognised integration patterns can be employed. Such patterns include File Transfer, Shared Database, Remote Procedure and Invocation and Messaging pattern. The hardware in this case can be a sensor or an RFID. These technologies can be integrated through using one of the above integration patterns where suitable depending on the situations [76].

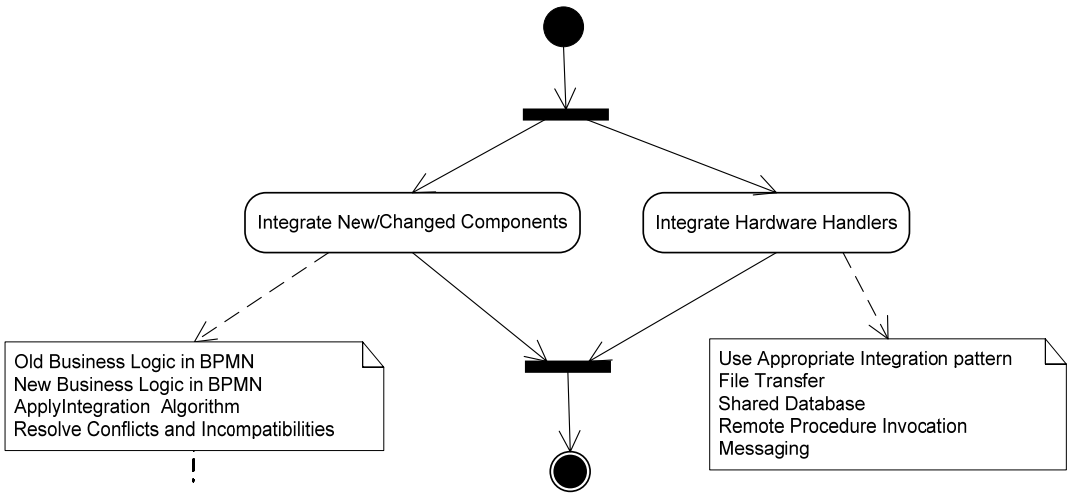

Figure 7-1: Integration Stage Objectives

7.2 Components Integration

Integration as defined in the preceding section, Section, 7.1, and as normally discussed in the software engineering literature, is concerned with answering the 'how' question of the integration process. In this research, the term 'integration' has been used with a slightly expanded objective. As shown in Figure 7-2, the process starts with the extracted old (existing) business logic and the new business logic, both of which are represented in BPMN. The business logics (new and old) are compared either manually or automatically using the integration algorithm, explained in detail in Section 7.2.1. Based on the outcome of the comparison, components may be added, modified or removed from the system. During the integration process, incompatibilities normally occur. These possible incompatibilities can be resolved using the techniques discussed in Section 7.2.2.

This objective of the integration stage in the proposed reengineering framework encompasses two sub-goals that aim to address three questions of the integration process: what, where and how.

- What: specifies the function that needs to be added, changed or deleted.
- Where: identifies the location of the change to be performed.
- How: explains how the integration is carried out using the available integration techniques.

The 'what' and 'where' questions of the integration process are addressed in the integration algorithm which is presented in the next section, Section 7.2.1. The 'how' question of the integration process is addressed in integration considerations discussed in section 7.2.2.

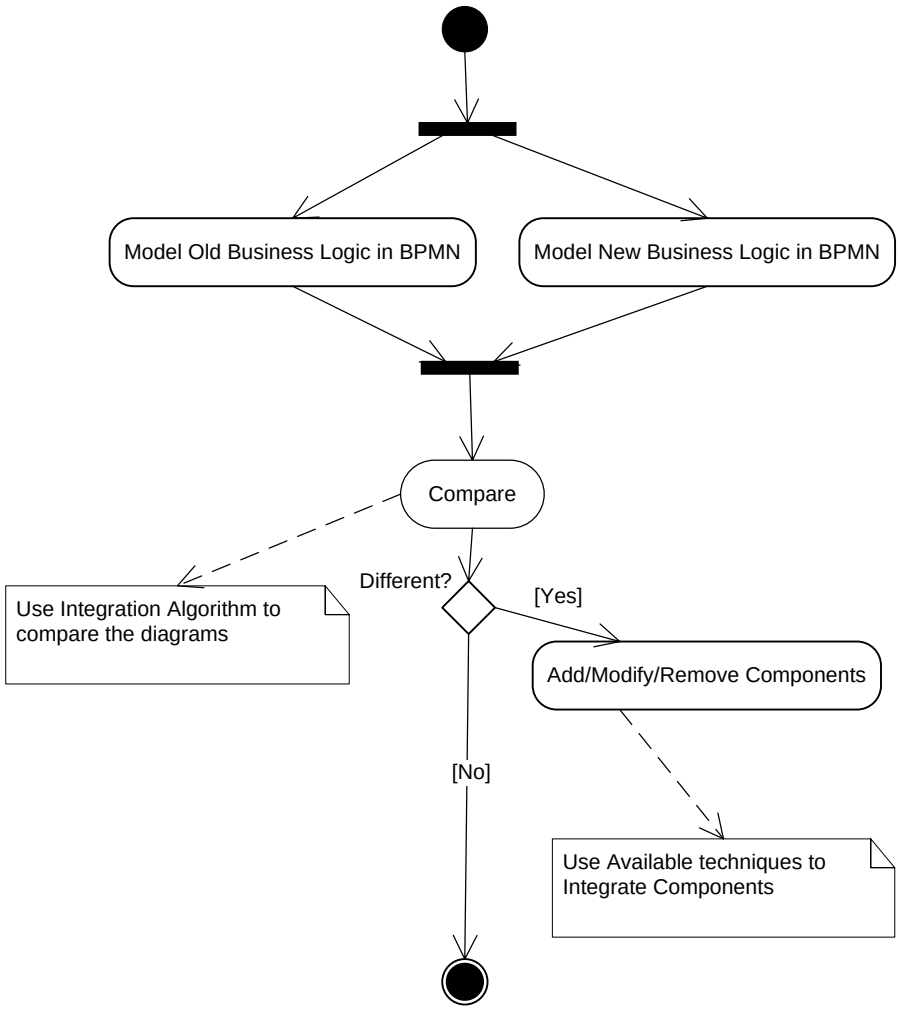

Figure 7-2: New Components Integration Steps

7.2.1 Integration Algorithm

The algorithm aims to answer the 'what' and 'where 'questions. These questions identify the software components which are required to be added, changed or deleted to satisfy the new requirements resulting from the introduction of the ubiquitous technologies. In order to understand the logic of the algorithm, it is important to explain the meaning of the term 'business rules' used in this algorithm. A business process "consists of a set of activities that are performed in coordination in an organisational and technical environment." [71] An activity (task) is "a logical unit of work that is carried out as a single whole by one resource." [1] An example of a task in

a borrowing book process is checking that the potential borrower is a member of the library. A constraint in this example can be that borrowing is allowed for members only. In this algorithm, a business rule has a different semantic. In the algorithm, a business rule is composed of the following elements, as shown in Figure 7-3:

- Previous Activity
- Constraint
- Post Activity

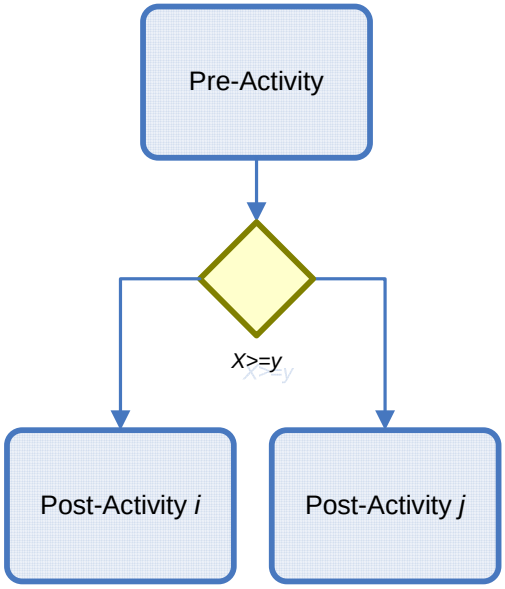

Figure 7-3: A business Rule in Algorithm One

The two activities, previous activity (pre-activity) and post activity, are represented by the BPMN activity element. In a similar manner, the constraint (condition) is represented by the BPMN gateway element. This definition of a business rule will be used in the description of the algorithm illustrated in Figure 7-4

```
For each function Identify all possible Business Rules
in the old and new business logic.
FOR all identified business rules DO
IF      Pre_Activity_New NOT FOUND in Pre_Activity_old
THEN
        Add New_Business Rule
ELSE
    IF  Post_Activity_New NOT FOUND Post_Activity_Old
    THEN
        Replace  Post_Activity_Old
    END_IF
END_IF

IF      Pre_Activity_old  NOT FOUND in Pre_Activity_New
THEN
        Delete  New_Business Rule
END_IF
END_FOR
```

Figure 7-4: Integration Algorithm

Having the existing business logic and new business logic both represented in BPMN, the comparison algorithm is executed as follows:

- **First**: based on the definition of a business rules explained in this section all business rules in both BPMN diagrams which view the old and new business logic are identified.

- **Second**: for all identified business rules the following checks (C) are performed:

- C1: if a previous activity of a new business rule does not match (≠) any previous activity of all old business rules, then the components of the entire new business rule are added.

- C2: if a previous activity of an old business rule does not match (≠) any previous activity of all new business rules, then the components of the entire old

business rule all deleted

- C3: if a previous activity of a new business rule matches (=) any previous activity of an old business rule, but the post activity of the new business rule does not match (≠) the post activity of the old business rule, then only the components of the post activity of the new business rule replace the post activity of the old business rule.

- **Third**: End of algorithm

To illustrate the above algorithm, let us consider the following example cases from a library management system. The library system manages the basic functions, such as book borrowing and book return, in addition to managing the data for library books and the library borrowers. The new business logic introduced to the library system is a result of utilising RFID technology. By using the RFID, the system is capable of tracking the different locations where a library patron may have spent some time, inside the library premises. Through analysing these locations, the log book containing the interest profile of the patron is updated accordingly.

Case 1: *if a previous activity of a new business rule does not match (≠) any previous activity of all old business rules then the components of the whole new business rule are added.* An *e*xample of this case is shown in Figure 7-5 (old business rule) and Figure 7-6 (new business rule).

- In the old business logic: if the book is not available, the patron cannot borrow the book and the patron is informed that the book is in loan.

- In the new business logic: if the book is not available the patron cannot borrow the book, the patron's profile is checked and if it is not a new patron an alternative book is offered based on his profile.

- In this case, the pre-activity in the new business logic "Check Patron Profile" does not match any of the pre-activities in the old business logic.

- In this case, the components that implement the patron's profile check are added (what to add) after checking the book's availability (where to add).

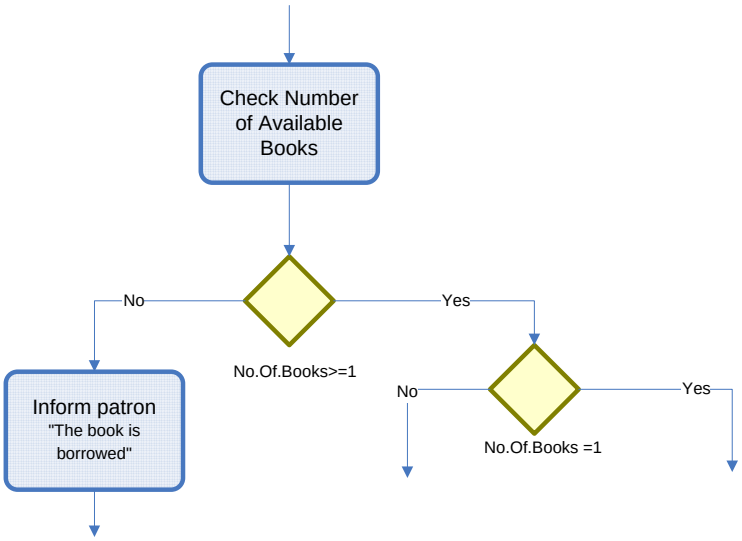

Figure 7-5: Case One Old Business Rule

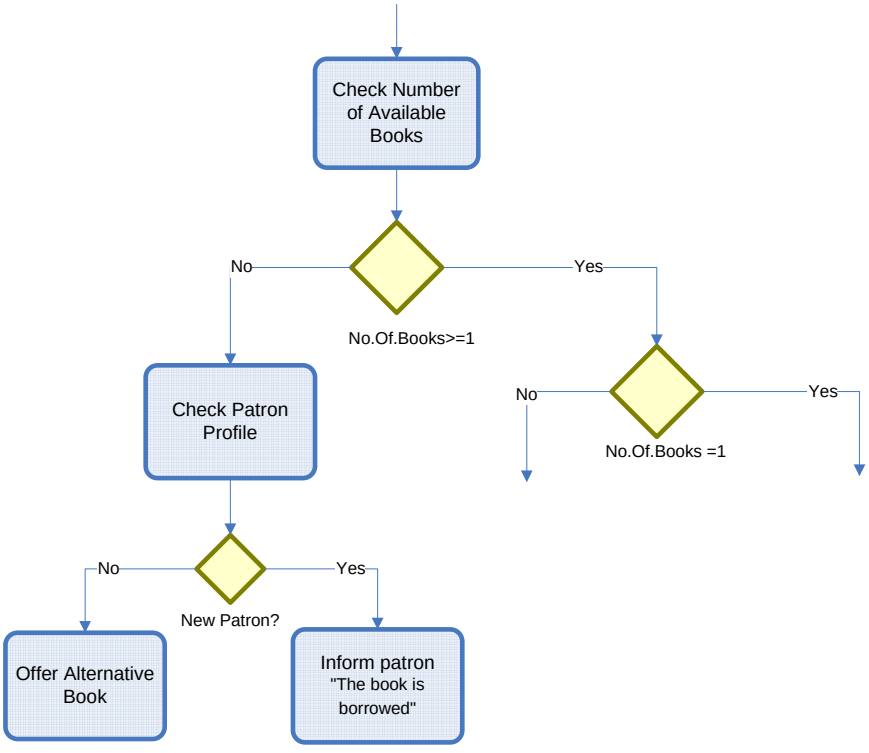

Figure 7-6: Case One New Business Rule

Case 2: *if a previous activity of an old business rule does not match (≠) any previous activity of all new business rules then the components of the whole old business rule are deleted.* An example of this case is shown in Figure 7-7 (old business rule) and Figure 7-8 (new business rule).

- In the old business logic: the data is entered manually. Hence, the entered data need to be validated before proceeding to the next activity.
- In the new business logic: the data is captured automatically using the RFID reader and tag. Hence, the old validation activity is not needed anymore.
- In this case, the pre-activity in the old business logic "Check Entered Data" does not match any of the pre-activities in the new business logic.
- In this case, the components which implement the data check are deleted (what to delete) after reading the data (where to delete).

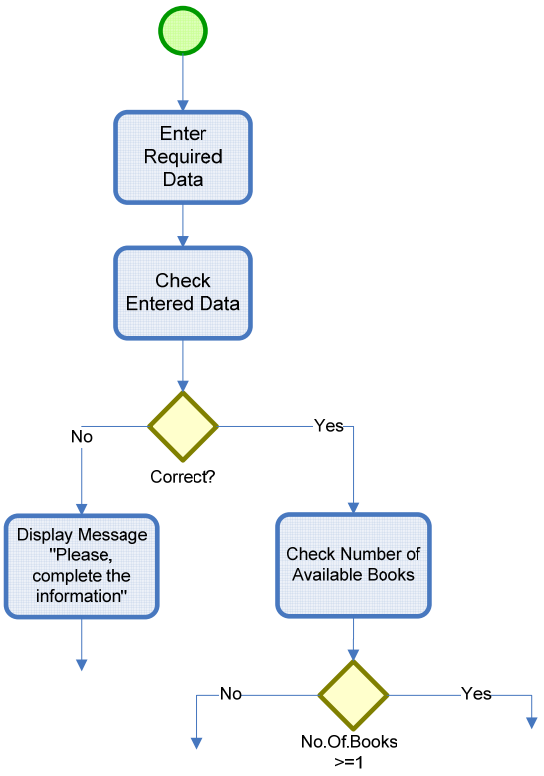

Figure 7-7: Case Two Old Business Rule

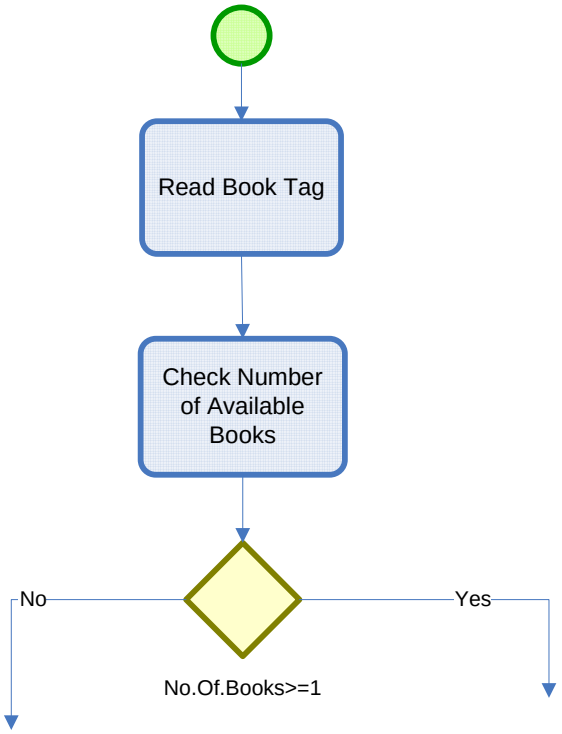

Figure 7-8: Case Two New Business Rule

Case 3: *if a previous activity of a new business rule matches (=) any previous activity of an old business rule, but the post activity of the new business rule does not match (≠) the post activity of the old business rule then only the components of the post activity of the new business rule replace the post activity of the old business rule.* An example of this case is shown in Figure 7-9 (old business rule) and Figure 7-10 (new business rule).

- In the old business logic: if the book is available, the book is lent and if it is the last book, it is flagged as an unavailable book. Then the book borrowing process ends.

- In the new business logic: after doing the steps in the old business logic, the patron ‘s profile is checked in order to promote books or services that match his or her interests.

- In this case, the pre-activity in the new business logic “Lend the Book” matches the pre-activities in the old business logic, but the post activity in the new

business logic “Check Patron Profile” does not match the post activity in the old business logic “End”.

- In this case, the components which implement the patron’s profile check and promotion of services are added to replace the old logic (what to replace) after the lending-the-book activity (where to replace).

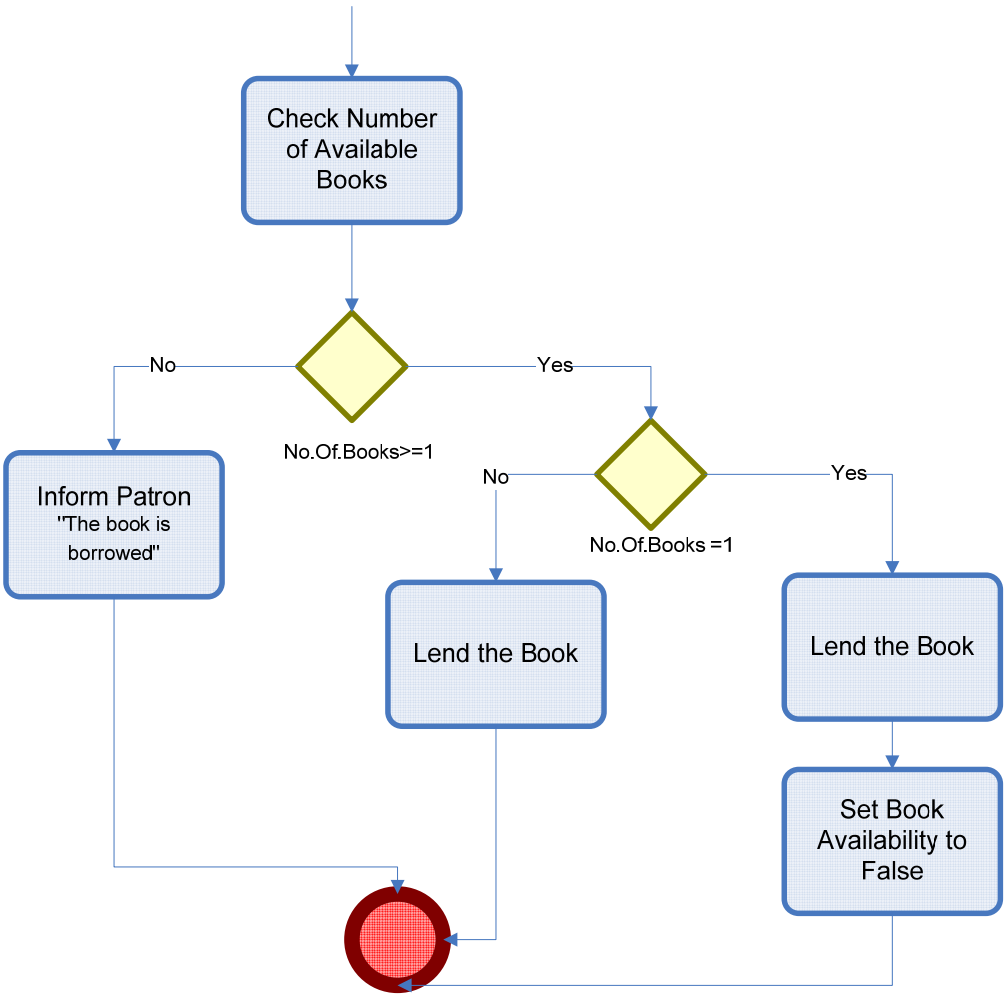

Figure 7-9: Case Three Old Business Rule

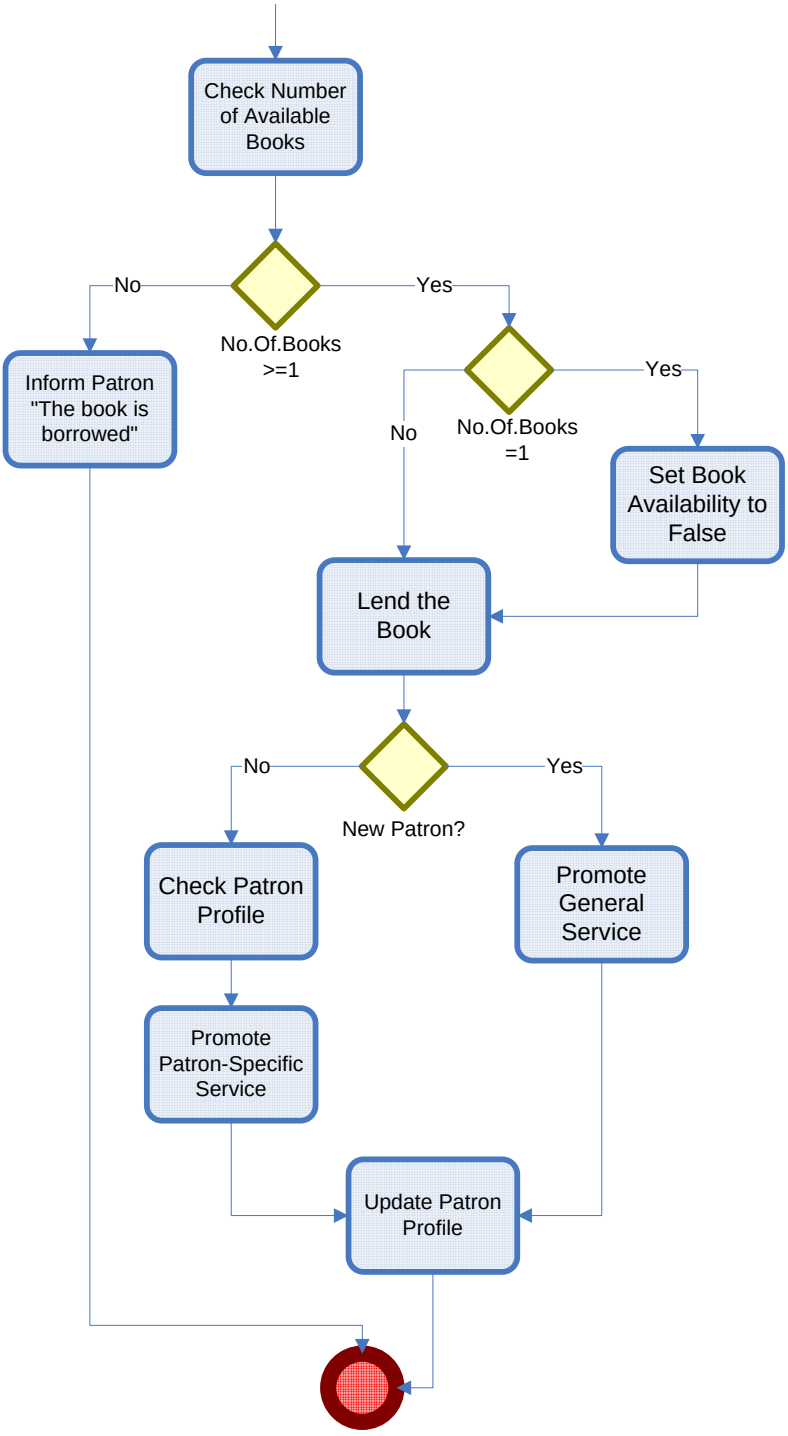

Figure 7-10: Case Three New Business Rule

7.2.2 Integration Considerations

In Section 7.2.1, the integration algorithm aimed at facilitating the integration process through identifying the required components in addition to the location where these components need to be integrated. The algorithm lies on a high level abstract of the system which is at a diagrammatic level that depicts the business logic. However, in order to carry out a complete integration process for the implemented components, the low level of integration issues need to be addressed. These considerations answer the 'how' question of the integration process. Since the low level of integration process is not within the focus of this research, in this section only some of the general considerations of any integration process, along with their possible solution, are presented. The information discussed here were based on literature in CBSD integration such as [42, 73, 95]

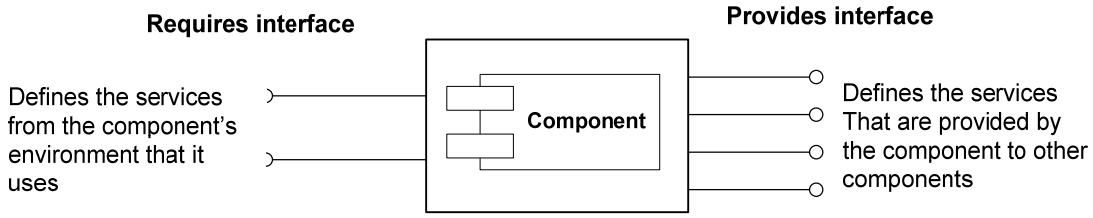

Figure 7-11: Requires and Provides Interfaces

One of the issues that need to be solved in any integration task is the interface incompatibilities of components. Figure 7-11 shows the roles of the 'provides' interface and the 'requires' interface of a component. Three types of incompatibilities (conflicts) that can occur are identified:

1. Interface incompatibility: in which the operations on each side of the interface have the same name but the number of parameters or their types are not the same.

2. Operation incompatibility: in which the names of operations in the 'provides' interface are different from the operations in the 'requires' interface.

3. Operation incompleteness: in which the ‘requires’ interface of a component is a subset of the ‘provides’ interface of another component, or the opposite.

The above possible conflicts can be resolved using a number of software integration techniques or components. These integration techniques - known as components adaptation - include using wrappers and adapters. An adapter is often referred to as a glue code.

Adapter:

An adapter (glue) code provides a solution for the interface incompatibilities issues by reconciling the interfaces of the components that are integrated. Precise forms of adaptors are required depending on the type of integration. For instance, an adapter role can be simply to take an output from one component and convert it into a form where it can be used as an input to another component.

Adapters can be employed for a number of reasons, such as control flow, component bridge and exception handling. In control flow, the adapter invokes the functionality of the underlying components as required, while in components bridging the adapter is used to resolve the possible interface incompatibilities via data conversions. In exception handling, the adapter can provide a consistent exception handling method through working as a trap for exceptions.

Wrapper:

Generally speaking, wrapping is a practice that transforms a component’s software interface from one form to another. Wrapping is used to eliminate mismatches between the interface exported by software artefacts and the interfaces required by current integration practices. Wrapping is normally utilised to modernise legacy systems at user interface, data or the functional level.

To address the compatibility problems mentioned earlier, a wrapper is a piece of code that can be used in the integration process to isolate the underlying component from other components, and provide a single point of access to that component. In object

oriented, one of the applications of wrapper classes is to wrap around objects in order to provide more functionality and/or flexibility by wrapping streams with classes that provide particular capabilities as needed. For example, the Java I/O framework uses wrapper classes to build specialised I/O streams.

7.3 Integrating Hardware Handlers

There are available several ubiquitous technologies hardware that can be used to enhance the functionality of existing systems. Such technologies provide various services, such as automatic identification, localisation, and sensor technology. Automatic identification (Auto-ID) systems use hardware such as barcode, RFID tags and reader, smart cards, and biometric devices. Localisation systems use other form of hardware such as GPS and RFID. In addition, there are many services that can be provided by systems through sensing their surrounding environment using different types of sensors. Different sensor types are available, including thermal, visual, infrared, magnetic, seismic or radar sensors. These sensors are used to monitor conditions like humidity, temperature, pressure, lightning conditions, mechanical stress levels on attached objects, noise levels, or current characteristics such as the size, direction and speed of an object. These sorts of hardware need to be integrated with heterogeneous systems to work together to achieve a set of functions.

Not all systems have a built-in integration mechanism to work seamlessly with other systems. Most of the time, it is difficult to add such a facility by pure reengineering techniques, especially for legacy systems. Therefore, integration patterns can provide solutions to most of these issues. The use of integration patterns to integrate such systems presents a number of advantages, such as the following:

- Integrated systems dependencies on each other are minimised so that each can be modified without interfering with the others.
- Changes to the system and the amount of integration code are minimised during integration.

- The format of the data exchanged between integrated applications is agreed on between them. An intermediate translator can unify applications with different data format requirements.

- In addition to data integration, functionality can be shared as well, which can provide better abstraction between systems.

Different patterns for integrating systems can be taken advantage of in various situations. These integration patterns as classified by Hohpe and Woolf [76] fall into four main categories: File Transfer, Shared Database, Remote Procedure Invocation and Messaging.

7.3.1 File Transfer

In a File Transfer integration pattern, each system produces files of shared data consumed by other systems, and consumes files produced by other systems. The role of the integrator in this pattern is transforming files into different formats and producing files at regular intervals based on the nature of the business. Figure 7-12 shows a File Transfer Integration pattern.

Figure 7-12: File Transfer Integration Pattern

In a File Transfer integration pattern, integration can be attained without the need for exposing the internal implementation details of a system. This means that the integrated applications are quite independent from each other. As long as systems produce the same data in the same format, making internal changes to one system will have no effect on other integrated systems. In a File Transfer integration pattern, files can be regarded as the public interface for each application.

7.3.2 Shared Database

Although File Transfer allows integrated systems to share data, it can lack timeliness which is a critical factor in systems integration. Even with rapid updates, inconsistencies remain a problem when every system must have the latest data. In addition, File Transfer may not be capable of effectively enforcing data format owing to the incompatible ways of observing the data. These concerns are addressed in the Shared Database integration pattern. Figure 7-13 shows a Shared Database integration pattern.

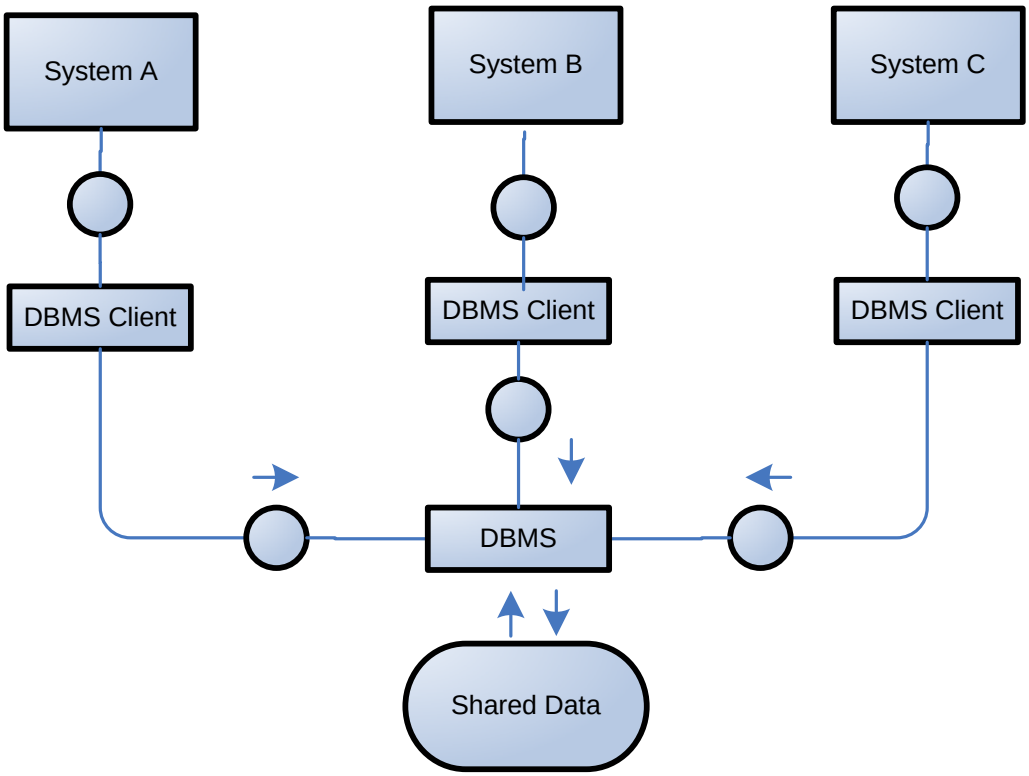

Figure 7-13: Shared Database Integration Pattern

In a Shared Database integration pattern, systems store the shared data in a common agreed-upon data store that all of the systems share. A Shared Database integration pattern allows information to be shared in a fast and consistent way through using a transaction management system. A Shared Database integration pattern can be used for applications that are being built independently with various languages and platforms.

Compared with File Transfer, since application development platforms can work with Structure Query Language (SQL), transformation of data is unnecessary, and the semantic dissonance problem is solved in Shared Database.

7.3.3 Remote Procedure Invocation

File Transfer and Shared Database enable integrated systems to share data among them. However, sharing data is often not sufficient, and changes in data often demand actions to be taken among different systems. Hence, the Remote Procedure Invocation (RPI) integration pattern has addressed this issue. In RPI, each application discloses some of its functions through appropriate interfaces. These interfaces allow factions to be invoked remotely by other systems to initiate actions and exchange data. Figure 7-14 shows an RPI pattern.

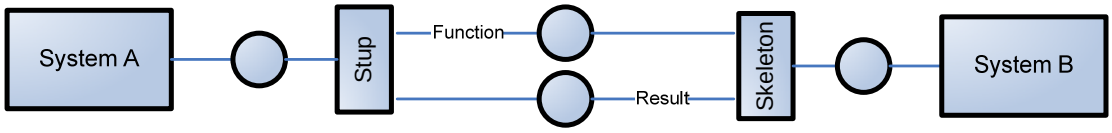

Figure 7-14: Remote Procedure Invocation Integration Pattern

RPI applies the principle of encapsulation, in which each application is a separate component with encapsulated data, which permits an interface to allow other systems to interact with other systems through function calling. This concept of encapsulation permits each system to retain the integrity of its data. Moreover, each system can alter its internal data format without interfering with the other system. The use of RPI makes it easier to deal with semantic dissonance via providing separate interfaces for different kinds of clients. In addition, the implementation of RPI is usually easy to do because its programming is very similar to a local method call.

7.3.4 Messaging

File Transfer and Shared Database allow data to be shared by integrated systems, but not functionality. RPI allows systems to share functionality but with the systems tightly

coupled. The tight coupling causes a synchronisation problem as one component waits while the other completes the procedure invocation. Also, it is not simple to be changed and developed independently because of the interface dependency. Therefore, a messaging integration pattern can be an alternative option. Figure 7-15 shows Messaging integration pattern.

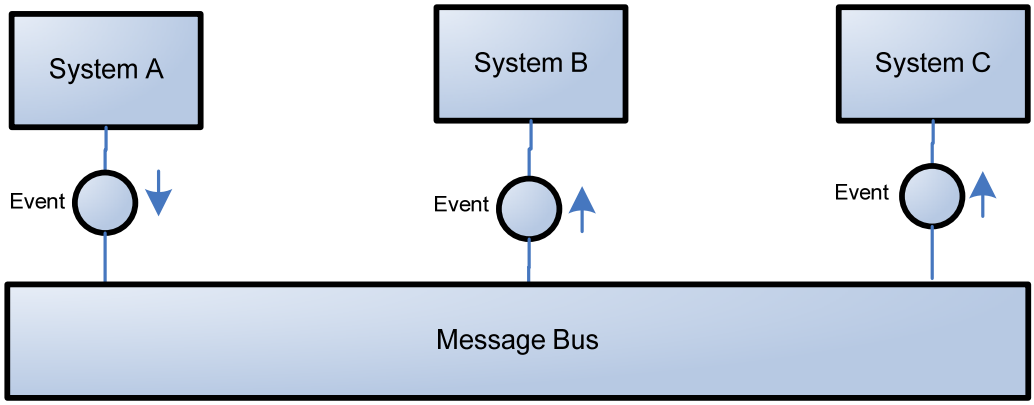

Figure 7-15: Messaging Integration Pattern

In a messaging integration pattern, each system connects to a common messaging bus to exchange data and invoke behaviour using messages. A messaging integration pattern works in an asynchronous way similar to File Transfer, but with rapid updating, automatic retry until success and automatic notification. A messaging integration pattern can create and transfer a number of small data packets quickly, while the receiver is notified when a new packet is available for consumption. Moreover, messages can be broadcast to many receivers, which does not require both systems to be up and running at the same time.

7.4 Summary

This chapter focuses on the integration stage of the proposed reengineering framework. At the beginning of the chapter, the terms related to the stage were defined. Since CBSD was used to illustrate the process of the integration stage, the terms component, interface and integration were discussed.

The aim of the integration stage was simply the process of assembling the developed or changed components to create a system as a whole. The stage encompasses two objectives: integrating the new/change components and integrating the hardware handlers.

In integrating the new/change components, an integration algorithm was developed. The devised integration algorithm makes use of both the extracted business logic and the new determined business logic, which are represented in BPMN.

In the algorithm, the term ‘business rule’ was defined within the context of the algorithm usage. The business rule, as defined in the algorithm, is composed of the following elements: Previous Activity, Constrain and Post Activity. The algorithm was explained using example cases to cover all the possible cases that can result from the three checks in the algorithm.

Then, the considerations which answer the ‘how’ question of the integration process were highlighted. The general considerations of any integration process along with their possible solution were given. The solution covered two approaches: adapter and wrapper.

At the end of the chapter, the integration of the hardware handlers of the ubiquitous technologies that can be utilised was discussed. The hardware needs to be integrated with heterogeneous systems to work together to achieve a unified set of functionality. Integration patterns which can provide solutions to most of hardware integration issues were provided. The integration patterns included: File Transfer, Shared, Database, Remote Procedure Invocation and Messaging. Each pattern has been explained along with the integration pattern for which a solution can be provided.

Chapter 8

Case Study

Objectives

- To review the tools experimented with but not used.
- To review the tools used in the case study. .
- To show how the proposed reengineering framework stages can be used to reengineer an existing system.
- To show how to use the selected reverse engineering tools in the software reengineering process.

8.1 Introduction

This chapter presents a case study which aims to evaluate the proposed reengineering framework. The objectives of the case study focus on implementing the various techniques and algorithms introduced in the framework. The case study covers the first three stages of the framework which were discussed in detail in the previous chapters. These stages are program understanding, additional-requirements engineering, and integration. The case study is conducted on a library management application. Background information about the library system used is given in the next section, Section 8.2.

Throughout this research, many tools have been experimented with. Some of the experiment tools have been used in this case study and some not. Brief descriptions of all the tools that have been experimented with are given in Section 8.3. The first stage of the framework, which is the program understanding, is demonstrated in Section 8.4. The second stage, new requirement engineering, is given in Section 8.5. At the end of the chapter, the third stage covered by this case study is presented in Section 8.6.

8.2 Background of the Library Management System

To demonstrate the reengineering approach, the proposed approach has been applied to a library management system. This free licence software is provided by *Planet-Source-Code.com* for academic purposes with the source code, which is free to use, to modify and to be changed as required.

The library management system is written in Java. It supports the main functions operated in a library. The software uses the Microsoft Access database. The system was used in this research as an example to demonstrate the implementation of the proposed reengineering framework. Figure 8-1 shows a screenshot of the system.

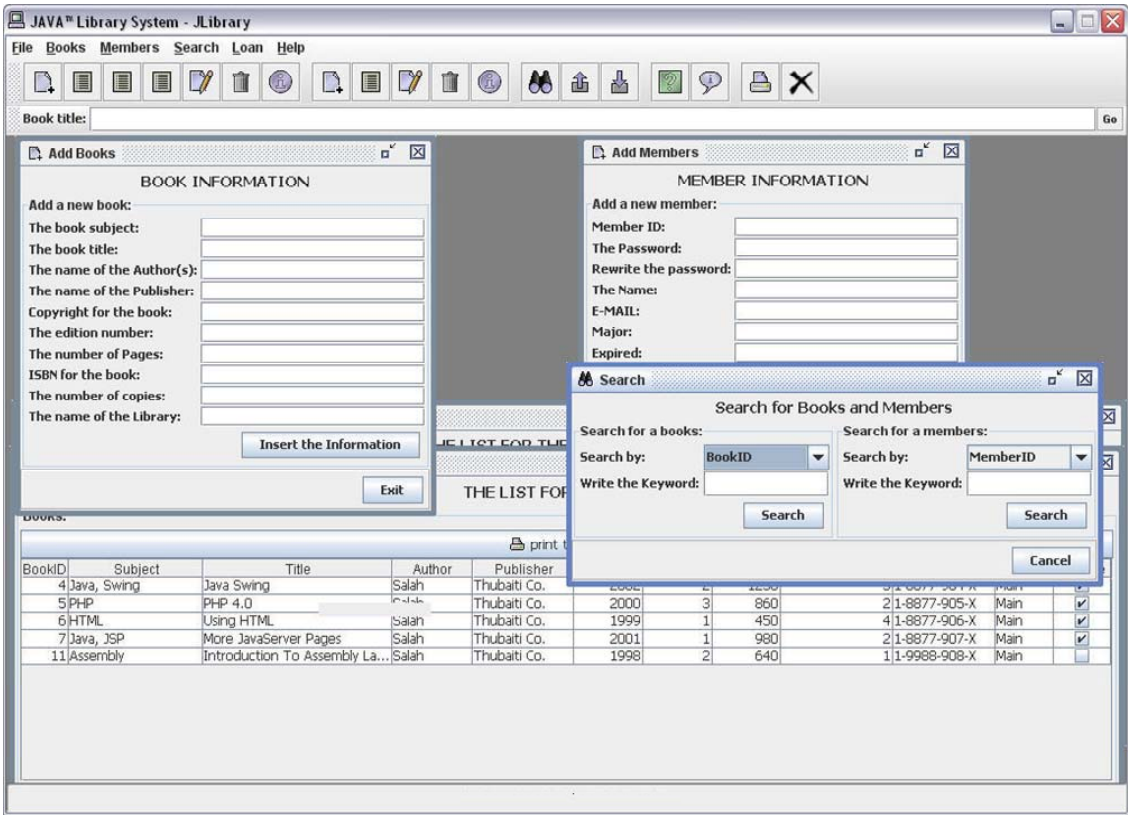

Figure 8-1: A Screenshot of The Library Management System

In a simplified library management system, the library holds only books. Each book is uniquely identified. Library members can request some of these books for loan, subject to proper borrowing rules. In order to borrow a book, users must be identified. For example, this could be achieved by distributing library cards to registered members.

With regard to the management of the books in the library management system, the librarian can add new books in the library and remove books no longer available in the library. Upon request, the librarian may need to search the database for books according to some search criterion, such as title, authors, ISBN code, etc. As far as the management of library members is concerned, a set of personal data (name, address, phone number, etc.) is maintained in the database. Similar to the book management, the librarian may need to search the database for members according to some search criterion, such as ID, name, email, etc

The main functionality of the library management system is loan management. The library management system performs the following basic functions which can be classified according to the following three categories, as shown in Figure 8-2:

1. Book Management
2. Member Management
3. Loan Management

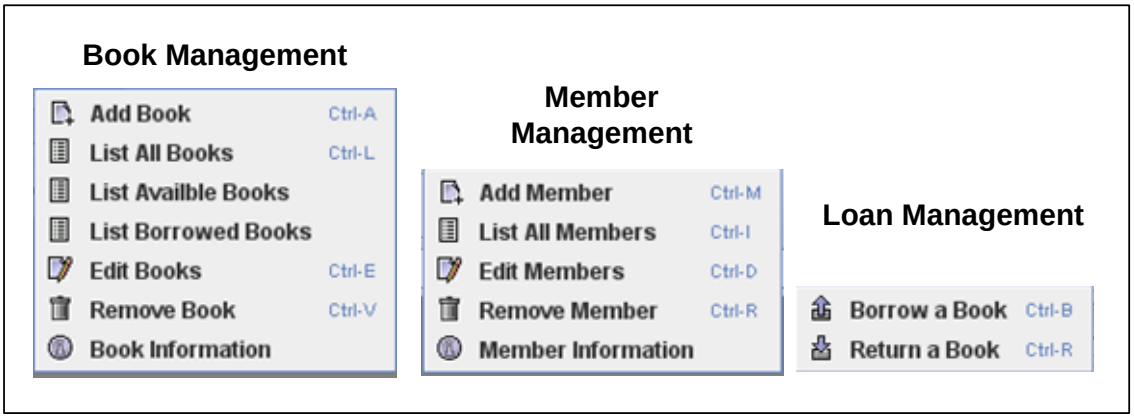

Figure 8-2: Functions in the Library Management System

Figure 8.1 shows some of the code metrics of the system after a static analysis on the source code using Metrics Eclipse Plug-in [155].

Metric	Total
Number of Classes	32
Number of Methods	102
Number of Attributes	373
Total Lines of Code	3275

Figure 8-3: Some Metrics of Library Management System Source Code

This library management system is not a complete library system, like that implemented in the real libraries, but it can still manage the basic loan functions for the library members using the MS Access database at the backend. In this case study, this library management system is nominated as an existing (legacy) library system, which is intended to be reengineered after the introduction to the library of RFID technology, as an example of ubiquitous technologies.

8.3 Tools Support

In this research, several tools were experimented with in order to implement the stages of the proposed reengineering framework on the selected case study. The following sections provide a brief description of the tools that were used in the case study in addition to the tools that were experimented with but not used. First, the next section, Section 8.3.1, presents the tools that were experimented with but not used. Then Section 8.3.2 provides a description of the tools that were used.

8.3.1 Tools Experimented but not Used

Several tools were tried in the process of locating the tools that can be used in the stages of the proposed framework. The tools presented in this section were installed configured and tried. However, as explained in the discussion section subsequent to the brief overview of the tools, some reasons hindered the use of these tools in this case study. Nevertheless, some of these tools can be put into use in the future work of this research.

8.3.1.1 PMD

PMD [10] is a free static code analyser for Java. The PMD home page lists a number of meanings for the acronym ‘PMD’. Some of the meanings given are ‘Pretty Much Done’, ‘Project Mess Detector’ and ‘Project Monitoring Directives’. As pointed out in the website, the acronym was chosen because the letters sound good together.

PMD is intended to be used by software developers in order to comply with coding standards and to deliver a quality code. PMD scans the Java source code for potential problems, such as possible suboptimal code, duplicate code, bugs, overcomplicated expressions and dead code. PMD checks for violations in three major areas: compliance with coding standards such as naming conventions, coding anti-patterns such as unused local variables, cut and paste such as suspect code replication.

PMD is available as an integrated plug-in for many IDEs, such as JDeveloper, Eclipse, JEdit and JBuilder. The PMD Plug-in for Eclipse presents the code checking rules into the Eclipse editor in an easy-to-use package. Even if PMD‘s automated scans are not used at build time, making it part of an IDE aids developers to write a cleaner code and to speed up code reviews.

Discussion

PMD is one of the tools experimented with among the Eclipse plug-ins within the code analysis category in the Eclipse plug-in web site. The tool was not used in this case study, because code refactoring is outwith the focus of the proposed reengineering framework. However, PMD can serve in future work of the framework. As a coder refactoring tool, PMD can be employed in the first stage of the framework which is the program understanding stage. To be more precise, the tool can be exploited in the business logic extraction step through introducing an additional step for code refactoring before the code refinement step. The version experimented with is PMD for Eclipse v.3.2.4.

8.3.1.2 Flowchart4J

Flowchart4j [40] is a commercial visualisation tool for the Java source code. The tool generates flowcharts and sequence diagrams for Java program methods. The generated flowchart and sequence diagrams help software engineers to understand code flow and the interaction between objects seamlessly. The tool can be used for documentation, conceptualisation and requirements specification.

Flowchart4j is also available as an Eclipse plug-in. The Flowchart4j Eclipse plug-in is an interactive tool which comprises features such as code-to-flowchart synchronisation, level folding, block expansion, highlighting complex conditions and exit points. Moreover, Flowchart4j comes with an Export feature to export the diagram to Microsoft Visio which makes it possible to edit the diagram. Developers can use this export feature to document the core functions of a Java program.

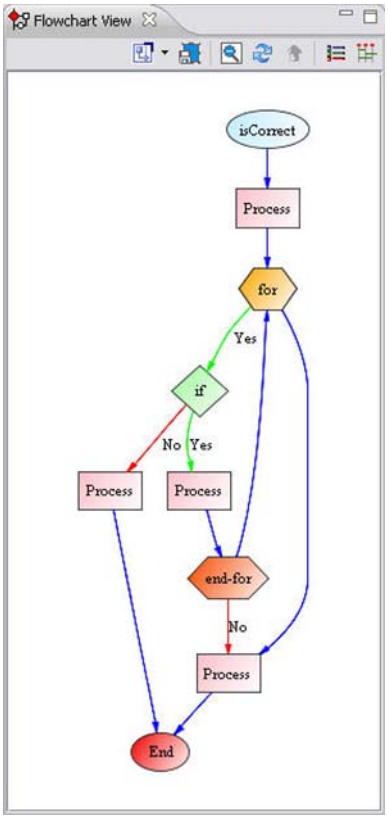

Figure 8-4: A Screen Shot of Flowchart4j Tool

Discussion

Since the Flowchart4j tool is a commercial tool, only the limited trial mode version was experimented with. Under trial mode, Flowchart4j generates flowcharts for functions with less than four conditional constructs, and sequence diagram message calls are limited to four calls.

The tool was not used in this case study because, as depicted in Figure 8-4, the

flowchart shows only the process as a black box without displaying what is going to be processed. The tool is useful for developers during the code development in order to check instantly the correctness of the logic implemented by looking at the flowchart. This is facilitated via clicking on the flowchart shapes to locate the line of code in the code editor. However, as a reverse engineering tool, the generated flowcharts do not provide sufficient information which can help in the overall program comprehension. The version experimented with is Flowchart4j for Eclipse v.2.0.1.

8.3.1.3 WebSphere Business Modeler Basic

The WebSphere Business Modeler Basic [80] provides a basic modeling, validation, and documentation tool to business users seeking to model, document, and print their business processes. WebSphere Business Modeler Basic offers analytical modeling of process flow, resources, costs, data, and performance management , as well as export to process design tools. More advanced simulation and analysis capabilities are available in the advanced version of the tool.

The tool is one of the four components of the IBM The WebSphere BPM suite. The three other components of the suite are WebSphere Integration Developer, WebSphere Process Server and WebSphere Business Monitor. The suite tools implement custom artifacts that leverage the infrastructure capabilities, monitor and manage the runtime implementations at both the IT and business process levels.

Discussion

The WebSphere Business Modeler Basic tool was used in experiments in the early stages of conducting the case study. The aim was to export the modelled business process model automatically to the UML activity diagram. A thirty-day trial version was used. With no previous experience with IBM products, the tool was relatively easy to master and use.

The tool was not used because it had been designed more for forward engineering rather than reverse engineering, and the research is focused more on the reverse

engineering tasks. Moreover, the feature of exporting the business process mode to UML activity diagram works mainly with IBM Rational software. Going through the process of learning these tools was beyond the interest of this research. Another reason for not using this tool was the limited time allowed to try the tool. The tool is not difficult to master but it is has a large umber of features and options that need to be understood as it is intended to be used by a group of developers, not as single user tool. The version experimented with is WebSphere Business Modeler Basic v.6.0.1.

8.3.1.4 Green UML

Green UML [170] , an ongoing project at the University of Buffalo, is a UML class diagram plug-in for the Eclipse IDE that includes features such as a customisable set of relationships and live round-tripping. Round-tripping allows both forward engineering and reverse engineering to be supported. Because it is a plug-in to Eclipse, Green UML can use the development environment to create a real-time synchronisation between the class diagrams and the Java source code. Green UML is capable of producing a UML class diagram using a code or generating a code by creating a class diagram.

Because it is an Eclipse plug-in, the editor provides an interface which is simple to use. For example, right-clicking on a class opens a context-sensitive menu that shows features such as the opening type hierarchy.

Discussion

The tool is a free full function utility which is not cumbersome to install and use. Green was experimented with to recover the class diagram of the case study system. However, in spite of the clarity of the generated class diagram from the code, exploring the diagram was difficult, especially with the large number of the class. The difficulty comes from the layout of the generated diagram which cannot be reorganised automatically. In order to make the diagram understandable and viewable, a time-consuming manual task required to be done. Furthermore, although the relationships between classes are labelled in the diagram, they cannot be highlighted. In addition, one has no control over the level of details that appear in the class diagram

after it is generated, other than showing or hiding the various types of relationships. The version experimented with is Green UML v.3.0.0. An example of a class diagram generated by the Green UML tool is shown in Figure 8-5.

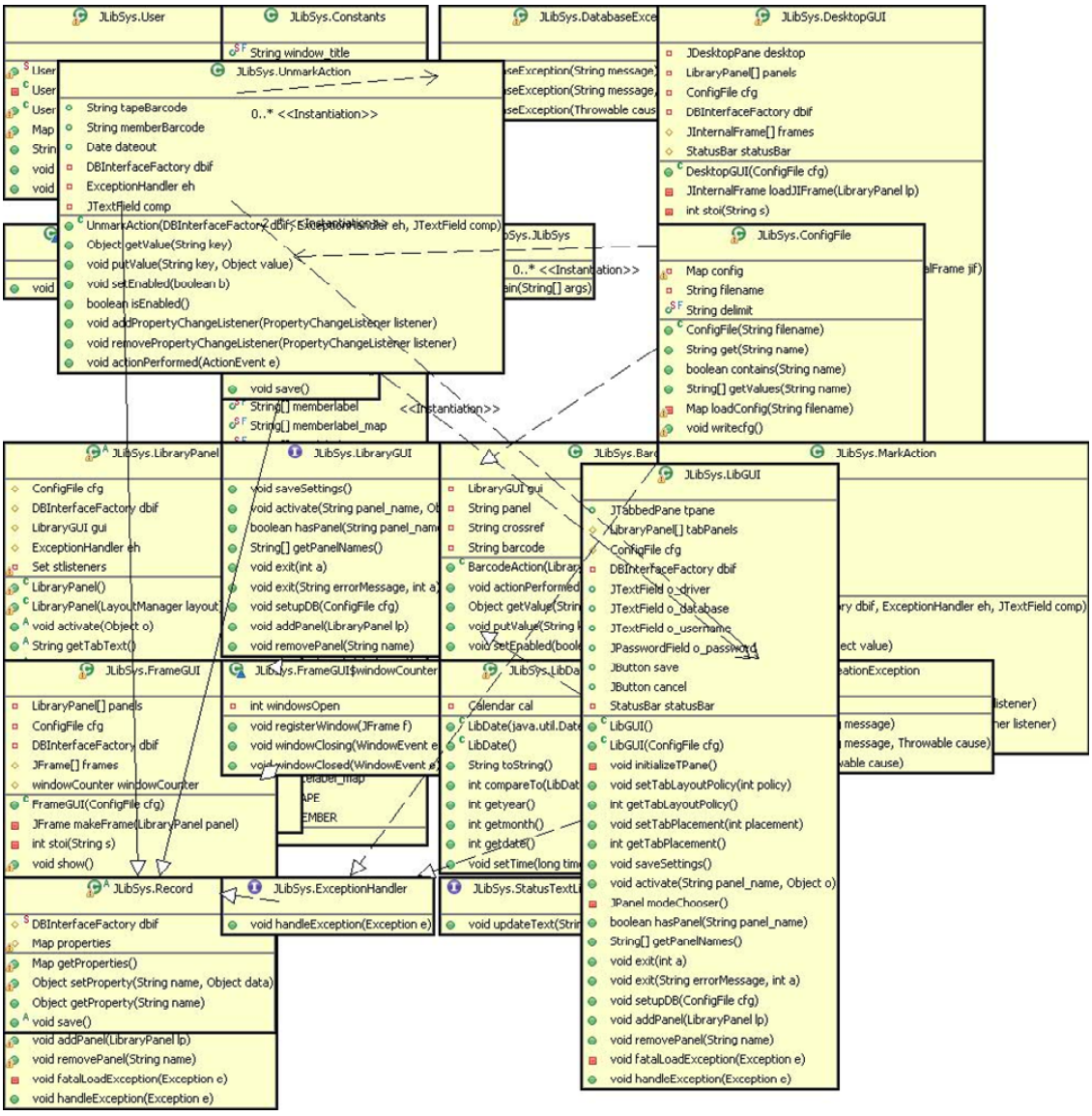

Figure 8-5: A Class Diagram Generated by Green UML

8.3.2 Tools Experimented and Used

The tools presented here are the tools that have been used in demonstrating the stages of the proposed framework. The rationale of choosing these tools is given in the discussion sections.

8.3.2.1 Eclipse

Eclipse Foundation [154] is an independent non-profit consortium of software industry vendors. The Eclipse community runs several open source projects which are focused on building an open development platform consisting of extensible frameworks, tools and runtimes for building, deploying and managing software. Eclipse open source projects started as a Java Integrated Development Environments (IDE), but has since grown to cover static and dynamic languages: thick-client, thin-client, and server-side frameworks. Eclipse also has tools for modelling and business reporting, embedded and mobile. In addition to Java, Eclipse can be used by means of the various plug-ins to develop applications in other languages as well such as C, C++, COBOL, Python, Perl, PHP and more.

Eclipse Java IDE is used to manage all of the software artefacts within a single project abstraction, and to coordinate all of the development activities from coding to deployment to debugging. The Eclipse Java Development Toolkit (JDT) includes a built-in incremental Java compiler and a full model of the Java source files. This allows for advanced code analysis and code refactoring techniques.

Discussion

In this case study, Eclipse IDE for Java Developers Version: 3.3.1.1 was used. The main focus of the proposed reengineering framework was on the reverse engineering tasks. For this reason, Eclipse was chosen because of the various free and commercial tools for code analysis available as Eclipse plug-ins. Also, as open source software, finding help, support and tutorials for using the IDE and plug-in was not an issue. An additional reason for using Eclipse is the hardware requirements. In spite of the huge

capabilities provided by the IDE, Eclipse runs acceptably on an average desktop computer.

8.3.2.2 Omondo EclipseUML

EclipseUML [122] allows software architects to model at a higher level of abstraction, and software developers to implement business logic inside the same tool. The tool offers live code and model synchronisation to keep the model and the code updated always. EclipseUML not only allows live UML and Java code synchronisation, but also provides model driven development to allow agile and efficient software development by delivering a good percentage of the required deployment code. The tool supports all thirteen UML 2.2 diagrams.

EclipseUML was designed primarily for Java developers. However, the tool can use either the Omondo Java code generator which has merge functionalities with existing Java code, or C, C++, PHP, C# code generator tool partners.

With regard to the reverse engineering capability, EclipseUML runs a complex processes for generating a class diagram from a source code. First it detects all information from the code and then maps it immediately to the UML editor and the UML Model. The tool works at three simultaneous levels and analyses all project byte codes, class files and Java files.

Discussion

In this case study, the free limited version, which includes comprehensive UML support, was used. The Studio version (non-free) adds the Java 2 Platform, Enterprise Edition (J2EE) support including visual deployment diagrams, visual assembly of jar files and database support. The free version was adequate for the work needed in this case study, which is the recovery of the class diagram of the library system.

The created class diagram was easy to navigate and customise. In the editor, it was possible to add or remove any details in the classes or in the relationships between the classes. Moreover, the layout of the class diagram can be manipulated automatically. In

a similar manner, the relationships can be highlighted, and they are differentiated by use of various colours. The diagram can be printed in multiple pages, and in the Studio version one page layout can be generated.

Unlike other tools, the synchronisation between code and diagram is very powerful. By clicking a method on the class diagram, the code of the method is highlighted in the source code. Moreover, any element in the diagram can be deleted either from the diagram only or from both the diagram and the code. The version used in this case study is EclipseUML v.3.3.0.

8.3.2.3 Visustin

Visustin [5] is a tool that visualises a source code with flow charts and UML activity diagrams. Visustin is a useful tool for software developers and software designers. Software developers use Visustin in software documentation through the automatic reverse engineering source code into flow charts or UML Activity Diagrams. Software designers also benefit from the tool through manually editing the created flowcharts or drawing new ones. Designers can insert additional comments, add shapes, highlight important points, adjust links and fine-tune the layout.

Flowcharts generated by Visustin can display the actual code or just the code comments. The tools support many programming languages, including for example ASP, Java, BASIC, C/C++, and COBOL. The generated flowcharts can be saved in various image formats and also can be exported to MS Visio for more manipulation. For large flowcharts, printing can be in multiple pages or squeezed to fit one page. In addition, the tool can export the generated diagrams to MS PowerPoint, MS Word and web pages. Visustin also calculates a few flow chart metrics, including lines of code, cyclomatic complexity and a number of links.

Discussion

The tool has two versions: standard and professional. In this case study the standard version is used. The professional version offers extra features such as drawing charts,

exporting to MS Visio and bulk flowchart creation. The tool is selected because of the support of the UML activity diagram. In the first stage of the framework, subsequent to the code refinement process, the tool is used to generate the UML activity diagram.

The support of a large number of languages and the flexibility offered by the editing feature gave advantage to Visustin over other code flowcharting tools such as Code Visual to Flowchart [53]. In addition, the updating of the flowchart and the switching between the conventional flow chart style and UML activity diagram was a matter of a mouse click. The version used in this case study is Visustin v.5.01.

8.3.2.4 Doxygen

Doxygen [72] is a documentation generator tool. The tool supports many programming languages including C++, C, Java, Objective-C, Python, and PHP and C #. 2. By configuring the tool the code structure can be extracted from source files that are either documented or undocumented. Undocumented source files help to understand large source distributions. Furthermore, the relations between elements can be visualised by using inheritance diagrams, dependency graphs and collaboration diagrams, all of which are generated automatically, whereas for documented source files, Doxygen can generate references to such documented classes, files, members and namespaces. In the same way global functions, global variables, type definitions, are documented and enumerations are also supported.

The Doxygen document can be presented in an on-line format (HTML) or an off-line format such as RTF, manual PostScript, UNIX man and hyperlinked PDF pages. As mentioned previously, because the documentation has been extracted from the source, it is therefore easier to maintain its consistency with the source code. Furthermore, the tool also has a rank-based search engine allowing for searches based on strings or words in the class and also member documentation.

Discussion

Doxygen has been used in this case study for recovering the class dependency

diagram. The recovery of the class dependency diagram is part of the architecture recovery step of the program understanding stage. Eclox [47] is a doxygen frontend plug-in for Eclipse is used in this case study. Eclox provides a high-level graphical user interface over doxygen which allows a simple integration of an additional code documentation process into Eclipse.

The documentation is generated in two formats: HTML and RTF. Although the class dependency diagram is used in this research, the other generated forms of diagrams can also assist in comprehending the reengineered system. The versions used in this case study are Doxygen v.1.5.6 and Eclox v. 0.8.0.

8.4 Program Understanding Stage

The first objective of the program understanding stage is to comprehend the functional aspect of the existing software via extracting the business logic embedded in the source code and representing it in a diagrammatical view. This step is demonstrated in the following section, Section 8.4.1. The second objective of this stage is to recover the structure of the software by recovering the class diagram from the source code of the system, as described in Section 8.4.2.

8.4.1 Business Logic Extraction

The aim of the business logic extraction step is to extract the business logic buried in the source code and represent it in BPMN, which is conceivable by both technical and non-technical professionals. The extraction process comprises three main steps for extracting the business logic embedded in the lines of codes in a software application. The three main steps are fully explained in Chapter 5, Section 5.2.2. The first step of the approach is grouping the code entities or classes into five groups: input, process, output, store, and supporting group. In the second step for each class in a certain group the business logic is extracted and represented at the end of the process in a BPMN diagram. This second step is further decomposed into more detailed sub-steps, as provided in Section 8.4.1.2. The third step of the approach is to integrate the modelled business

logic, if needed, to make it more readable and understandable for the non-technical professionals. In this case study, only the first and second steps are elaborated upon in the following sections.

8.4.1.1 Code Grouping

The purpose of this step is to group all the classes into the three groups: boundary (interface) classes, control classes, and entity classes. This grouping facilitates the extraction of the business logic from the system that is required to be reengineered through focusing on the source code entities that are part of the control classes group. The control classes group will hold the code entities that are most related to the core functions of the system. Code grouping is illustrated in detail in Chapter 5, Section 5.3.4.

The thirty two classes of the library management system had been grouped according to their purpose. A slicing tool Indus [130] which provides static slicing of Java programs, could be utilised in this step. Kaveri [130] is an Eclipse plug-in front-end for the Indus Java Slicer. The control classes group of the library management system's source code is depicted in Figure 8-6.

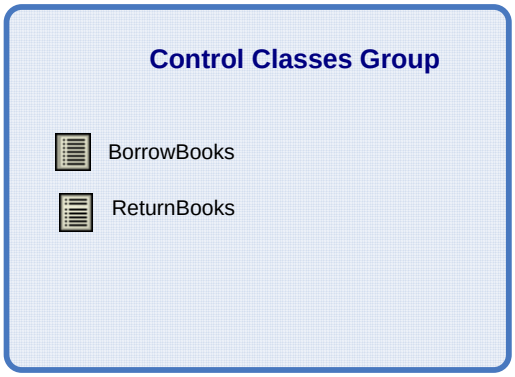

Figure 8-6: Library Management System's Control Classes Group

8.4.1.2 Extracting Business Logic in a Code Group

The second step of the process is to extract the business logic in each class which belongs to the control classes group and to represent the extracted business logic at the end of the process in a BPMN. In the proposed approach, a static-analysis-based method is used. The method is independent of the input and does not require the setup and execution of the system. This step is decomposed into four more internal steps which are:

1. Code parsing
2. Code refinement
3. UML activity diagram generation
4. BPMN diagram generation

In this case study, only the control classes group is considered for demonstrating the process of business logic extraction. The control classes group contains two classes:

- BorrowBooks class
- ReturnBooks class

Although the focus will be on the control classes group other groups may also be considered depending on the type of system. The determination of including an entity or a collection of classes from a group amongst the five groups is granted to the software engineer who has experience in the system domain.

8.4.1.3 Code Parsing

The purpose of the code parsing is to generate a code AST. The generated AST can be traversed by other tools to analyse the source code as a tree of nodes, in which each node represents a part of the source code. The purpose of traversing the generated AST is to facilitate the execution of the subsequent step of the process, which is the code

refinement. Code parsing is explained in Chapter 5, Section 5.2.5.1.

Figure 8-7: AST of BorrowBooks Class

Here, the Eclipse built-in parser is used to parse the code of the two classes: BorrowBooks and Returnbooks. The BorrowBooks AST and ReturnBooks shown respectively in Figure 8-7 and Figure 8-8 are viewed using the AST View Eclipse Plug-in [153]. The AST View plug-in provides a view to visualise the AST of a Java file open in the editor, and to navigate from text selection to AST nodes and from nodes to selections.

Figure 8-8: AST of ReturnBooks Class

8.4.1.4 Code Refinement

The aim of the code refinement step is to raise the abstraction level of the source code by filtering out non- business-logic entities based on the characteristics of the code and the programming language used. The generated AST of a source code will be traversed to analyse the source code as a tree of nodes, in which each node represents a part of the source code. The elimination of non-business related code is carried out, based on the abstraction algorithm used to determine the candidate code.

Once the AST of the source code is created by the parser, it can be traversed by a

visitor. In this case study, the AST generated can be analysed as a tree of nodes using the Java Document Object Model (JDOM) provided by Eclipse Development Tools (JDT). The objective of traversing the created AST is to apply the abstraction algorithm presented in Figure 5-7, Chapter 5, Section 5.2.5.2.

Listing 8-1 and Listing 8-2 show a part of the BorrowBooks class and the ReturnBooks class, before applying the abstraction algorithm. The code of the complete classes are shown in Appendix A.

```
//import the packages for using the classes in them into the program
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.SimpleDateFormat;
import java.util.Locale;
public class BorrowBooks extends JInternalFrame {

	// for creating the North Panel
	private JPanel northPanel = new JPanel();
	// for creating the label
	private JLabel title = new JLabel("BOOK INFORMATION");

	// for creating the Center Panel
	private JPanel centerPanel = new JPanel();
	// for creating an Internal Panel in the center panel
	private JPanel informationPanel = new JPanel();
	// for creating an array of JLabel
	private JLabel[] informationLabel = new JLabel[4];
	// for creating an array of String
	private String[] informationString = { " Write the Book ID:",
			" Write the Member ID:", " The Current Data:", "
The Return Date:" };
	// for creating an array of JTextField
	private JTextField[] informationTextField = new JTextField[4];
	// for creating the date in the String
	private String date = new SimpleDateFormat("dd-MM-yy",
```

```
Locale.getDefault())
                    .format(new java.util.Date());
      // for creating an array of string to store the data
      private String[] data;

      // for creating an Internal Panel in the center panel
      private JPanel borrowButtonPanel = new JPanel();
      // for creating the button
      private JButton borrowButton = new JButton("Borrow");

      // for creating South Panel
      private JPanel southPanel = new JPanel();
      // for creating the button
      private JButton cancelButton = new JButton("Cancel");

      // for creating an object
```

Listing 8-1: Part of the BorrowBooks Class before Refinement

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ReturnBooks extends JInternalFrame {

      //for creating the North Panel
      private JPanel northPanel = new JPanel();
      //for creating the label
      private JLabel title = new JLabel("BOOK INFORMATION");
      //for creating the Center Panel
      private JPanel centerPanel = new JPanel();
      //for creating an Internal Panel in the center panel
      private JPanel informationPanel = new JPanel();
      //for creating an array of JLabel
      private JLabel[] informationLabel = new JLabel[2];
      //for creating an array of String
```

```
	private String[] informationString = {" Write the Book ID:", "
Write the Member ID:"};
	//for creating an array of JTextField
	private JTextField[] informationTextField = new JTextField[2];
	//for creating an array of string to store the data
	private String[] data;

	//for creating an Internal Panel in the center panel
	private JPanel returnButtonPanel = new JPanel();
	//for creating the buton
	private JButton returnButton = new JButton("Return");
	//for creating the panel
	private JPanel southPanel = new JPanel();
	//for creating the button
	private JButton cancelButton = new JButton("Cancel");

	//for creating an object
	private Books book;
	private Members member;
	private Borrow borrow;

	//for checking the information from the text field
	public boolean isCorrect() {
		data = new String[2];
		for (int i = 0; i < informationLabel.length; i++) {
			if
(!informationTextField[i].getText().equals(""))
				data[i] =
informationTextField[i].getText();
			else
				return false;
		}
		return true;
	}
```

Listing 8-2: Part of the ReturnBooks Class before Refinement

Listing 8-3 and Listing 8-4 show the BorrowBooks class and the ReturnBooks class,

after applying the abstraction algorithm.

```
public class BorrowBooks extends JInternalFrame {

            borrowButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                  // for checking if there is a missing information
                  if (isCorrect()) {
                        Thread runner = new Thread() {
                              @Override
                              public void run() {
                                    book = new Books();
                                    member = new Members();
                                    borrow = new Borrow();
                                    book
                                    .connection("SELECT * FROM
Books WHERE BookID = "
                                                + data[0]);
                                    member
                                    .connection("SELECT * FROM
Members WHERE MemberID = "
                                                + data[1]);
                                    int numberOfAvailbleBooks =
book
                                    .getNumberOfAvailbleBooks();
                                    int numberOfBorrowedBooks = 1
+ book
                                    .getNumberOfBorrowedBooks();
                                    int numberOfBooks = 1 +
member.getNumberOfBooks();
   // for checking if there is no same information in
                                          // the database
  if (numberOfAvailbleBooks == 1) {
                                                numberOfAvailbleBooks
-= 1;
                                                book
                                                .update("UPDATE Books
SET NumberOfAvailbleBooks ="
```

```
                                                                    +
numberOfAvailbleBooks
                                                                    +
",NumberOfBorrowedBooks ="
                                                                    +
numberOfBorrowedBooks
                                                                    +
",Availble = false WHERE BookID ="
                                                                    +
data[0]);
                                                        member
                                                        .update("UPDATE
Members SET NumberOfBooks = "
                                                                    +
numberOfBooks
                                                                    + " WHERE
MemberID = "
                                                                    +
data[1]);
                                                        borrow
                                                        .update("INSERT INTO
Borrow (BookID, MemberID, DayOfBorrowed, DayOfReturn) VALUES ("
                                                         + data[0]
                                                                        +
","
                                                                        +
data[1]

+ ",'"
+ data[2]
+ "','"
+ data[3] + "')");

                                                } else if
(numberOfAvailbleBooks > 1) {
                                                        numberOfAvailbleBooks
-= 1;
                                                        book
```

```
                                                .update("UPDATE Books
SET NumberOfAvailbleBooks ="
                                                                    +
numberOfAvailbleBooks
                                                                    +
",NumberOfBorrowedBooks ="
                                                                    +
numberOfBorrowedBooks
                          WHERE BookID =" + data[0]);
                                                member
                                                .update("UPDATE
Members SET NumberOfBooks ="
                                                                    +
numberOfBooks
                                    + " WHERE MemberID =" + data[1]);
                                                borrow
                                                .update("INSERT INTO
Borrow (BookID, MemberID, DayOfBorrowed, DayOfReturn) VALUES ("
                                    + data[0]
                                                                        +
","
                                                                        +
data[1]

+ ",'"

+ data[2]

+ "','"

+ data[3] + "')");

                                            } else {

JOptionPane.showMessageDialog(null,
                                                                    "The
book is borrowed", "Warning",
```

```
JOptionPane.WARNING_MESSAGE);
                                        }
                                    }
                                };
                        }
  // if there is a missing data, then display Message Dialog
                        else {
                                JOptionPane.showMessageDialog(null,
    "Please, complete the information", "Warning",

JOptionPane.WARNING_MESSAGE);
                        }
                }
        });}
```

Listing 8-3: BorrowBooks Class after Refinement

```
public class ReturnBooks extends JInternalFrame {

                returnButton.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent ae) {
                        // for checking if there is a missing information
                        if (isCorrect()) {
                                Thread runner = new Thread() {
                                        public void run() {
                                                book = new Books();
                                                member = new Members();
                                                borrow = new Borrow();
                                                book.connection("SELECT *
FROM Books WHERE BookID = " + data[0]);
                                                member.connection("SELECT *
FROM Members WHERE MemberID = " + data[1]);
                                                int numberOfAvailbleBooks =
book.getNumberOfAvailbleBooks();
                                                int numberOfBorrowedBooks =
book.getNumberOfBorrowedBooks() - 1;
                                                int numberOfBooks =
member.getNumberOfBooks();
```

```
                                        // for checking if there is no
same information in
                                        // the database
                                        if (numberOfAvailbleBooks ==
0 && numberOfBooks > 0) {
                                                numberOfAvailbleBooks
+= 1;
                                                numberOfBooks -= 1;
                                                book.update("UPDATE
Books SET NumberOfAvailbleBooks =" + numberOfAvailbleBooks +
",NumberOfBorrowedBooks =" + numberOfBorrowedBooks + ",Availble = true
WHERE BookID =" + data[0]);
                                                member.update("UPDATE
Members SET NumberOfBooks =" + numberOfBooks + " WHERE MemberID =" +
data[1]);
                                                borrow.update("DELETE
FROM Borrow WHERE BookID =" + data[0] + " AND MemberID =" + data[1]);
                                                //for setting the array
of JTextField to null

                                        }
                                        else if
(numberOfAvailbleBooks > 0 && numberOfBooks > 0) {
                                                numberOfAvailbleBooks
+= 1;
                                                numberOfBooks -= 1;
                                                book.update("UPDATE
Books SET NumberOfAvailbleBooks =" + numberOfAvailbleBooks +
",NumberOfBorrowedBooks =" + numberOfBorrowedBooks + " WHERE BookID ="
+ data[0]);
                                                member.update("UPDATE
Members SET NumberOfBooks =" + numberOfBooks + " WHERE MemberID =" +
data[1]);
                                                borrow.update("DELETE
FROM Borrow WHERE BookID =" + data[0] + " AND MemberID =" + data[1]);
                                                //for setting the array
```

```
of JTextField to null

                                        }
                                        else

JOptionPane.showMessageDialog(null, "The book is not borrowed",
"Warning", JOptionPane.WARNING_MESSAGE);
                                    }
                            };
                    }
                    // if there is a missing data, then display Message
Dialog
                    else
                            JOptionPane.showMessageDialog(null,
"Please, complete the information", "Warning",
JOptionPane.WARNING_MESSAGE);
            }
      });
}
```

Listing 8-4: ReturnBooks Class after Refinement

8.4.1.5 UML Activity Diagram Generation

This step is executed after the abstraction of the source code entities achieved by the preceding step, the code refinement. In this step, the UML activity diagram is generated automatically using available tools. Since the non-business logic code has been filtered out using the abstraction algorithm, the activity diagram created will represent the flow of the business logic embedded in the source code.

In this case study, the abstracted versions of the two classes from the control classes group, BorrowBooks and ReturnBooks, have been used. The Visustin tool was used to generate the UML activity diagram automatically for the source code of the two classes, as depicted in Figure 8-9 and Figure 8-10. The diagrams can be simplified further using the same tool in many ways, such as shortening the code, compressing continuous blocks, truncating long lines and producing a birdseye view of the diagram. The activity

diagram generated in this step follows the basic standard notation of the UML activity diagram, as described in Chapter 3, Section 3.2. Since the diagrams extracted from the source in this step are going to be simplified further using BPMN, activity diagram advanced notation elements are not used.

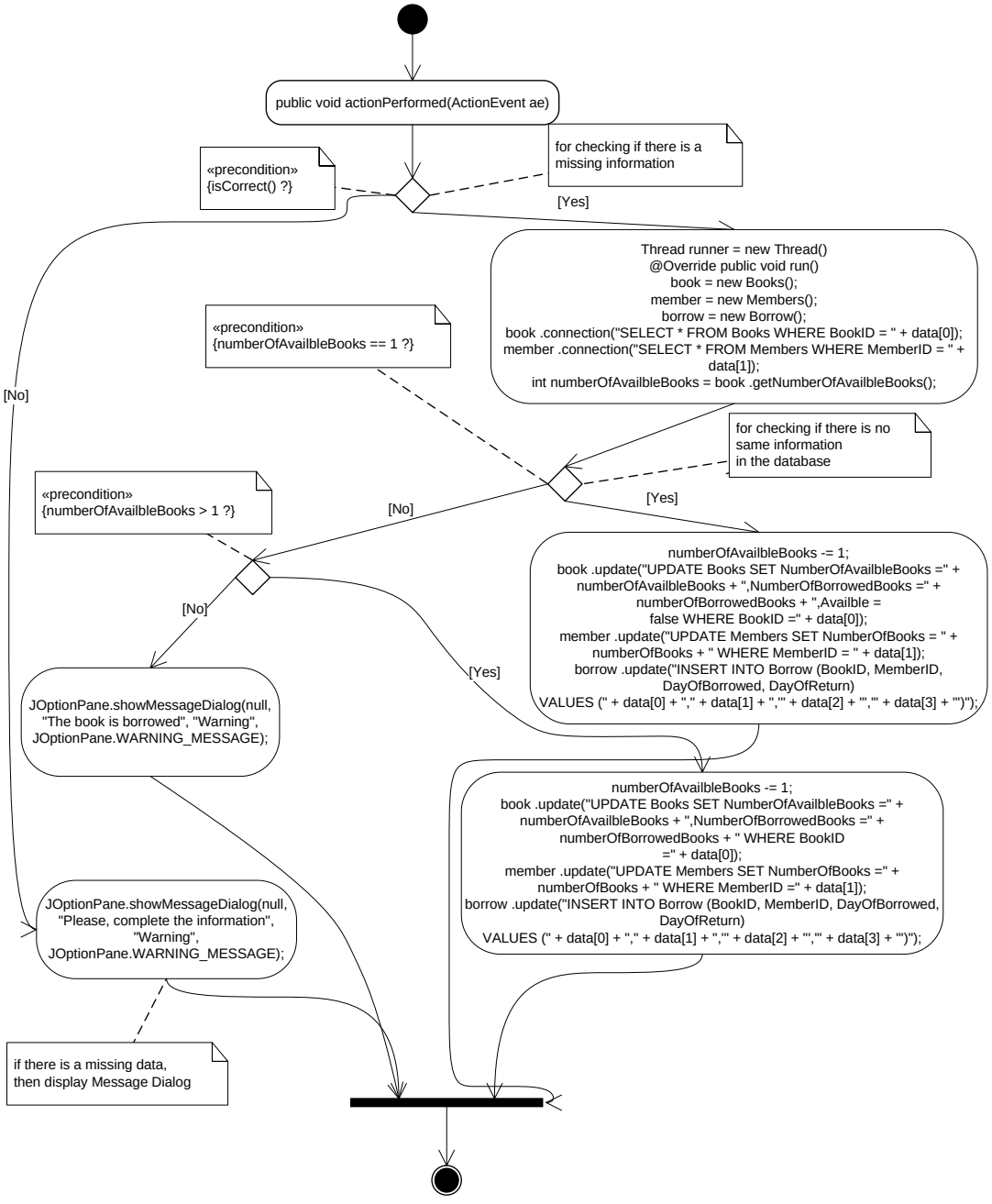

Figure 8-9: Abstracted BorrowBooks Class in UML Activity Diagram

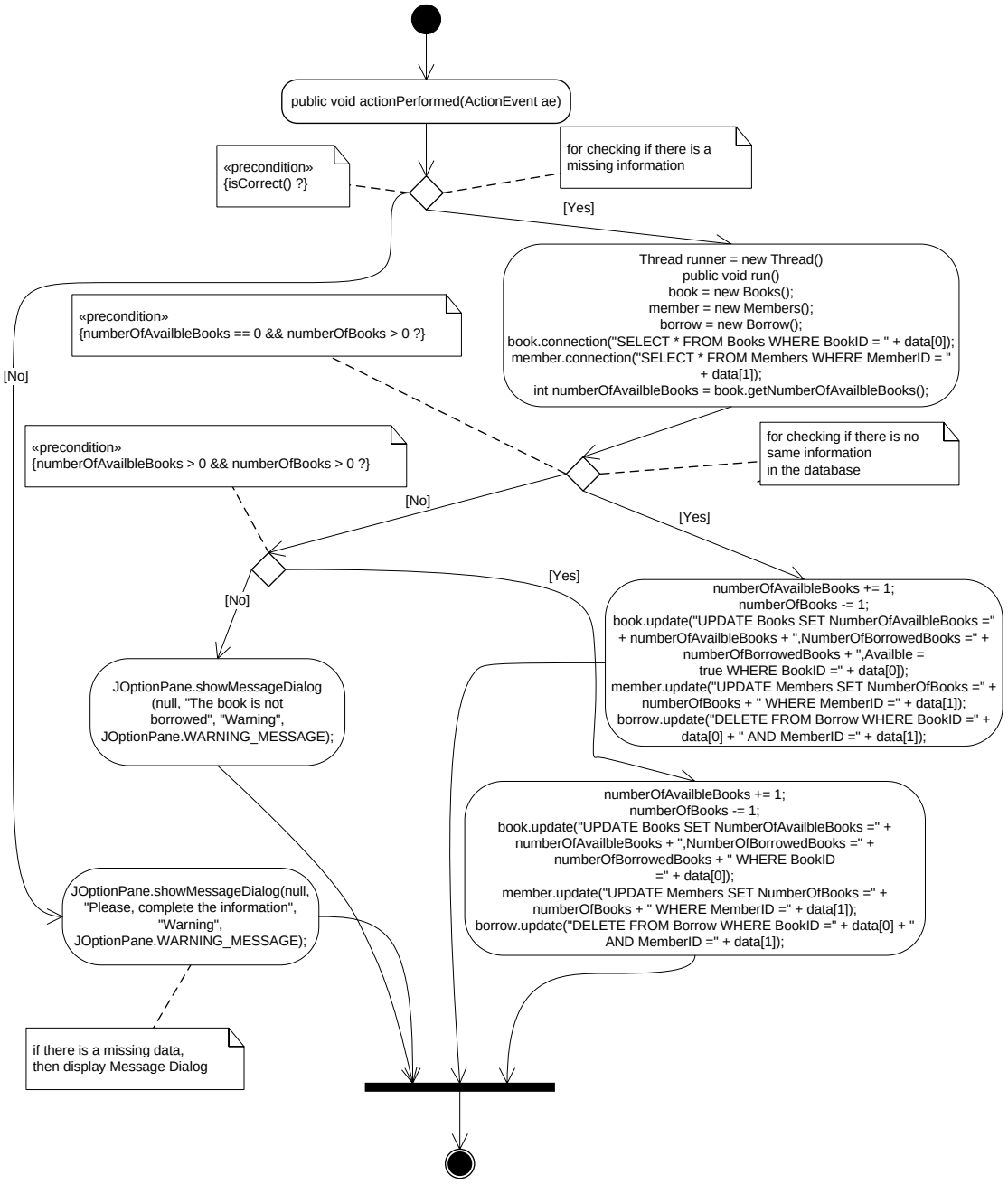

Figure 8-10: Abstracted ReturnBooks Class in UML Activity Diagram

8.4.1.6 BPMN Diagram Creation

The proposed business logic approach developed in this research aims to represent the extracted business logic in an understandable manner for both technical and non technical professionals. Since the BPMN has been adopted and is currently widely used

by both professionals, in this last step of the process of extracting the business logic, the generated activity diagram is transformed into BPMN.

The last three steps of the process as described above are carried out without the need of human intervention. Code parsing, AST generation and traversal, code refinement, and lastly UML activity diagram generation are achieved using appropriate tools. However, in this step, software engineers with adequate knowledge in the domain of the application and the programming language used are needed. The software engineer's job in this step is to simplify the extracted activity diagram by replacing the code inside the diagram with more human comprehensible terms using BPMN.

IBM has demonstrated in [64] how to transform UML activity diagrams automatically into WebSphere Business Modeler processes. However, in this case study, the business logic extraction process was performed manually. The decision to carry out this step manually was because of the limitation of the method presented by IBM to its software development platform. Moreover, although the tool claimed to be capable of automatically transforming UML activity diagrams into a more business-oriented fashion notation, human intervention is still required to complete the final transformation.

Figure 8-11 and Figure 8-12 show the BPMN diagrams generated for the previously created UML activity diagrams shown in Section 8.4.1.5. These BPMN diagrams will be used later in the integration stage.

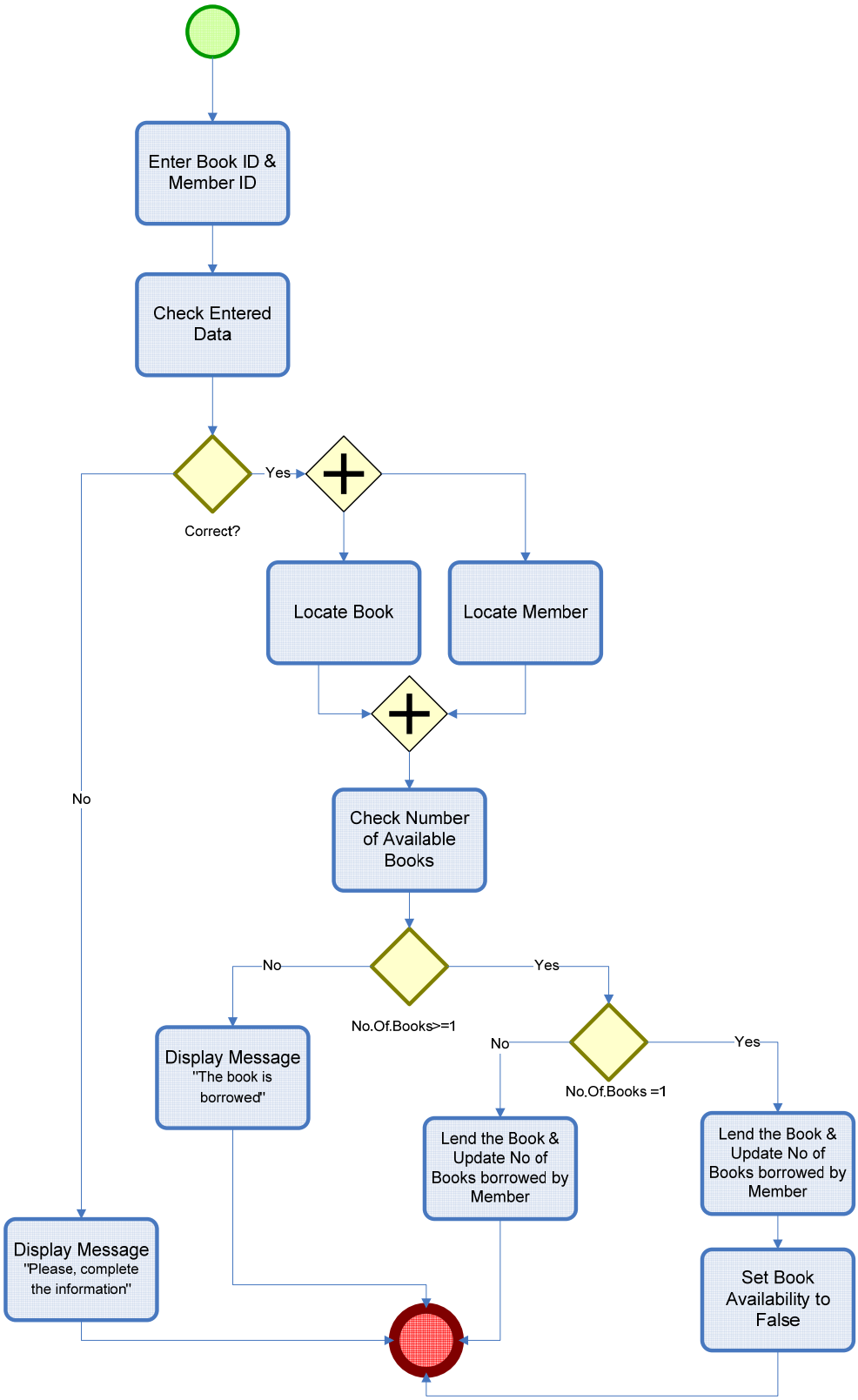

Figure 8-11: Business Logic Extracted from the BorrowBooks Class in BPMN.

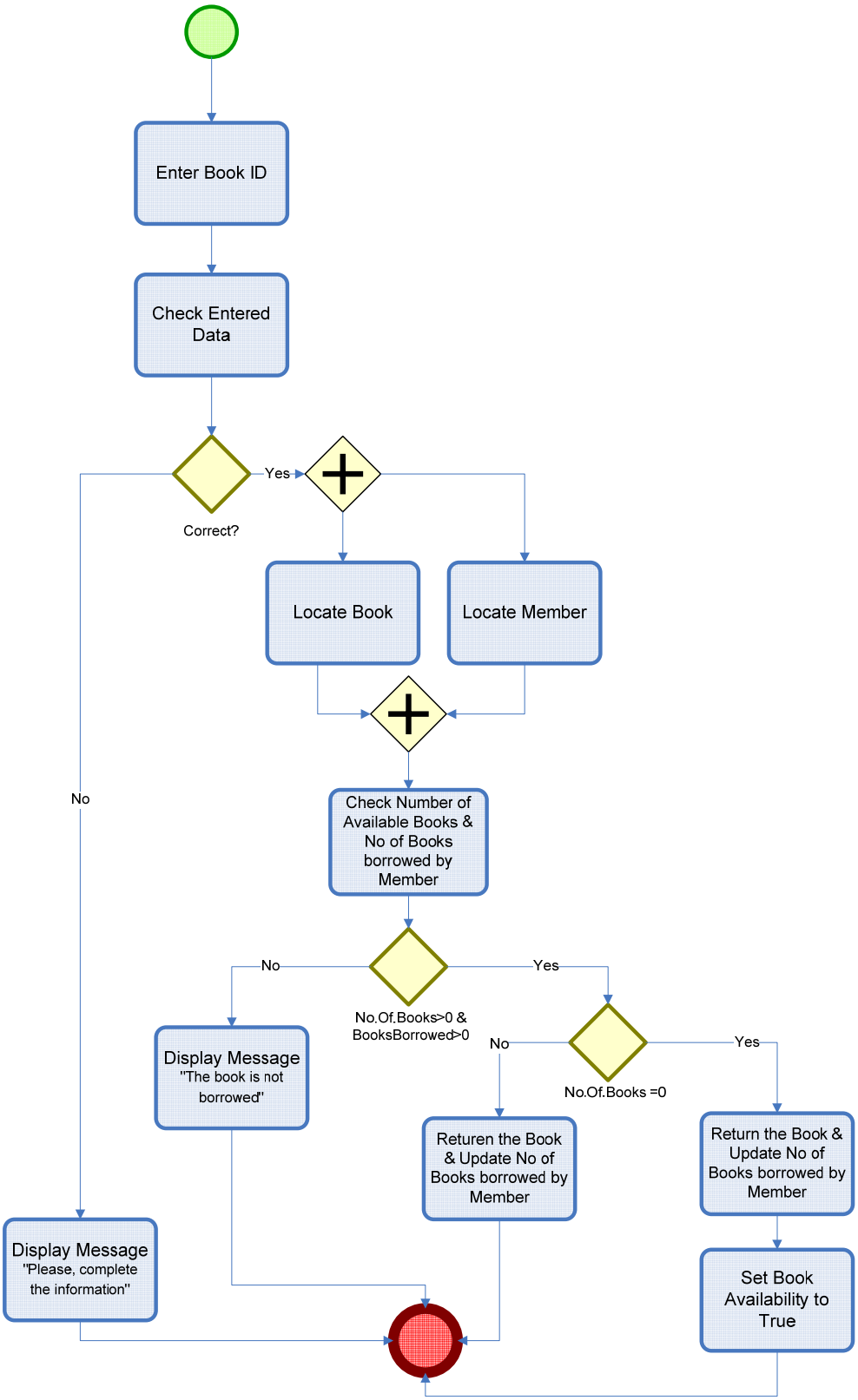

Figure 8-12: Business Logic Extracted from the ReturnBooks Class in BPMN

8.4.2 Architecture Recovery

In this step of the program understanding stage, the UML diagrams was extracted from the source code of the library management system to present the static structure of the system along with class dependency diagram to illustrate the relationships between classes.

8.4.2.1 Class Diagram Recovery

Figure 8-13 shows the class diagram of the existing library management system. The diagram views the static structure of the system. The reverse engineered class diagram using Omondo EclipseUML assists in understanding the system's organisation in general. Information about Omondo EclipseUML is given in Section 8.3.2.2. Moreover, it helps in identifying the kind of interclass connections that exist in the system.

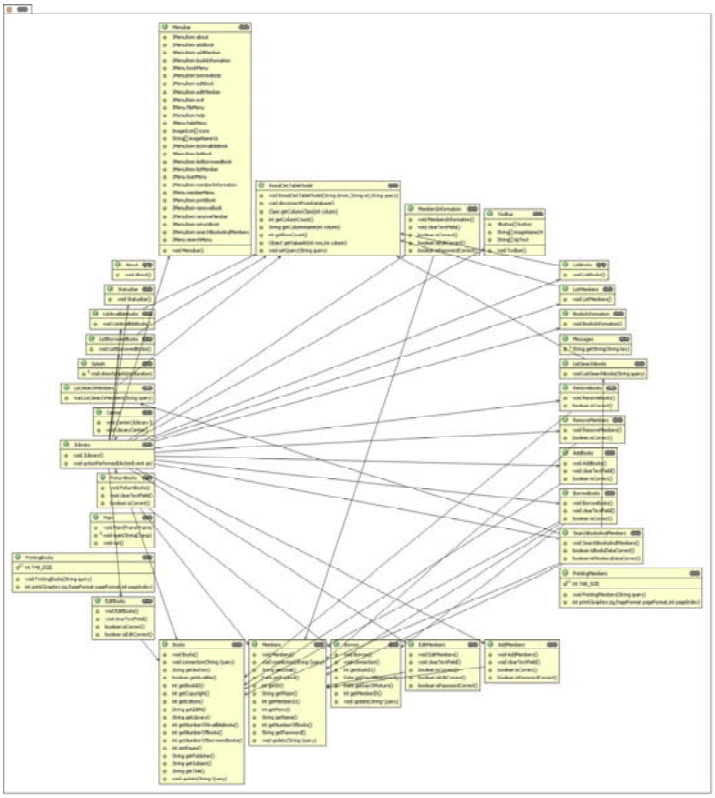

Figure 8-13: Class Diagram of the Existing Library Management System

8.4.2.2 Class Dependency Diagram Recovery

Figure 8-14 and Figure 8-15 depict examples of class dependency diagrams generated from the source code of the existing library management system. The Doxygen documentation tool is used to create these class dependency diagrams. Information about Doxygen is provided in Section 8.3.2.4. The class dependency diagram illustrates the relationship and interaction between the system's classes. The statically reverse engineered class dependency diagram helps in understanding the relationships between classes implementing a software system.

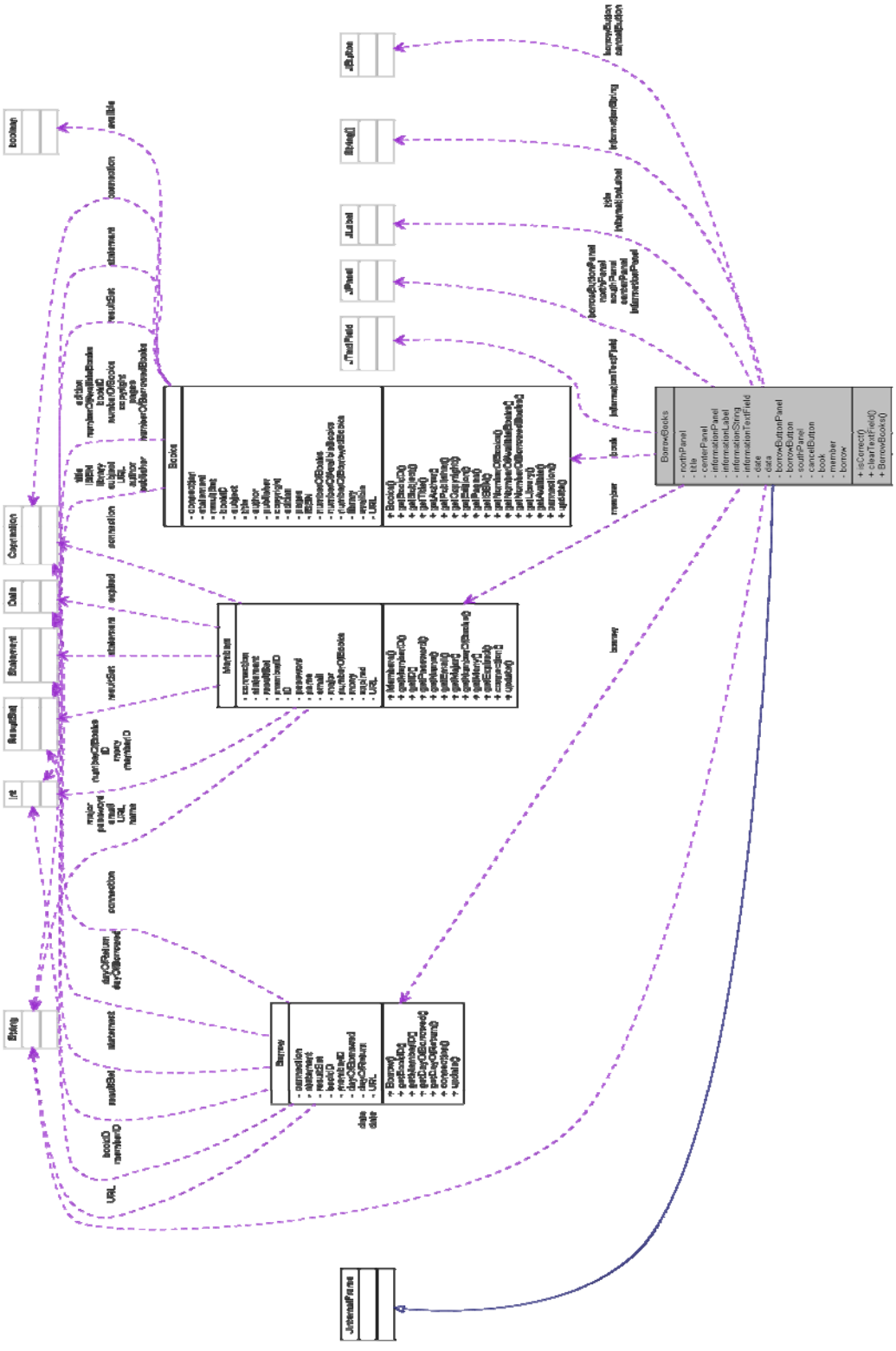

Figure 8-14: Class Dependency Diagram for BorrowBooks Class

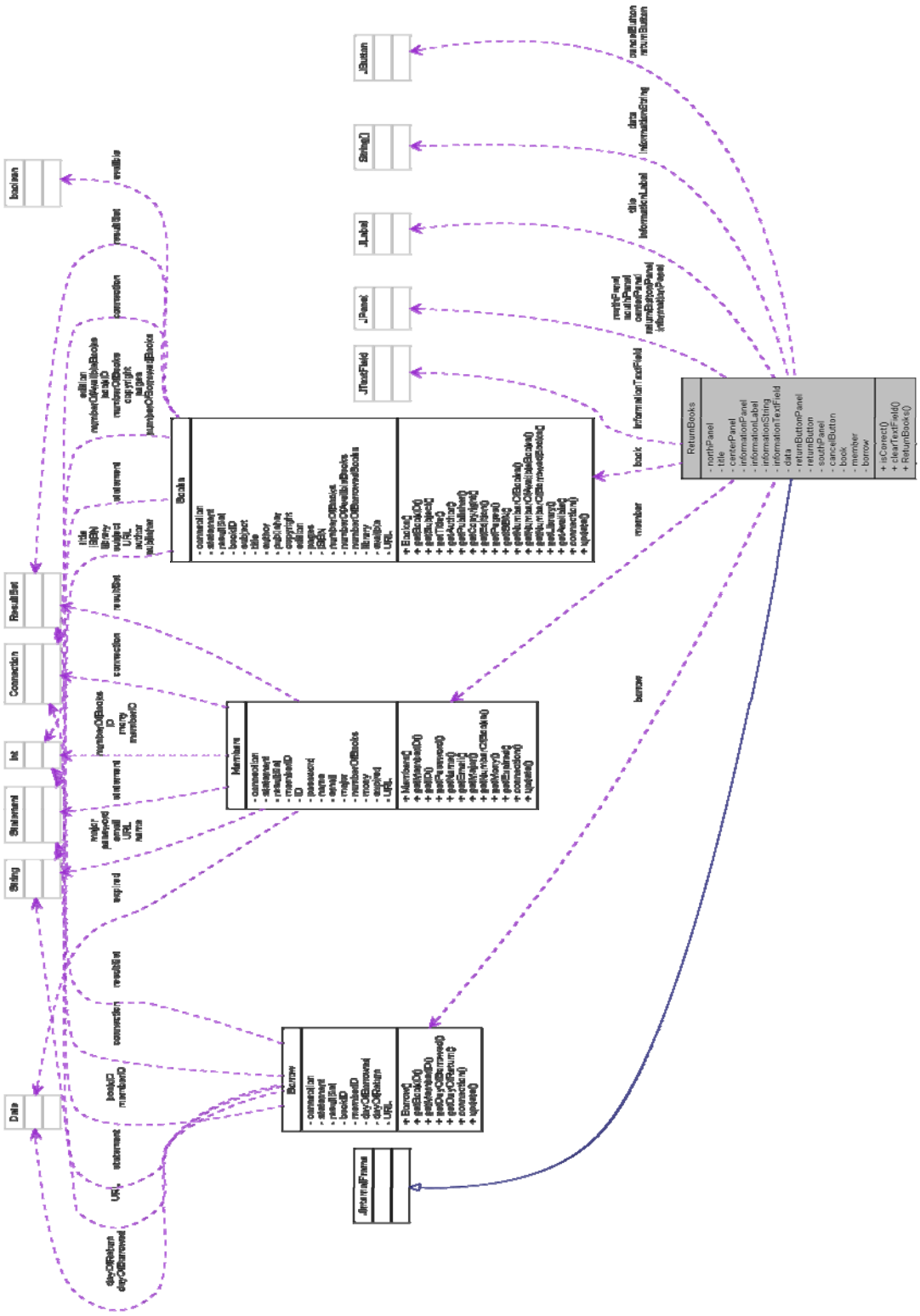

Figure 8-15: Class Dependency Diagram for ReturnBooks Class

8.5 Additional-Requirements Engineering Stage

As discussed earlier in Chapter 6, in this stage only the requirements elicitation process is covered in this research. The elicitation process followed in the proposed framework comprises three steps: scenario description, scenario modelling and pseudo code development. The possible scenarios which may occur after the introduction of a ubiquitous technology are described in plain text, and then analysed using the Carroll [33] analysis method explained in Chapter 6, Section 6.3.1. Then BPMN is used to model the function resulting from the analysis of possible scenarios as consequences of the introduction of the ubiquitous technology. Finally, pseudo codes of the new functions are provided.

8.5.1 Background

RFID is used in this case study as an example of a ubiquitous technology that has been introduced to a library. A typical RFID system used in libraries consists of smart RFID labels, hardware and software, and provides libraries with a more effective method of managing their inventory and at the same time providing better customer service to their patrons. In general, RFID allows libraries to adopt the technology with the following advantages:

- Rapid item check-in and checkout.
- Simplified patron self check-in and checkout.
- High reliability and security.
- High-speed conduct of inventorying.
- Automated material handling.
- Fast track circulation operation

In a typical implementation of RFID in a library, the RFID tags are attached to the

library's collection of books, CDs, video tapes, etc. In this case study, the tags are provided for books in addition to the library's patrons (members). The usual locations of RFID readers inside the library are normally in a self check-out station, a staff check-out station, a self-return book drop, a tagging station, security gates, a shelf scanner and an administrative station. In this case study, the readers are also placed near sitting and study areas. In addition, additional readers are positioned in the middle of the aisles which have the shelves holding books related to a specific subject.

8.5.2 Scenarios

Considering the above physical layout of the RFID system inside the library premises, a number of usage scenarios can be assumed. In this case study three usage scenarios were generated. The analysis of these three scenarios will result in three services that can be provided by the existing library system. The three scenarios are given in plain text, modelled using BPMN, analysed using the Carroll's method (see Chapter 6, Section 6.3.1.2) and then described in pseudo code. The modelled scenarios in BPMN were used in the next stage of the framework which is the integration stage. The pseudo codes are used in the actual implementation of the new or modified functions of the system.

8.5.2.1 Scenario One

Plain Text

The RFID reader located in the aisles of the book shelves is capable of tracking the different locations in which a library patron may have spent some time. A timeframe can be specified to define an appropriate period of time spent by a patron in a certain location by which the location is considered to have been truly visited by a patron. From the location, the subject or area of the books can be recognised. Another formula can be devised to match the location information accurately with the subjects of books stored in the location shelves. Based on the result of this formula, the patron's profile is updated by adding the subject as one of the patron's interests. This addition of subject interest is used later in deciding the promotion of books or services.

This scenario is the enabler of the two scenarios described in the next sections, Section 8.5.2.2 and Section 8.5.2.3.

Analysis

For the above scenario, the scenario elements based on Carroll method are:

- **Setting**: is a library aisle. The scenario implies further setting elements by identifying the person as a patron, and the aisles are equipped with an RFID reader.
- **Actors**: the patron is the only actor in this scenario.
- **Actions**: spending some time near a bookshelf browsing the available book titles or skimming through a book.
- **Events**: detecting the location, determining the books' subjects, and updating the patron's profile.
- **Goal**: updating the patron's profile with subjects of interests.

Model

For the above scenario in use case tabular form is presented in Figure 8-16, the scenario model in BPMN is depicted in Figure 8-17.

Use case No.*x*	Recording Patron's Location
Precondition	(none)
Trigger	Patron(s) is pending some time in a location
Post-condition	Patron's location is recorded
Main Scenario	1. Read reader ID 2. Determine Location 3. Read Patron's Tag. 4. Retrieve patron's profile. 5. Update Patron's profile accordingly.
Extensions	(none)

Figure 8-16: Recording Patron's Location in Use Case Tabular Form

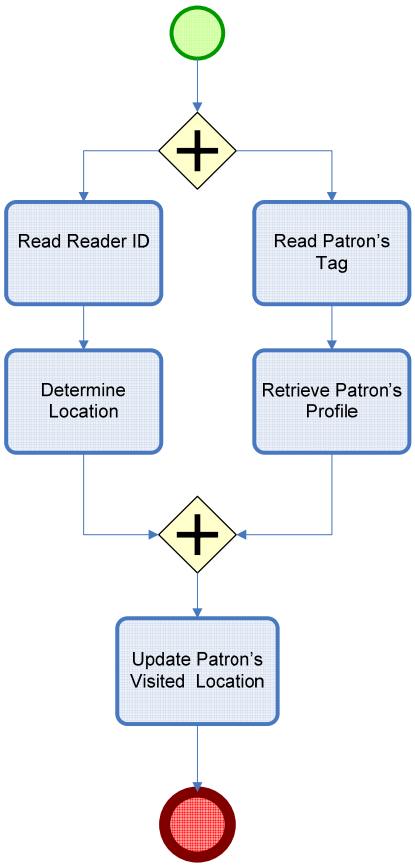

Figure 8-17: Scenario of Recording Patron's Location in BPMN

Pseudo Code

For the above scenario, the pseudo code for the recording of the patron's locations is illustrated in Listing 8-5.

```
public void recordLocation(String readerIp,List tags)
{
	//get an iterator to iterate the tags list
	tagsIterator=tagList.iterator();
	//while there's still tags in the list
	while(tagsIterator.hasNext())
	{
		// get the tag on turn
		patronTag=(Tag)tagsIterator.next();
		//get the patron profile using his tag id
		patron=this.getPatronByTagId(patronTag.getId());
		//update patron's visted locations
		patron.updateVistedLocations(readerIp);
	}

}
```

Listing 8-5: Recording Patron's Location Pseudo Code

8.5.2.2 Scenario Two

Plain Text

In the existing system, the borrowing process of a book runs as follows. The patron walks into the library, searches for the book he/she wants, and then takes it to the checkout station. A library staff keys in the patron's ID and the book's ID. The reason for checking the patron's ID is to verify his information and to check the number of books he/she is allowed to borrow. The purpose of checking the book's ID is to find out if the book has no restrictions for borrowing by a patron. If the result of these checks is positive, the book is lent. Otherwise the patron is notified with the appropriate information.
With the introduction of RFID and the services offered by this introduction, the borrowing process logic is different from the existing one. In the new logic, after performing the same checks and when the patron is ready to borrow the book, the patron is offered help, based on his profile. The help can be by offering a service provided by the library or by promoting a new book or a list of book based on the patron's profile. A patron's profile contains the patron's subjects of interest accumulated by direct questioning and by the tracking service presented by the RFID utilisation.

Analysis

For the above scenario, the scenario elements based on Carroll method are:

- **Setting**: is a library checkout station. The scenario implies further setting elements by identifying the people as a patron and a librarian. The library uses an RFID system in its operations.
- **Actors**: the patron and the librarian are the actors in this example.
- **Actions**: verifying the patron and the book information are actions that facilitate the goal of borrowing a book and promoting services or books.
- **Events**: retrieving the information about the patron and the book, retrieving the patron's profile, and listing available services or related books.
- **Goal**: borrowing a book and providing help to the patron.

Model

For the above scenario, the scenario in use case tabular form is presented in Figure 8-18 and the scenario model in BPMN is depicted in Figure 8-19.

Use case No.*y*	Borrowing a book after RFID
Precondition	(none)
Trigger	Patron(s) want to bowwro a book from the library
Post-condition	Patron's request accepted or rejected. Patron's profile is updated accordingly.
Main Scenario	1. Read Patron's Tag 2. Read's Book's Tag 3. Check number of available books. 4. If available lend the book and update quantity. 5. Check patron's profile 6. If not new promote suggest book or service based on patron's interest. 7. Otherwise suggest a generic service. 8. Update patron's profile accordingly.
Extensions	(none)

Figure 8-18: Borrowing a Book after RFID in Use Case Tabular Form

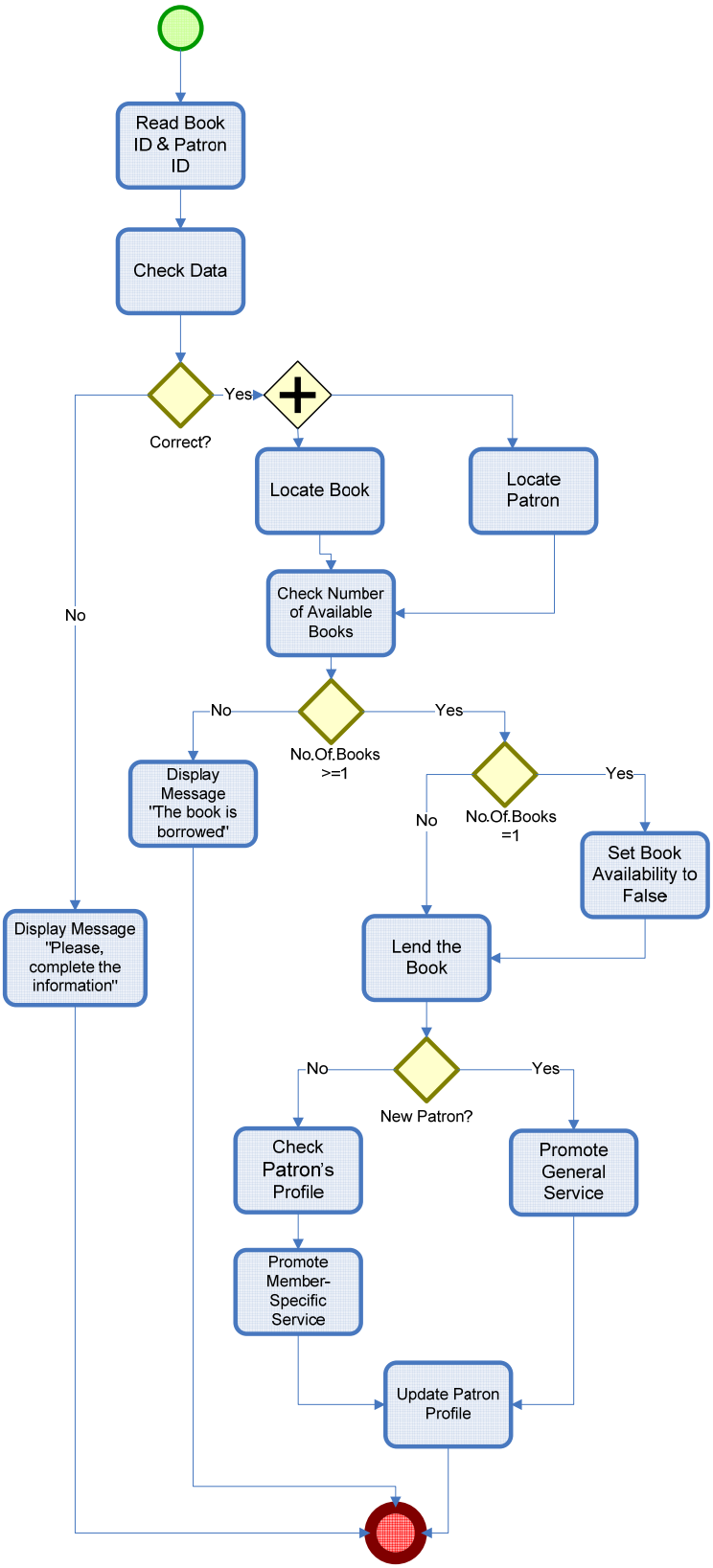

Figure 8-19: Scenario of Borrowing a Book after RFID in BPMN

Pseudo Code

For the above scenario, the pseudo code for the recording of borrowing book process after RFID is illustrated in Listing 8-6

```
  public void borrowBook(String bookId)
  {
        //read the memeber tag to retrieve his profile
        memberTag=readTag();
        //get the patron profile using his tag id
        member=this.getPatronByTagId(memberTag.getId());
        //get the book data
        book=this.getBookByBookId(bookId)
        //update the book availability and member profile
        member.borrow(book,currentDate,estimatedReturnDate);
        //mine the memeber profile searching for member field of
interests

promotedService=member.mineProfile("getInterestingOffers");
        //display the promoted service on the PC output for the
employee to tell the member about it
        this.Display(promotedService);

  }
```

Listing 8-6: Borrowing a Book after RFID Pseudo Code

8.5.2.3 Scenario Three

Plain Text

In the method before the introduction of RFID, the service and new collection are announced for all the library patrons through emails or through the LCD screens positioned near the setting and study areas. The services are prompted in a generic fashion without considering the different types of the library patrons and their variety of interests. After utilising RFID, the library installs a number of RFID readers next to the LCD screens. This is done in addition to replacing the current library patrons' cards with RFID-enabled ones. In the new system, the library wants to change its current promotion method of the new services and the new book collection.
With the RFID system tracking service, the promotions for new services or new books displayed on the LCDs are customised as much as possible for a specific patron or group of patrons. If only one patron is sitting near the LCD, a patron-customised message which matches his interest is displayed, based on his profile. If the patron is new, a generic promotion is displayed. In the case of a group of patrons existing in the same location, the promotion displayed on the LCD matches a common interest of the group if possible. Otherwise a generic promotion is displayed.

Analysis

For the above scenario, the scenario elements based on Carroll method are:

- **Setting**: is a library setting and study areas. The scenario implies further setting elements by identifying the person as a patron or a group of patrons. The library uses an RFID system in its operations and LCD screens as bulletin boards.
- **Actors**: the patron or group of patrons are the actors in this scenario.
- **Actions**: sitting or reading a book in the study or reading areas.
- **Events**: retrieving the patron or group of patrons' profiles, displaying the appropriate promotion of the LCD screen.
- **Goal**: displaying a promotion on the LCD screen.

Model

For the above scenario, the scenario in use case tabular form is presented in Figure 8-20 and the model in BPMN is depicted in Figure 8-21.

Use case No.*z*	LCD Promotion
Precondition	(none)
Trigger	Patron(s) is sitting near the LCD
Post-condition	Promotion is displayed
Main Scenario	9. Check for patron(s) 10. Check for patron profile 11. If one patron with profile then display patron's interest. 12. If more than one patron with profiles then display common interest promotion. 13. If no profiles then display generic promotion.
Extensions	1a if no patrons near LCD then continue check for patrons

Figure 8-20: LCD Promotion in Use Case Tabular Form

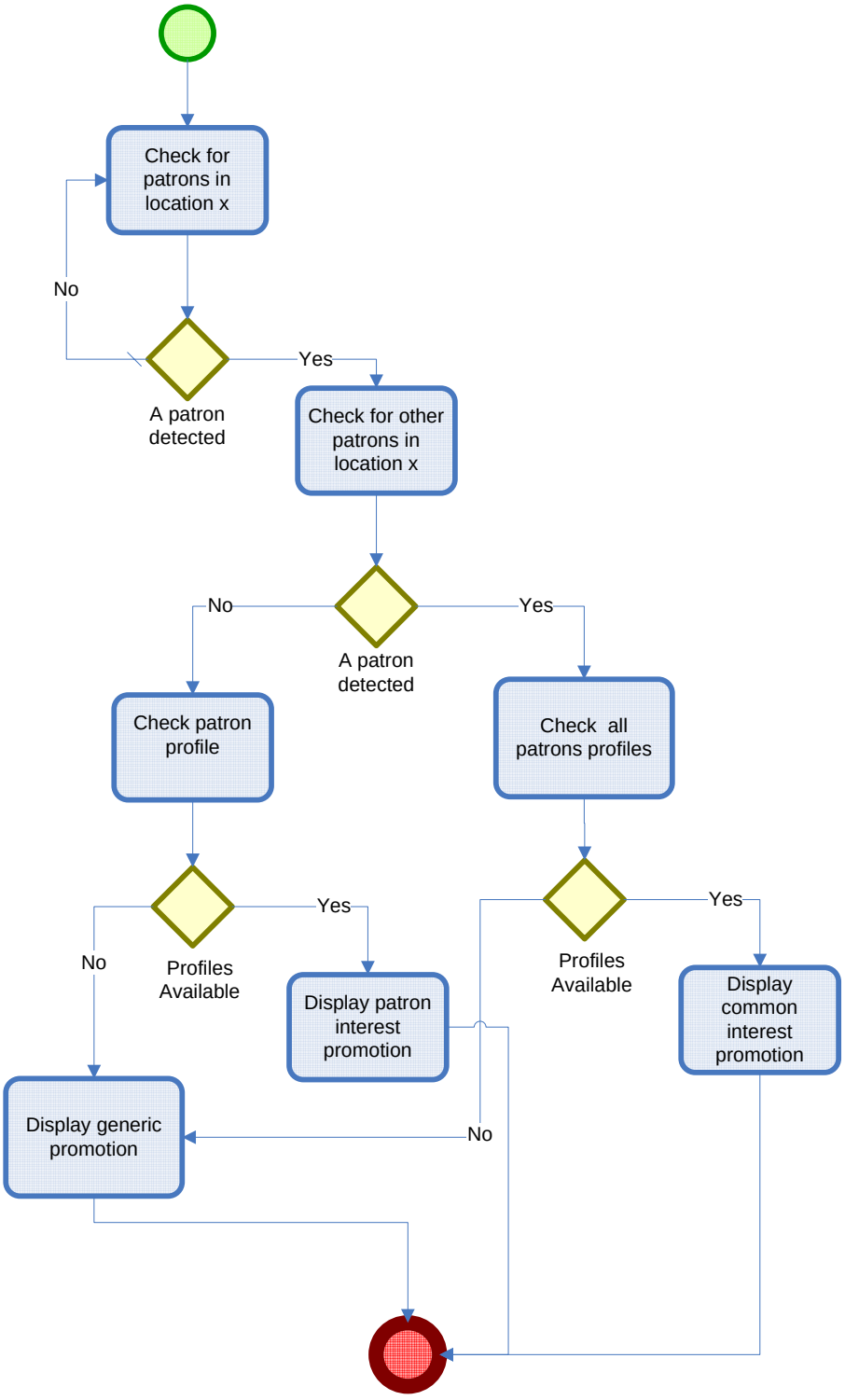

Figure 8-21: Scenario of LCD Promotion after RFID in BPMN

Pseudo Code

For the above scenario, the pseudo code for the recording of promoting a service or a book after RFID is illustrated in Listing 8-7

```
 public void promoteAService(String readerIp,List tags)
{   //get the LCD using the reader IP
   currentLocationLCD=this.getLCDByReaderIp(readerIp);
   //if there was only one tag (patron) in the range of the reader
   if(tags.size()==1)
   {       //use the first and only tag in the list
      patronTag=tags.get(0);
      //get the patron profile using his tag id
      patron=this.getPatronByTagId(patronTag.getId());
      //using the method getNewPromotedService which is assumed to
do some data minning and promote a new suitable service for the patron
      promotedService=patron.getNewPromotedService();
      //update patron's history
      patron.updatePromotedServices(promotedService);
   }
   // else it's more than one patron in the range
   else
   {       //get a general promotion service
      promotedService=this.getGeneralPromotionService();
      //get an iterator to iterate the tags list
      tagsIterator=tagList.iterator();
      //while there;s still tags in the list
      while(tagsIterator.hasNext())
      {          // get the tag on turn
         patronTag=(Tag)tagsIterator.next();
         //get the patron profile using his tag id
         patron=this.getPatronByTagId(patronTag.getId());
         //update patron's history
         patron.updatePromotedServices(promotedService);
      }   }   //display the service message on the LCD
   currentLocationLCD.display(promotedService.getMessage());
```

Listing 8-7: Promoting a Service after RFID Pseudo Code

8.6 Integration Stage

This stage aims to integrate new or changed components and integration of the required hardware handler for the utilised ubiquitous computing technology. In this case study, the technology employed is RFID.

8.6.1 Integrating Hardware Handlers

Depending on the RFID system used, the integration of the hardware handler will be done according to the integration patterns which were described in Chapter 7, Section 7.3. Integration patterns include File Transfer, Shared Database, Remote Procedure and Invocation, Messaging pattern. An example of RFID hardware handler components which can be used in this case study is shown in Figure 8-22. The class diagram was generated using Omondo EclipseUML.

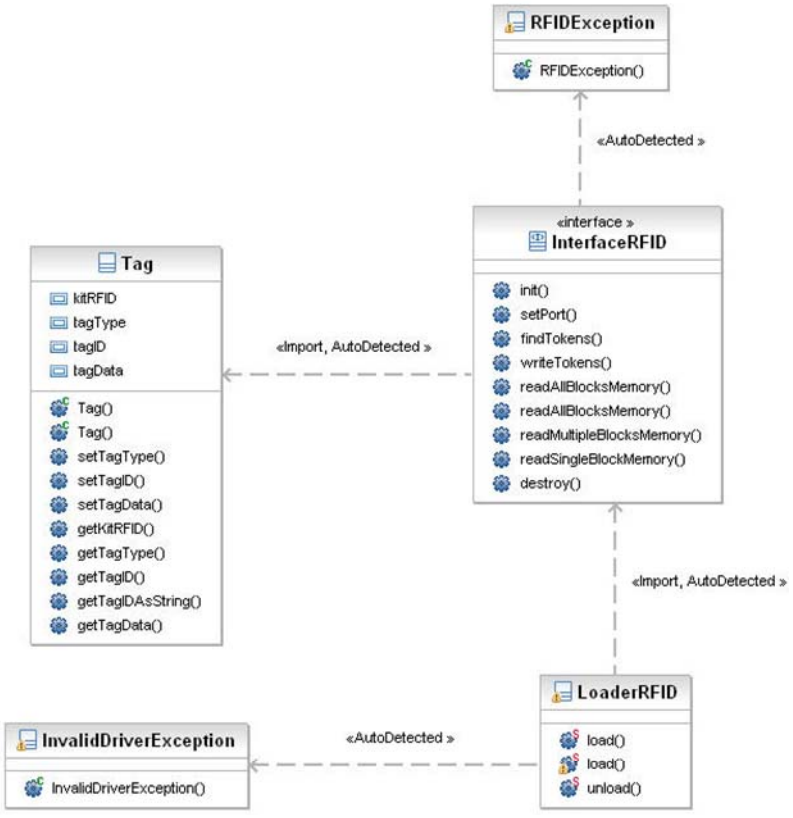

Figure 8-22: Example of RFID Hardware Handler Components

8.6.2 Integrating New/Changed Components

In this step, the algorithm explained in Chapter 7, Section 7.2.1, is executed. The algorithm makes use of both the extracted business logic and the new determined business logic, both of which are represented in BPMN. In this case study, the extracted business logic contains two functions:

1. Borrowing a book
2. Returning a book

The business logics for these two functions represented in BPMN are shown in Figure 8-11 and Figure 8-12 as explained in Section 8.4.1. Whereas the new business logic resulted from the additional requirements engineering stage comprises three functions:

1. Recording a patron's location.
2. Borrowing a book.
3. Promoting a service on LCD screens.

The business logics for these three functions represented in BPMN are shown in Figure 8-17, Figure 8-19 and Figure 8-21, as discussed in Section 8.5.

The integration process targeted by the integration algorithm is at a higher abstract layer which in our approach is at the diagram level. The algorithm runs a comparison between the two BPMN diagrams of the business logic of each function. In this case study, the comparison was carried out manually. After going through the logic of the algorithm, the result of the comparison led to the following actions.

8.6.2.1 First: Adding Two Functions

The two functions: record a patron location and promote a service on LCD screen are to be added to the system. This change is proposed by the first case of the

integration algorithm, which states that:

Case 1: if a previous activity of a new business rule does not match (≠) any previous activity of all old business rules, then the components of the entire new business rule are added.

The two functions pre-activities do not match any of the old business logic pre-activities. This is understandable as the two functions were an outcome from the introduction of the RFID to the system

8.6.2.2 Second: Not Changing the Return Book Function

The return books function business logic has not been changed after the introduction of RFID. Hence, the function will not be changed as it does not fall into any of the algorithm cases. The pre-activities and the post-activities of the business rules in the logic of function match each other. Therefore, there is no change required. The return book function business logic remains the same as illustrated in Figure 8-12

8.6.2.3 Third: Changing the Borrow Book Function

The borrowing book business logic changed to the new business logic to comply with the new logic resulting from the introduction of RFID. This change is proposed by the third case of the integration algorithm, which states that:

Case 3: if a previous activity of a new business rule matches (=) any previous activity of an old business rule, but the post activity of the new business rule does not match (≠) the post activity of the old business rule then only the components of the post activity of the new business rule replace the post activity of the old business rule.

As show in Figure 8-23 and Figure 8-24, the pre-activity in the new business logic "Lend the Book & Update No of Books borrowed by Patron" matches the pre-activities in the old business logic, but the post activity in the new business logic "Check Patron Profile" does not match the post activity in the old business logic "End". As a consequence, in the library system used in this case study; the required changes will be

to add what inside the dotted line shown in Figure 8-25.

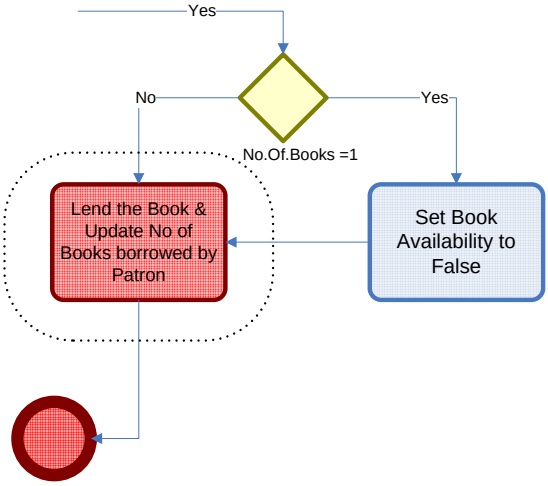

Figure 8-23: Matched Pre-activity in Old Business Logic

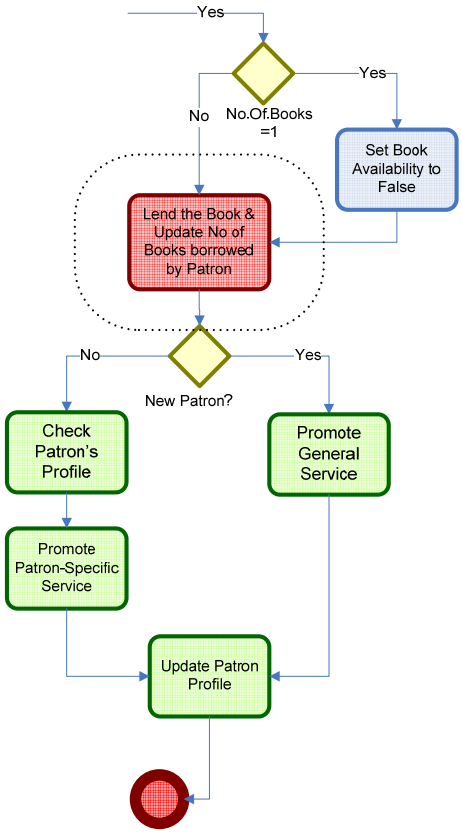

Figure 8-24: Matched Pre-activity in New Business Logic

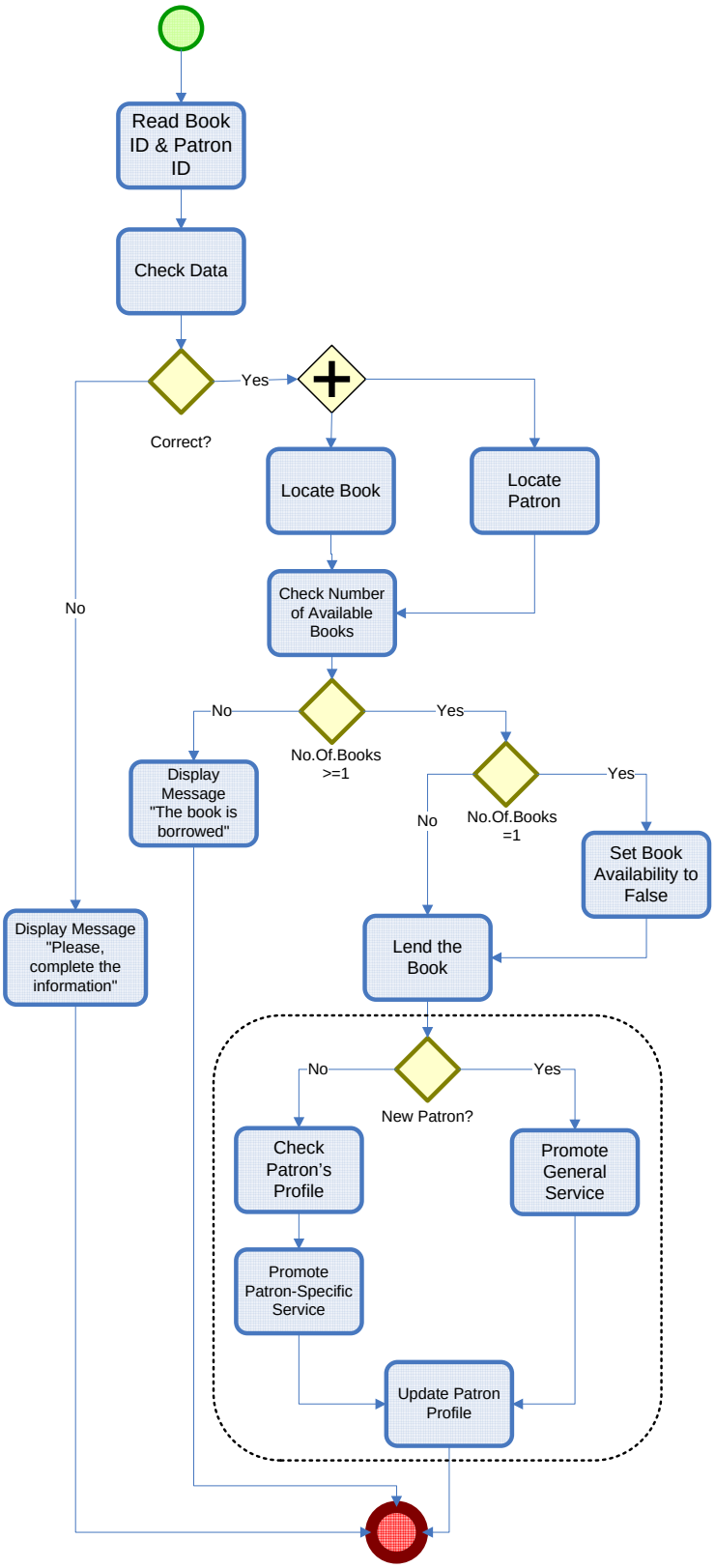

Figure 8-25: Changes to be made in the Borrowing Book Function

The overall outcome of this integration is one of these possibilities: addition of new component, removing a component, or modifying an existing component. Since the business logic compared may be extracted from a single or multiple classes or code entities, a component can be a number of lines of code in a method, method of a class, or in some cases a complete class. The integration process is at a higher abstract layer which in our approach is at the diagram level. The lower layer integration tasks at the source code need to address all the integration considerations discussed in Chapter 7, Section 7.2.2.

8.7 Summary

The purpose of this library management system case study is to demonstrate that the proposed reengineering framework has the ability to assist in reengineering existing systems, which aim to benefit the function introduced by a ubiquitous technology. The library management system is a Java program, which uses a Microsoft Access database to perform the basic functions operated in a library. The system's functions, along with some source code metrics, were given in the overview about the system.

After providing background information about the system used as a case study, a brief description was given about the tools that were used in the case study in addition to the tools that were experimented with but not used. Tools that were not used include PMD, Flowchart4j, WebSphere Business Modeller Basic and Green UML. On the other hand, tools that were experimented with and used are Eclipse, Omondo EclipseUML , Visustin and Doxygen.

Then in the chapter, the first stage of the framework is demonstrated. First, the business logic extraction step buried in the source code and represented in BPMN is given. The extraction process comprises three main steps: code grouping, extracting code in a code group and integrating the extracted business logics. The two steps are demonstrated in this case study. Second, the static structure of the system is recovered. In this step, the class diagram is recovered statically from the source code.

The code grouping steps cluster all the classes into the three groups. Then four more internal steps are executed on the source code; these are: code parsing, code refinement, UML activity diagram generation and BPMN diagram generation.

The then second stage of the framework is demonstrated, using three different scenarios. The three scenarios are describer in plain text, analysed using the Carroll's analysis method and modelled using BPMN, and then the pseudo codes of the new functions are provided. The RFID is the ubiquitous technology example used in this case study.

At the end of the chapter, the integration stage is provided. The stage demonstration showed both the integration of the new/changed system components and the integration of the hardware handlers of the RFID technology.

Chapter 9

Conclusions

Objectives

- To provide a summary for the thesis.
- To provide an evaluation by revisiting the research questions.
- To highlight the limitations of the work.
- To identify areas for future work.

9.1 Summary of Thesis

In this thesis a reengineering approach for software systems complying with the utilisation of ubiquitous technologies is introduced. The reengineering approach is represented as a unified framework composed of four stages. The stages are: program understanding, additional-requirements engineering, integration, and testing and operation.

The purpose of the first stage is to extract the existing business logic from the source code into a BPMN diagram. In addition, in this stage the static architecture of the software is recovered using UML class diagram. In the second stage, the user functional requirements are elicited using a scenario-based requirements elicitation method. The scenario is first described in plain text, analysed and modelled, and finally a pseudo code of the potential new functions is developed. In the third stage, the system is integrated in a high view layer depicted in BPMN diagrams through which the required changes are identified. The hardware handlers of the used ubiquitous technology are also integrated in this stage. The last stage of the reengineering framework, testing and operation, is discussed in a broad fashion, as it is beyond the scope of this thesis.

In the second chapter of the thesis, the background and the definitions of the concepts that form the foundation of this research were reviewed. Such basic concepts related to software engineering include software evolution, legacy systems, software reengineering, reverse engineering and forward engineering. In addition, other concepts related to the proposed reengineering framework which are business logic and ubiquitous computing were also explored.

In the third chapter, the work related to the proposed reengineering framework were reviewed. The concepts and studies covered in this chapter include program understanding and reverse engineering methods, UML and BPMN. The up-to-date approaches in extracting business logic from the source code were also reviewed.

In chapter four, an overview of the proposed reengineering framework is provided. In chapters five to seven, the program understanding stage, additional-requirements

engineering stage and integration stage were elaborated upon respectively.

Before the conclusions of the thesis, the supporting tools that can be used in implementing the reengineering approach were discussed. The case study demonstrated confirms that the proposed approach was useful and applicable.

9.2 Research Questions Revisited

To evaluate the work presented in the thesis which highlights the significance of the contributions claimed in the first chapter, the research questions are revisited. The overall research question presented in Chapter 1 is:

How to reengineer software systems to incorporate the new features introduced through the utilisation of ubiquitous technologies?

The question was answered in general by proposing a reengineering framework. The novelty of the proposed framework comes from the consideration of three aspects: business logic, ubiquitous technologies and the reengineering process itself. In order to be able to answer this question in detail, a set of research questions were defined.

RQ1. How to recover the business logic implemented in the existing system?

RQ1.1.What is business logic?

- This question was answered in chapter 5, Section 5.3.1.4 in which the term business logic used in this research was defined.

RQ1.2. How to extract business logic from source code?

- This question was answered in chapter 5, Section 5.3 in which the steps of the business logic extraction method was described.

RQ1.3.What other information need to be recovered to facilitate the completion of the reengineering process?

- These questions were answered in chapter 5, Section 5.2 which discussed the architecture recovery.

RQ2. How will features introduced by ubiquitous technologies impact on the existing systems?

RQ2.1. What is a ubiquitous technology?

RQ2.2. What are the features introduced by ubiquitous technologies?

RQ2.3. How will the new features affect the business logic?

RQ2.4 What new requirements may result from effects of ubiquitous technologies?

- These questions were answered in chapter 6, Section 6.1.1 which provided the related information.

RQ3. How can the new requirements resulted from effects of ubiquitous technologies be elicitated?

RQ3.1. What type of requirements to be elicitated?

RQ3.2 How the requirements can be elicitated?

- These questions were answered in chapter 6, Section 6.1.2 which discussed the proposed scenario-based requirements elicitation approach.

RQ4. How will the reengineered system be integrated?

RQ4.1. How to identify the software components which need to be added modified or removed?

RQ4.2. How to integrate the new or modified components?

RQ4.3. How to integrate the hardware handlers of the ubiquitous technologies?

- These questions were answered in chapter 7 which presented the developed integration

algorithm, integration considerations and to the methods of integrating the hardware handlers.

9.3 Limitations and Future Directions

Although the research questions raised at the beginning of the research have been addressed in the chapters of this thesis, any research has limitations. This research is not an exception. The proposed reengineering approach has facilitated the reengineering process of software systems aiming to take advantage of the new features offered by ubiquitous technologies. However, because of the time limitation, some details of the reengineering process have not been completely addressed. These limitations are spread among the four stages of the framework, and are the starting point for the suggested future development directions of this research.

In the first stage, recovering the architecture of the software was carried out using a static-analysis based method. Hence, a dynamic-analysis based method can also be utilised in this manner. In the business logic extraction process, the last step of the process, namely the integration of the extracted business logic, was not elaborated upon. In the code grouping step how code slicing techniques can be used to do the grouping is important future work. In the code refinement step, the business logic implemented inside the exception handling statements was not covered. Also, the abstraction algorithm can be described in a general form to cover more programming languages. Moreover, the code can be refactored before it is refined. At the end of the process, the additional elements of the UML and BPMN notation can be used to depict the extracted business logic.

In the second stage, this thesis focused on the user functional requirements only. Nevertheless, other forms of requirements can affect the reengineering process. Moreover, the other steps of requirements engineering process also can to be addressed.

In the third stage, the integration as a process is complex and needs more considerations that need to be resolved carefully to accomplish the aim of the process.

Low level integration issues in addition to the integration of the hardware handlers can be discussed further in future research.

The last stage of the framework is a good prospect for further investigation. The three aspects of the reengineering framework, which are the business logic, ubiquitous technologies and software engineering, all need to be regarded when designing a testing and operation strategy for a reengineered software system.

Considering the work done in a general manner, there are several areas of future work that can be pursued based on the present work:

- **Tool Support**. For the business logic extraction process, in order to reduce the human intervention in the evolution process, the steps which are currently performed manually will be automated where possible. For instance, the step of the framework can be integrated in a tool that will automate the whole process of business logic extraction. The source code will be parsed, the AST will be traversed, the abstraction algorithm will be applied, the activity diagram will be generated and finally the BPMN diagram will be generated.

- **Tool IDE Support**. It is important for the proposed tool to be developed as a Plug-in for mature production level IDEs, such as Eclipse, so that the work can be done with the full support of IDE supplied functionalities.

- **Infrastructure and Framework.** In this thesis, the discussion of some stages of the proposed framework is enough to demonstrate the reengineering of object-oriented software systems. However, the framework needs to be extended in the future to consider additional types of software systems, including legacy systems developed using COBOL or FORTRAN.

- **Case Studies.** Additional case studies can be conducted to provide a better determination of the industrial effectiveness of the proposed approach. The case studies should include real systems for the industrial field.

References

[1] W. Aalst and K. Hee, *Workflow Management: Models, Methods, and Systems*: MIT Press, 2004.

[2] W. Aalst, T. Weijters, and L. Maruster, "Workflow Mining: Discovering Process Models from Event Logs," *IEEE Transactions on Knowledge and Data Engineering,* vol. 16(9), pp. 1128-1142, Sep. 2004.

[3] H. Agrawal and J. R. Horgan, "Dynamic Program Slicing," in *Proceedings of the ACM SIGPLAN 1990 conference on Programming language design and implementation*, White Plains, New York, USA, 1990, pp. 246-256

[4] A. Aho, R. Sethi, and J. Ullman, *Compilers - Principles, Techniques & Tools*, 2nd ed.: Addison Wesley, 2007.

[5] Aivosto Oy, "Visustin," http://www.aivosto.com/visustin.html.

[6] M. Alawairdhi, H. Yang, and M. AL-Akhras, "BlueCRM: A New Trend of Customer Relationship Management Systems " in *The 12th IEEE International Workshop on Future Trends of Distributed Computing Systems*, Kunming, China, 2008, pp. 226-232.

[7] S. Ali, B. Soh, and T. Torabi, "Using Software Engineering Principles to Develop Reusable Business Rules," in *First International Conference on Information and Communication Technologies*, Cairo, Egypt, 2005, pp. 276-283.

[8] G. Alkhatib, "The Maintenance Problem of Application Software: An Empirical Analysis," *Journal of Software Maintenance: Research and Practice,* vol. 4(2), pp. 83-104, June 1992.

[9] P. Ammann and J. Offutt, *Introduction to Software Testing*: Cambridge University Press, 2008.

[10] Apache Maven Project, "PMD," http://pmd.sourceforge.net/.

[11] R. Arnold, *A Road Map Guide to Software Re-engineering*: IEEE Computer Society Press, 1992.

[12] L. Arthur, *Software Evolution: A Software Maintenance Challenge*: John Wiley and Sons Ltd., 1988.

[13] Ask.com, "Dictionary.com," http://dictionary.reference.com.

[14] M. Baker and S. Eick, "Visualizing software systems," in *The 16th International Conference on Software Engineering*, Sorrento, Italy, 1994, pp. 59-67

[15] F. Balmas, "Toward a Framework for Conceptual and Formal Outlines of Programs," in *The Fourth Working Conference on Reverse Engineering*, Amsterdam, Netherlands, 1997, pp. 226-235.

[16] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*: Addison-Wesley, 1998.

[17] P. Baumann, J. Faessler, M. Kiser, Z. Oeztuerk, and L. Richter, "Semantics-based Reverse Engineering," Department of Computer Science, University of Zurich, Switzerland, Technical Report 94.08, 1994.

[18] S. Ben, "The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations," in *The 1996 IEEE Symposium on Visual Languages*, Boulder, Colorado, USA., 1996, pp. 336-343.

[19] K. Bennett, "An Overview of Maintenance and Reverse Engineering," in *The Book - REDO Compendium: Reverse Engineering for Software Maintenance*: John Wiley and Sons Ltd., 1993, pp. 13-34.

[20] J. Bergeretti and B. Carr, "Information-Flow and Data-Flow Analysis of while-Programs," *ACM Transactions on Programming Languages and Systems,* vol. 7(1), pp. 37-61, Jan. 1985.

[21] D. Binkley, N. Gold, and M. Harman, "An Empirical Study of Static Program Slice Size," *ACM Transactions on Software Engineering and Methodology,* vol. 16(2), pp. 103-114, April 2007.

[22] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*: Addison Wesley, 1999.

[23] C. Bornhövd, T. Lin, S. Haller, and J. Schaper, "Integrating Automatic Data Acquisition With Business Processes Experiences With SAP's Auto-ID Infrastructure," in *The 13th International Conference on Very Large Data Bases* Toronto, Canada, 2004, pp. 1182-1188.

[24] M. Broadbent, P. Weill, and D. Clair, "The Implications of Information Technology Infrastructure for Business Process Redesign," *MIS Quarterly,* vol. 23(2), pp. 159-182, June 1999.

[25] M. Brodie and M. Stonebraker, *Migrating Legacy Systems: Gateways, Interfaces & the Incremental Approach*: Morgan Kaufmann Publishers Inc., 1995.

[26] F. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering," *IEEE Computer,* vol. 20(4), pp. 10-19, Apr. 1987.

[27] D. Buede, "Integrating Requirements Development and Decision Analysis," in *The IEEE International Conference on Systems, Man, and Cybernetics,* Orlando, Florida, USA: IEEE, 1997, pp. 1574-1579.

[28] D. Butler, J. Almond, R. Bergeron, K. Brodlie, and R. Haber, "Visualization Reference Models," in *The 4th Conference on Visualization* San Jose, California, 1993, pp. 337-342

[29] A. Campbell and A. Cox, "Scenario-Based Program Slicing," in *The 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, Belfast, Northern Ireland 2008, pp. 428-436.

[30] G. Canfora, A. Cimitile, and U. Carlini, "A Logic-Based Approach to Reverse Engineering Tools Production," *IEEE Transactions on Software Engineering* vol. 18(12), pp. 1053-1064, Dec. 1992.

[31] A. Caplinskas and O. Vasilecas, "Information Systems Research Methodologies and Models," in *The 5th international conference on Computer systems and technologies*, Rousse, Bulgaria, 2004, pp. 1 -6.

[32] S. Card, J. Mackinlay, and B. Shneiderman, "Readings in Information Visualization: Using Vision to Think," Morgan Kaufmann Publishers Inc., January 1999.

[33] J. Carroll, *Making Use Scenario-Based Design of Human Computer Interactions*: MIT Press, 2000.

[34] S. Cesare and A. Serrano, "Collaborative Modeling Using UML and Business Process Simulation," in *The 39th Annual Hawaii International Conference on System Sciences*, Kaua, Hawaii, USA, 2006, pp. 10b-10b.

[35] E. Chikofsky and J. Cross, "Reverse Engineering and Design Recovery: a Taxonomy," *IEEE Software,* vol. 7(1), pp. 13-17, Jan. 1990.

[36] A. Cimitile and U. Carlini, "Reverse Engineering: Algorithms for Program Graph Production," *Softwar: Practice and Experience,* vol. 21(5), pp. 519-537, May 1991.

[37] E. Clarke and J. Wing, "Formal Methods: State of the Art and Future Directions," *ACM Computing Surveys,* vol. 28(4), pp. 626-643, Sep. 1996.

[38] R. Clayton, S. Rugaber, and L. Wills, "On the Knowledge Required to Understand a Program," in *The Working Conference on Reverse Engineering,* Honolulu, Hawaii, USA: IEEE, 1998, pp. 69-78.

[39] A. Cockburn, *Writing Effective Use Cases*: Addison Wesley, 2000.

[40] CodeSWAT, "Flowchart4j Eclipse plug-in," http://www.codeswat.com.

[41] D. Cook and S. Das, *Smart Environments: Technology, Protocols and Applications*: John Wiley and Sons Ltd., 2004.

[42] I. Crnkovic and M. Larsson, *Building Reliable Component-Based Software Systems*: Artech House, 2002.

[43] T. Davenport, *Process Innovation: Reengineering Work Through Information Technology*: Harvard Business School Press, 1993.

[44] A. Davis, *Software Requirements: Objects, Functions and States*, 2nd ed.: Prentice-Hall Inc., 1993.

[45] R. Depke, G. Engels, S. Thöne, M. Langham, and B. Lütkemeier, "Process-Oriented, Consistent Integration of Software Components," in *The 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment*, Oxford, England, 2002, pp. 13

-18.

[46] A. Dey, "Understanding and Using Context," *Personal and Ubiquitous Computing,* vol. 5(1), pp. 4-7, Feb. 2001.

[47] Doxygen Plugin for Eclipse, "Eclox," http://www.easyeclipse.org/site/plugins/eclox-doxygen.html.

[48] S. Eick, J. Steffen, and E. Sumner, "Seesoft-A Tool for Visualizing Line Oriented Software Statistics," *IEEE Transactions on Software Engineering,* vol. 18(11), pp. 957-968, Nov. 1992.

[49] F. Elgar, "Business Impact of Pervasive Technologies: Opportunities and Risks," *Human and Ecological Risk Assessment Articles,* vol. 10(5), pp. 817 - 829, Oct. 2004.

[50] O. Elsawy, *Redesigning Enterprise Processes for E-Business*: McGraw-Hill Inc., 2001.

[51] H. Eriksson and M. Penker, *Business Modeling With UML: Business Patterns at Work*: John Wiley and Sons Ltd., 2000.

[52] Farlex Inc., "TheFreeDictionary.com," http://www.thefreedictionary.com.

[53] FateSoft, "Code Visual to Flow Chart," http://www.fatesoft.com.

[54] J. Ferrante, K. Ottenstein, and J. Warren, "The Program Dependence Graph and Its Use in Optimization," *ACM Transactions on Programming Languages and Systems,* vol. 9(3), pp. 319-349, July 1987.

[55] K. Finkenzeller, *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification*: John Wiley and Sons Ltd., 2003.

[56] E. Fleisch and M. Dierkes, "Ubiquitous Computing: Why Auto-ID is the Logical Next Step in Enterprise Automation," Auto-ID Center, Technical Report STG-AUTOID-WH-004, 2003.

[57] D. Foo, J. Guo, and Y. Zou, "Verifying Business Processes Extracted from E-Commerce Systems Using Dynamic Analysis " in *The 3rd International Workshop on Program Comprehension through Dynamic Analysis*, Vancouver, BC, Canada, 2007, pp. 43-48.

[58] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition*: Addison Wesley, Sep. 2003.

[59] J. Fraden, *Handbook of Modern Sensors: Physics, Designs, and Applications*: Springer Publishing, 2003.

[60] M. Fraser, K. Kumar, and V. Vaishnavi, "Informal and Formal Requirements Specification Languages: Bridging the Gap," *IEEE Transactions on Software Engineering,* vol. 17(5), pp. 454-466, May 1991.

[61] C. Gal, J. Martin, A. Lux, and J. Crowley, "SmartOffice: Design of an Intelligent Environment," *IEEE Intelligent Systems,* vol. 16(4), pp. 60-66, July 2001.

[62] B. Glover and H. Bhatt, *RFID Essentials*: O'Reilly Media Inc., 2006.

[63] D. Grant, "A Wider View of Business Process Reengineering," *Communications of the ACM,* vol. 45(2), pp. 85-90, Feb. 2002.

[64] C. Griffen, R. Huang, Z. Sen, and M. Fiammante, "Transforming UML «Activity» Diagrams to WebSphere Business Modeler processes," *IBM WebSphere Developer Technical Journal,* vol. 6, 18 July 2007.

[65] J. Guo, K. Foo, L. Barbour, and Y. Zou, "A Business Process Explorer: Recovering Business Processes from Business Applications," in *The 30th international Conference on Software Engineering*, Leipzig, Germany, 2008, pp. 871-874.

[66] P. Gupta and D. Moitra, "Evolving A Pervasive IT Infrastructure: A Technology Integration Approach," *Personal and Ubiquitous Computing,* vol. 8(1), pp. 31-41, Feb. 2004.

[67] P. Hall, *Software Reuse and Reverse Engineering in Practice*: Chapman and Hall Ltd., 1992.

[68] B. Halle, *Business Rules Applied: Building Better Systems Using the Business Rules Approach*: John Wiley and Sons Ltd., 2001.

[69] M. Hammer and J. Champy, *Reengineering the Corporation: A Manifesto for Business Revolution*: Harper Business, 1994.

[70] H. Hamza and M. Fayad, "A Novel Approach for Managing and Reusing

Business Rules in Business Architectures," in *The ACS/IEEE International Conference on Computer Systems and Applications*, Cairo, Egypt, 2005, pp. 973-978.

[71] H. Harrington, *Business Process Improvement: The Breakthrough Strategy for Total Quality, Productivity, and Competitiveness*: McGraw-Hill Professional, 1991.

[72] D. Heesch, "Doxygen," http://www.stack.nl/~dimitri/doxygen/index.html.

[73] G. Heineman and W. Council, *Component-Based Software Engineering: Putting the Pieces Together*: Addsion-Wesley, 2001.

[74] B. Heorhiy and T. Alexandru, "Texture-based Visualization of Metrics on Software Architectures," in *The 4th ACM Symposium on Software Visuallization*, Ammersee, Germany, 2008, pp. 205-206

[75] J. Hightower and G. Borriello, "Location Systems for Ubiquitous Computing," *IEEE Computer,* vol. 34(8), pp. 57-66, Aug. 2001.

[76] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*: Addison Wesley, 2003.

[77] H. Holbrook, "A Scenario-Based Methodology for Conducting Requirements Elicitation," *ACM SIGSOFT Software Engineering Notes* vol. 15(1), pp. 95-104 Jan. 1990.

[78] M. Hung and Y. Zou, "Extracting Business Policies and Business Data from the Three-Tier Architecture Systems," in *International Workshop on Reverse Engineering To Requirements*, Pittsburgh, Pennsylvania, USA, 2005, pp. 212-217.

[79] M. Hung and Y. Zou, "Recovering Workflows from Multi Tiered E-commerce Systems," in *The 15th IEEE International Conference on Program Comprehension*, Alberta, Canada, 2007, pp. 198-207.

[80] IBM Business Process Modeling Software, "WebSphere Business Modeler Basic," http://www-01.ibm.com/software/integration/wbimodeler/entry/.

[81] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development*

Process: Addison Wesley, 1999.

[82] M. Jones, *Fundamentals of Object-Oriented Design in UML*, 2nd edition ed.: Addison Wesley, November 2000.

[83] B. Kitchenham, S. Pfleeger, L. Pickard, P. Jones, D. Hoaglin, K. E. Emam, and J. Rosenberg, "Preliminary Guidelines for Empirical Research in Software Engineering," *IEEE Transaction on Software Engineering,* vol. 28 (8), pp. 721-734, Aug. 2002.

[84] J. Koehler, G. Tirenni, and S. Kumaran, "From Business Process Model to Consistent Implementation: A Case for Formal Verification Methods," in *The 6th IEEE International Enterprise Distributed Object Computing Conference*, Lausanne, Switzerland, 2002, pp. 96-106.

[85] E. Korshunova, M. Petkovic, M. Brand, and M. Mousavi, "CPP2XMI: Reverse Engineering of UML Class, Sequence, and Activity Diagrams from C++ Source Code," in *The 13th Working Conference on Reverse Engineering*, Benevento, Italy, 2006, pp. 297-298

[86] A. Kovacic and A. Groznik, "The Business Rule-Transformation Approach," in *The 26th International Conference on Information Technology Interfaces*, Dubrovnik, Croatia, 2004, pp. 113-117.

[87] B. Kovitz, *Practical Software Requirements: A Manual of Content and Style*: Manning Publications, 1999.

[88] P. Kruchten, "The 4+1 View Model of Architecture," *IEEE Software,* vol. 12(6), pp. 42-50, Nov. 1995.

[89] M. Lanza, "CodeCrawler - Lessons Learned in Building a Software Visualization Tool," in *The Seventh European Conference on Software Maintenance and Reengineering* Benevento, Italy, 2003, pp. 409-418.

[90] S. Lauesen, *Software Requirements: Styles and Techniques*: Addison-Wesley, 2001.

[91] J. Leduc, Y. Labiche, and L. Briand, "Toward the Reverse Engineering of UML Sequence Diagrams for Distributed Java Software," *IEEE Transactions on*

Software Engineering, vol. 32(9), pp. 642-663, Sep. 2006.

[92] M. Lehman, "Laws of Software Evolution Revisited," in *The 5th European Workshop on Software Process Technology,* Nancy, France: Springer-Verlag, 1996, pp. 108-124.

[93] M. Lehman and L. Belady, *Program Evolution: Processes of Software Change*: Academic Press Professional Inc., 1985.

[94] M. Lehman and J. Ramil, "Software Evolution and Software Evolution Processes," *Annals of Software Engineering,* vol. 14(1), pp. 275-309, Dec. 2002.

[95] M. Lehman and J. Ramil, "Software Evolution in the Age of Component-Based Software Engineering," in *IEE Proceedings Software*, Dec. 2000, pp. 249-255.

[96] S. Letovsky, "Cognitive Processes in Program Comprehension," in *The 1st Workshop on Empirical Studies of Programmers*, Washington, D.C., USA, 1986, pp. 102-107.

[97] P. Liew, K. Kontogiannis, and T. Tong, "A Framework for Business Model Driven Development," in *The 12 International Workshop on Software Technology and Engineering Practice*, Chicago, Illinois, USA 2004, pp. 47-56

[98] R. Lintern, J. Michaud, M. Storey, and X. Wu, "Plugging-in Visualization: Experiences Integrating a. Visualization Tool with Eclipse," in *The 2003 ACM symposium on Software visualization*, San Diego, California, 2003, pp. 47-56.

[99] D. Lübke and K. Schneider, "Visualizing Use Case Sets as BPMN Processes," in *The 2008 Requirements Engineering Visualization* Vienna, Austria, 2008, pp. 21-25.

[100] A. Lucia, "Program Slicing: Methods and Applications," in *The 1st IEEE International Workshop on Source Code Analysis and Manipulation*, Florence, Italy, 2001, pp. 142-149.

[101] C. Lung, X. Xu, and M. Zaman, "Software Architecture Decomposition Using Attributes," *International Journal of Software Engineering and Knowledge,* vol. 17(5), pp. 599-613, Oct. 2007.

[102] N. Madhavji, J. Ramil, and D. Perry, *Software Evolution and Feedback: Theory*

and Practice: John Wiley and Sons Ltd., 2006.

[103] T. Maibaum and Z. Li, "A Test Framework for Integration Testing of Object-Oriented Programs," in *The 2007 Conference of the Center for Advanced Studies on Collaborative Research,* Richmond Hill, Ontario, Canada: ACM, 2007, pp. 252-255.

[104] S. J. Mellor and M. J. Balcer, *Executable UML: A Foundation for Model-Driven*: Addison Wesley, 2002.

[105] T. Mens and S. Demeyer, *Software Evolution*: Springer Publishing, 2008.

[106] S. Miles, S. Sarma, and J. Williams, *RFID Technology and Applications*: Cambridge University Press, 2008.

[107] R. Mili and R. Steiner, "Software Engineering - Introduction," *Lecture Notes In Computer Science,* vol. 2269, pp. 129-137, May 2002.

[108] T. Morgan, *Business Rules and Information Systems: Aligning IT with Business Goals*: Addison-Wesley, 2002.

[109] J. Morrison and J. George, "Exploring the Software Engineering Component in MIS Research," *Communications of the ACM,* vol. 38 (7), pp. 80–91, July 1995.

[110] H. Müller, J. Jahnke, D. Smith, M. Storey, S. Tilley, and K. Wong, "Reverse Engineering: A Roadmap," in *The Conference on The Future of Software Engineering*, Limerick, Ireland, 2000, pp. 47-60.

[111] H. Müller, S. Tilley, and K. Wong, "Understanding Software Systems Using Reverse Engineering Technology Perspectives from the Rigi Project," in *The Conference of the IBM Centre For Advanced Studies on Collaborative Research* Toronto, Ontario, Canada, 1993, pp. 217-226.

[112] P. Muller, *Instant UML*: Wrox Press, December 2000.

[113] E. Murnane and V. Sinha, "Interactive Exploration of Compacted Visualizations for Understanding Behavior in Complex Software," in *The Conference on Object Oriented Programming Systems Languages and Applications* Nashville, TN, USA, 2008, pp. 729-730.

[114] P. Nancy, "Comprehension Strategies in Programming," in *The 2nd Workshop*

on Empirical Studies of Programmers, Washington, D.C.,USA, 1987, pp. 100-113.

[115] S. Naqvi and M. Riguidel, "Security and Trust Assurances for Smart Environments," in *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference*, Washington, D.C., USA, 2005, pp. 234-242.

[116] I. Neamtiu, J. Foster, and M. Hicks, "Understanding Source Code Evolution Using Abstract Syntax Tree Matching," in *The International Workshop on Mining Software Repositories,* St. Louis, Missouri: ACM, 2005, pp. 1-5.

[117] B. Nuseibeh and S. Easterbrook, "Requirements Engineering: A Roadmap," in *The Conference on The Future of Software Engineering*, Limerick, Ireland, 2000, pp. 35-46.

[118] T. Okawa, T. Kaminishi, S. Hirabayashi, H. Koizumi, and J. Sawamoto, "An Information System Development Method Based on the Link of Business Process Modeling with Executable UML Modeling and its Evaluation by Prototyping," in *The 22nd International Conference on Advanced Information Networking and Applications*, Okinawa, Japan, 2008, pp. 1057-1064

[119] M. Olsem, "An Incremental Approach to Software Systems Re-engineering," *Journal of Software Maintenance: Research and Practice,* vol. 10(3), pp. 181-202, May 1998.

[120] OMG, "Business Process Modeling Notation, V1.2 (Beta 3)," OMG Document: BMI/2008-02-07, 2008

[121] OMG, "UML 2.1.2 Superstructure and Infrastructure," Document: formal/2007-11-04, 2007.

[122] Omondo Inc., "Omondo EclipseUML," http://www.uml2.org/index.html.

[123] M. Pacione, M. Roper, and M. Wood, "A Novel Software Visualisation Model to Support Software Comprehension," in *The 11th Working Conference on Reverse Engineering*, Delft, Netherlands, 2004, pp. 70-79

[124] D. Perry and A. Wolf, "Foundations for the Study of Software Architecture," *SIGSOFT Software Engineering Notes* vol. 17(4), pp. 40-52, Oct. 1992.

[125] C. Potts, K. Takahashi, and A. Anton, "Inquiry-Based Scenario Analysis of System Requirements," *IEEE Software,* vol. 11(2), pp. 21-32, March 1994.

[126] P. Prekop and M. Burnett, "Activities, Context and Ubiquitous Computing," *Computer Communications,* vol. 26(11), p. 1168, July 2003.

[127] A. Price, I. Small, and R. Baecker, "A Taxonomy of Software Visualization," in *The Twenty-Fifth Hawaii International Conference on System Sciences*, 1992, pp. 597-606

[128] J. Qian and B. Xu, "Program Slicing Under UML Scenario Models," *ACM SIGPLAN Notices,* vol. 43(2), pp. 21-24, Feb. 2008.

[129] B. Qiao, "Evolution of WEB-based Systems in Model Driven Architecture," PhD Thesis, De Montfort University, England, 2005.

[130] V. Ranganath and J. Hatcliff, "Slicing concurrent Java programs using Indus and Kaveri," *International Journal on Software Tools for Technology Transfer,* vol. 9(5), pp. 489-504, Oct. 2007.

[131] J. Rilling, M. Wen Jun, C. Fuzhi, and P. Charland, "Software Visualization - A Process Perspective," in *The 4th IEEE International Workshop onVisualizing Software for Understanding and Analysis* Alberta, Canada, 2007, pp. 10-17.

[132] D. Robert, K. Amir, C. Mary, and R. George, "Towards Understanding Programs. through Wear-based Filtering," in *The 2005 ACM Symposium on Software Visualization*, St. Louis, Missouri, 2005, pp. 183 -192.

[133] G. Roussos, *Networked RFID: Systems, Software and Services*: Springer Publishing, 2008.

[134] S. Rugaber, "The Use of Domain Knowledge in Program Understanding," *Annals of Software Engineering* vol. 9(4), pp. 143-192, Jan. 2000.

[135] D. Salber, A. Dey, and G. Abowd, "The Context Toolkit: Aiding the Development of Context-Enabled Applications," in *The SIGCHI Conference on Human Factors in Computing Systems*, Pittsburgh, Pennsylvania, USA, 1999, pp. 434-441.

[136] B. Schilit, N. Adams, and R. Want, "Context-Aware Computing Applications,"

in *The Workshop on Mobile Computing Systems and Applications*, Santa Cruz, Calfornia, USA, 1994, pp. 85-90.

[137] T. Scott and H. Shihong, "A Qualitative Assessment of the Efficacy of UML Diagrams as a Form of Graphical Documentation in Aiding Program Understanding," in *The 21st Annual International Conference on Documentation*, San Francisco, CA, USA, 2003, pp. 184-191

[138] H. Shah, C. Görg, and M. Harrold, "Visualization of Exception Handling Constructs to Support Program Understanding," in *The 4th ACM Symposium on Software Visuallization*, Ammersee, Germany, 2008, pp. 19-28.

[139] B. Shishkov and J. Dietz, "Design of Software Applications Using Generic Business Components," in *The 37th Annual Hawaii International Conference on System Sciences*, Big Island, Hawaii, USA, 2004, pp. 10-20.

[140] R. Sindhgatta and K. Pooloth, "Identifying Software Decompositions by Applying Transaction Clustering on Source Code," in *The 31st Annual International Computer Software and Applications Conference* Beijing, China: IEEE, 2007, pp. 317-326.

[141] H. Smith, "On Tool Selection for Illustrating the Use of UML in System Development," *Journal of Computing Sciences in Colleges,* vol. 19(5), pp. 53-63, May 2004.

[142] H. Sneed, "Extracting Business Logic from Existing COBOL Programs as a Basis for Redevelopment," in *The 9th International Workshop on Program Comprehension*, Toronto, Canada, 2001, pp. 167-175.

[143] H. Sneed and K. Erdos, "Extracting business rules from source code," in *The Fourth Workshop on Program Comprehension* Berlin, Germany: IEEE, 1996, pp. 240-247.

[144] I. Sommerville, *Software Engineering, 7th Edition*: Addison Wesley, 2004.

[145] C. Soon and J. Gutiérrez, "Effects of the RFID Mandate on Supply Chain Management," *Journal of Theoretical and Applied Electronic Commerce Research,* vol. 3(1), pp. 81-91, Apr. 2008.

[146] K. Spurr, L. Jennison, and P. Layzell, *Software Assistance for Business Re-Engineering* John Wiley and Son Ltd, 1994.

[147] M. Storey, F. Fracchia, and H. Mueller, "Cognitive Design Elements to Support the Construction of a Mental Model During Software Visualization," in *The 5th Workshop on Program Comprehension*, Dearborn, Michigan, USA, 1997, pp. 17-28.

[148] M. Strassner and T. Schoch, "Today's Impact of Ubiquitous Computing on Business Processes," in *1st International Conference on Pervasive Computing*, Zurich, Switzerland, 2002, pp. 62--74.

[149] A. Suenbuel and M. Shan, "Towards Enterprise Archeology: Extracting Business Processes from Runtime Event Data," in *The 6th IEEE International Conference on Data Mining*, Hong Kong, 2006, pp. 468-473.

[150] A. Sutcliffe, "Scenario-Based Requirements Engineering," in *The 11th IEEE International Conference on Requirements Engineering,* Monterey, California, USA: IEEE, 2003.

[151] A. Teyseyre and M. Campo, "An Overview of 3D Software Visualization," *IEEE Transactions onVisualization and Computer Graphics,* vol. 15(1), pp. 87-105, Jan. 2009.

[152] R. Thayer, S. Bailin, and M. Dorfman, *Software Requirements Engineerings, 2nd Edition*: IEEE Computer Society Press, 1997.

[153] The Eclipse Project, "AST View Eclipse plug-in," http://www.eclipse.org/jdt/ui/astview/index.php.

[154] The Eclipse Project, "Eclipse," http://www.eclipse.org.

[155] The Eclipse Project, "Metrics Eclipse Plug-in," http://metrics.sourceforge.net/.

[156] O. Thomann and T. Kuhn, "Abstract Syntax Tree," in *Eclipse Corner Articles*, 2006.

[157] M. Tian, "Scenario-Driven Requirements Engineering: Method and Tool," Master Thesis, Concordia University, Montreal, Quebec, Canada, 2003.

[158] S. Tilley, K. Wong, M. Storey, and H. Muller, "Programmable Reverse

Engineering," *International Journal of Software Engineering and Knowledge Engineering,* vol. 4(4), pp. 501-520, Dec. 1994.

[159] M. Tom and D. Serge, *Software Evolution*: Springer Publishing Company, Incorporated, 2008.

[160] P. Tonella and A. Potrich, *Reverse Engineering of Object Oriented Code*: Springer, 2005.

[161] P. Tonella and A. Potrich, "Reverse Engineering of the Interaction Diagrams from C++ Code," in *The International Conference on Software Maintenance*, Amsterdam, Netherlands, 2003, pp. 159-168.

[162] M. Tudoreanu, "Program Visualization: Designing Effective Visualization Tools for Reducing User's Cognitive Effort," in *The 2003 ACM symposium on Software visualization*, San Diego, California, 2003, pp. 105-111.

[163] C. Turner, A. Tiwari, and J. Mehnen, "A Genetic Programming Approach to Business Process Mining," in *The 10th Annual Conference on Genetic and Evolutionary Computation*, Atlanta, Georgia, USA, 2008, pp. 1307-1314.

[164] D. Valtchev and I. Frankov, "Service Gateway Architecture for a Smart Home," *IEEE Communications Magazine,* vol. 40(4), pp. 126-132, Apr. 2002.

[165] O. Vasilecas and A. Smaizys, "The Framework: an Approach to Support Business. Rule Based Data Analysis," in *The 7th International Baltic Conference on Databases and Information Systems*, 2006, pp. 141-147.

[166] N. Venkatraman, "IT-enabled Business Transformation: From Automation to Business Scope Redefinition," *MIT Sloan Management Review,* pp. 73-87, Winter, 1994.

[167] R. Walker, G. Murphy, J. Steinbok, and M. Robillard, "Efficient Mapping of Software System Traces to Architectural Views," in *The Conference of the IBM Centre for Advanced Studies on Collaborative Research*, Mississauga, Ontario, Canada, 2000, pp. 31-40.

[168] N. Walkinshaw, M. Roper, and M. Wood, "Understanding Object-Oriented Source Code from the Behavioural Perspective," in *The 13th International*

Workshop on Program Comprehension, St. Louis, Missouri, USA, 2005, pp. 215-224

[169] A. Wang and K. Qian, *Component-Oriented Programming*: John Wiley and Sons Ltd., 2005.

[170] G. Wang, B. McSkimming, Z. Marzec, J. Gardner, A. Decker, and C. Alphonce, "Green: A Flexible UML Class Diagramming Tool for Eclipse," in *The Conference on Object Oriented Programming Systems Languages and Applications* Montreal, Quebec, Canada, 2007, pp. 791-792

[171] S. Wang, W. Chen, C. Ong, L. Liu, and Y. Chuang, "RFID Application in Hospitals: A Case Study on a Demonstration RFID Project in a Taiwan Hospital," in *The 39th Annual Hawaii International Conference on System Sciences*, Kaua, Hawaii, USA, 2006, pp. 184a-184a.

[172] M. Ward and H. Zedan, "Slicing as a Program Transformation," *ACM Transactions on Programming Languages and Systems,* vol. 29(2), Mar. 2007.

[173] M. Weiser, "The Computer for the 21st Century," *ACM Mobile Computing and Communications Review,* vol. 3(3), pp. 933-940, July 1995.

[174] M. Weiser, "Program Slicing," in *The 5th international conference on Software engineering*, San Diego, California, USA, 1981, pp. 439-449

[175] M. Weiser, "Some Computer Science Issues in Ubiquitous Computing," *Communications of the ACM,* vol. 36(7), pp. 75-84, July 1993.

[176] M. Weske, *Business Process Management: Concepts, Languages, Architectures*: Springer Publishing, 2007.

[177] S. A. White, D. Miers, and L. Fischer, *BPMN Modeling and Reference Guide*: Future Strategies Inc., August 2008.

[178] B. Xu, J. Qian, X. Zhang, Z. Wu, and L. Chen, "A Brief Survey of Program Slicing," *ACM SIGSOFT Software Engineering Notes* vol. 30(2), pp. 1-36, March 2005.

[179] R. Xu and D. Wunsch, *Clustering*: Wiley-IEEE Press, 2008.

[180] H. Yang, *Software Evolution with UML and XML*: Idea Group Publishing, 2005.

[181] H. Yang and M. Ward, *Successful Evolution of Software Systems*: Artech House Inc., 2003.

[182] D. Zanardini, "The Semantics of Abstract Program Slicing," in *The 8th IEEE International Working Conference on Source Code Analysis and Manipulation*, Beijing, China, 2008, pp. 89-98.

[183] P. Zave, "Classification of Research Efforts in Requirements Engineering," *ACM Computing Surveys,* vol. 29(4), pp. 315-321, Dec. 1997.

[184] Y. Zou and M. Hung, "An Approach for Extracting Workflows from E-Commerce Applications," in *The 14th IEEE International Conference on Program Comprehension*, Athens, Greece, 2006, pp. 127-136

[185] Y. Zou, T. Lau, K. Kontogiannis, T. Tong, and R. McKegney, "Model-Driven Business Process Recovery," in *The 11th Working Conference on Reverse Engineering*, Delft, Netherlands, 2004, pp. 224-233.

Appendix A: Source Code of BorrowBooks Class

```
//import the packages for using the classes in them into the program

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.SimpleDateFormat;
import java.util.Locale;

/**
 * A public class
 */
public class BorrowBooks extends JInternalFrame {

/*****************************************************************
******
        * ** declaration of the private variables used in the program ***

*****************************************************************
****/

       // for creating the North Panel
       private JPanel northPanel = new JPanel();
       // for creating the label
       private JLabel title = new JLabel("BOOK INFORMATION");

       // for creating the Center Panel
       private JPanel centerPanel = new JPanel();
       // for creating an Internal Panel in the center panel
       private JPanel informationPanel = new JPanel();
       // for creating an array of JLabel
       private JLabel[] informationLabel = new JLabel[4];
       // for creating an array of String
       private String[] informationString = { " Write the Book ID:",
                     " Write the Member ID:", " The Current Data:", " The
Return Date:" };
       // for creating an array of JTextField
       private JTextField[] informationTextField = new JTextField[4];
       // for creating the date in the String
       private String date = new SimpleDateFormat("dd-MM-yy",
Locale.getDefault())
                     .format(new java.util.Date());
       // for creating an array of string to store the data
       private String[] data;

       // for creating an Internal Panel in the center panel
       private JPanel borrowButtonPanel = new JPanel();
       // for creating the button
       private JButton borrowButton = new JButton("Borrow");
```

```
	// for creating South Panel
	private JPanel southPanel = new JPanel();
	// for creating the button
	private JButton cancelButton = new JButton("Cancel");

	// for creating an object
	private Books book;
	private Members member;
	private Borrow borrow;

	// for checking the information from the text field
	public boolean isCorrect() {
		data = new String[4];
		for (int i = 0; i < informationLabel.length; i++) {
			if (!informationTextField[i].getText().equals("")) {
				data[i] = informationTextField[i].getText();
			} else {
				return false;
			}
		}
		return true;
	}

	// for setting the array of JTextField to null
	public void clearTextField() {
		for (int i = 0; i < informationTextField.length; i++) {
			if (i != 2) {
				informationTextField[i].setText(null);
			}
		}
	}

	// constructor of borrowBooks
	public BorrowBooks() {
		// for setting the title for the internal frame
		super("Borrow Books", false, true, false, true);
		// for setting the icon
		setFrameIcon(new ImageIcon(ClassLoader
				.getSystemResource("images/Export16.gif")));
		// for getting the graphical user interface components display
area
		Container cp = getContentPane();

		// for setting the layout
		northPanel.setLayout(new FlowLayout(FlowLayout.CENTER));
		// for setting the font
		title.setFont(new Font("Tahoma", Font.BOLD, 14));
		// for adding the label to the panel
		northPanel.add(title);
		// for adding the panel to the container
		cp.add("North", northPanel);

		// for setting the layout
		centerPanel.setLayout(new BorderLayout());
		// for setting the layout for the internal panel
		informationPanel.setLayout(new GridLayout(4, 2, 1, 1));
```

```
/*****************************************************************
**
                * for adding the strings to the labels, for setting the font
* and
                * adding these labels to the panel. * finally adding the panel
to the
                * container *

*****************************************************************
/
              for (int i = 0; i < informationLabel.length; i++) {
                     informationPanel.add(informationLabel[i] = new
JLabel(
                                   informationString[i]));
                     informationLabel[i].setFont(new Font("Tahoma",
Font.BOLD, 11));
                     if (i == 2) {
                            informationPanel.add(informationTextField[i]
= new JTextField(
                                          date));
                            informationTextField[i].setFont(new
Font("Tahoma", Font.PLAIN,
                                          11));
                            informationTextField[i].setEnabled(false);
                     } else {
                            informationPanel
                                          .add(informationTextField[i] =
new JTextField());
                            informationTextField[i].setFont(new
Font("Tahoma", Font.PLAIN,
                                          11));
                     }
              }
              centerPanel.add("Center", informationPanel);

              // for setting the layout
              borrowButtonPanel.setLayout(new
FlowLayout(FlowLayout.RIGHT));
              // for setting the font to the button
              borrowButton.setFont(new Font("Tahoma", Font.BOLD, 11));
              // for adding the button to the panel
              borrowButtonPanel.add(borrowButton);
              // for adding the panel to the center panel
              centerPanel.add("South", borrowButtonPanel);
              // for setting the border to the panel
              centerPanel.setBorder(BorderFactory
                            .createTitledBorder("Borrow a book:"));
              // for adding the panel to the container
              cp.add("Center", centerPanel);

              // for adding the layout
              southPanel.setLayout(new FlowLayout(FlowLayout.RIGHT));
              // for setting the font to the button
              cancelButton.setFont(new Font("Tahoma", Font.BOLD, 11));
              // for adding the button to the panel
              southPanel.add(cancelButton);
              // for setting the border to the panel
              southPanel.setBorder(BorderFactory.createEtchedBorder());
```

```
            // for adding the panel to the container
            cp.add("South", southPanel);


/**************************************************************
**
             * for adding the action listener to the button,first the text
will be *
             * taken from the JTextField[] and make the connection for
database, *
             * after that update the table in the database with the new
value *
**************************************************************
/
            borrowButton.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent ae) {
                        // for checking if there is a missing information
                        if (isCorrect()) {
                              Thread runner = new Thread() {
                                    @Override
                                    public void run() {
                                          book = new Books();
                                          member = new Members();
                                          borrow = new Borrow();
                                          book
                                                      .connectio
n("SELECT * FROM Books WHERE BookID = "

+ data[0]);
                                          member
                                                      .connectio
n("SELECT * FROM Members WHERE MemberID = "

+ data[1]);
                                          int
numberOfAvailbleBooks = book
                                                      .getNumber
OfAvailbleBooks();
                                          int
numberOfBorrowedBooks = 1 + book
                                                      .getNumber
OfBorrowedBooks();
                                          int numberOfBooks = 1 +
member.getNumberOfBooks();
                                          // for checking if there is
no same information in
                                          // the database
                                          if
(numberOfAvailbleBooks == 1) {

numberOfAvailbleBooks -= 1;
                                                book
                                                            .upd
ate("UPDATE Books SET NumberOfAvailbleBooks ="

+ numberOfAvailbleBooks
```

```
+ ",NumberOfBorrowedBooks ="

+ numberOfBorrowedBooks

+ ",Availble = false WHERE BookID ="

+ data[0]);
                                                        member
                                                                .upd
ate("UPDATE Members SET NumberOfBooks = "

+ numberOfBooks

+ " WHERE MemberID = "

+ data[1]);
                                                        borrow
                                                                .upd
ate("INSERT INTO Borrow (BookID, MemberID, DayOfBorrowed, DayOfReturn)
VALUES ("

+ data[0]

+ ","

+ data[1]

+ ",'"

+ data[2]

+ "','"

+ data[3] + "')");
                                                        // for setting the
array of JTextField to null
                                                        clearTextField();
                                                    } else if
(numberOfAvailbleBooks > 1) {

numberOfAvailbleBooks -= 1;
                                                        book
                                                                .upd
ate("UPDATE Books SET NumberOfAvailbleBooks ="

+ numberOfAvailbleBooks

+ ",NumberOfBorrowedBooks ="

+ numberOfBorrowedBooks

+ " WHERE BookID =" + data[0]);
                                                        member
                                                                .upd
ate("UPDATE Members SET NumberOfBooks ="

+ numberOfBooks
```

```
+ " WHERE MemberID =" + data[1]);
                                                        borrow
                                                                .upd
ate("INSERT INTO Borrow (BookID, MemberID, DayOfBorrowed, DayOfReturn)
VALUES ("

+ data[0]

+ ","

+ data[1]

+ ",'"

+ data[2]

+ "','"

+ data[3] + "')");
                                                        // for setting the
array of JTextField to null
                                                        clearTextField();
                                                } else {

JOptionPane.showMessageDialog(null,
                                                                "The
book is borrowed", "Warning",

JOptionPane.WARNING_MESSAGE);
                                                }
                                        }
                                };
                                runner.start();
                        }
                        // if there is a missing data, then display
Message Dialog
                        else {
                                JOptionPane.showMessageDialog(null,
                                        "Please, complete the
information", "Warning",

JOptionPane.WARNING_MESSAGE);
                        }
                }
        });
        // for adding the action listener for the button to dispose
the frame
        cancelButton.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent ae) {
                        dispose();
                }
        });
        // for setting the visible to true
        setVisible(true);
        // show the internal frame
        pack();
    }
}
```

Appendix B: Source Code of ReturnBooks Class

```
//import the packages for using the classes in them into the program

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 *A public class
 */
public class ReturnBooks extends JInternalFrame {

/*************************************************************************
******
       ***     declaration of the private variables used in the program
***

************************************************************************
*****/

       //for creating the North Panel
       private JPanel northPanel = new JPanel();
       //for creating the label
       private JLabel title = new JLabel("BOOK INFORMATION");

       //for creating the Center Panel
       private JPanel centerPanel = new JPanel();
       //for creating an Internal Panel in the center panel
       private JPanel informationPanel = new JPanel();
       //for creating an array of JLabel
       private JLabel[] informationLabel = new JLabel[2];
       //for creating an array of String
       private String[] informationString = {" Write the Book ID:", " Write
the Member ID:"};
       //for creating an array of JTextField
       private JTextField[] informationTextField = new JTextField[2];
       //for creating an array of string to store the data
       private String[] data;

       //for creating an Internal Panel in the center panel
       private JPanel returnButtonPanel = new JPanel();
       //for creating the buton
       private JButton returnButton = new JButton("Return");

       //for creating the panel
       private JPanel southPanel = new JPanel();
       //for creating the button
       private JButton cancelButton = new JButton("Cancel");

       //for creating an object
```

```
	private Books book;
	private Members member;
	private Borrow borrow;


	//for checking the information from the text field
	public boolean isCorrect() {
		data = new String[2];
		for (int i = 0; i < informationLabel.length; i++) {
			if (!informationTextField[i].getText().equals(""))
				data[i] = informationTextField[i].getText();
			else
				return false;
		}
		return true;
	}

	//for setting the array of JTextField to null
	public void clearTextField() {
		for (int i = 0; i < informationTextField.length; i++)
			if (i != 2)
				informationTextField[i].setText(null);
	}

	//constructor of returnBooks
	public ReturnBooks() {
		//for setting the title for the internal frame
		super("Return books", false, true, false, true);
		//for setting the icon
		setFrameIcon(new
ImageIcon(ClassLoader.getSystemResource("images/Import16.gif")));
		//for getting the graphical user interface components display
area
		Container cp = getContentPane();

		//for setting the layout
		northPanel.setLayout(new FlowLayout(FlowLayout.CENTER));
		//for setting the font
		title.setFont(new Font("Tahoma", Font.BOLD, 14));
		//for adding the label
		northPanel.add(title);
		//for adding the north panel to the container
		cp.add("North", northPanel);

		//for setting the layout
		centerPanel.setLayout(new BorderLayout());
		//for setting the layout for the internal panel
		informationPanel.setLayout(new GridLayout(2, 2, 1, 1));


/*****************************************************************
**
		 * for adding the strings to the labels, for setting the font
*
		 * and adding these labels to the panel.
*
		 * finally adding the panel to the container
*
```

```
*************************************************************************
*/
              for (int i = 0; i < informationLabel.length; i++) {
                     informationPanel.add(informationLabel[i] = new
JLabel(informationString[i]));
                     informationLabel[i].setFont(new Font("Tahoma",
Font.BOLD, 11));
                     informationPanel.add(informationTextField[i] = new
JTextField());
                     informationTextField[i].setFont(new Font("Tahoma",
Font.PLAIN, 11));
              }
              centerPanel.add("Center", informationPanel);
              //for setting the layout
              returnButtonPanel.setLayout(new
FlowLayout(FlowLayout.RIGHT));
              //for adding the button
              returnButtonPanel.add(returnButton);
              //for setting the font to the button
              returnButton.setFont(new Font("Tahoma", Font.BOLD, 11));
              //for adding the internal panel to the panel
              centerPanel.add("South", returnButtonPanel);
              //for setting the border

centerPanel.setBorder(BorderFactory.createTitledBorder("Return a
book:"));
              //for adding the center panel to the container
              cp.add("Center", centerPanel);

              //for setting the layout
              southPanel.setLayout(new FlowLayout(FlowLayout.RIGHT));
              //for adding the button
              southPanel.add(cancelButton);
              //for setting the font to the button
              cancelButton.setFont(new Font("Tahoma", Font.BOLD, 11));
              //for setting the border
              southPanel.setBorder(BorderFactory.createEtchedBorder());
              //for adding the south panel to the container
              cp.add("South", southPanel);


/************************************************************************
**
               * for adding the action listener to the button,first the text
will be *
               * taken from the JTextField and make the connection for database,
*
               * after that update the table in the database with the new
value     *

*************************************************************************
*/
              returnButton.addActionListener(new ActionListener() {
                     public void actionPerformed(ActionEvent ae) {
                            //for checking if there is a missing information
                            if (isCorrect()) {
                                   Thread runner = new Thread() {
```

```
                                        public void run() {
                                                book = new Books();
                                                member = new Members();
                                                borrow = new Borrow();
                                                book.connection("SELECT
* FROM Books WHERE BookID = " + data[0]);

member.connection("SELECT * FROM Members WHERE MemberID = " + data[1]);
                                                int
numberOfAvailbleBooks = book.getNumberOfAvailbleBooks();
                                                int
numberOfBorrowedBooks = book.getNumberOfBorrowedBooks() - 1;
                                                int numberOfBooks =
member.getNumberOfBooks();
                                                //for checking if there is
no same information in the database
                                                if
(numberOfAvailbleBooks == 0 && numberOfBooks > 0) {

numberOfAvailbleBooks += 1;
                                                        numberOfBooks -=1;

book.update("UPDATE Books SET NumberOfAvailbleBooks =" +
numberOfAvailbleBooks +

",NumberOfBorrowedBooks =" + numberOfBorrowedBooks + ",Availble = true WHERE
BookID =" + data[0]);

member.update("UPDATE Members SET NumberOfBooks =" + numberOfBooks + " WHERE
MemberID =" + data[1]);

borrow.update("DELETE FROM Borrow WHERE BookID =" + data[0] + " AND MemberID
=" + data[1]);
                                                        //for setting the
array of JTextField to null
                                                        clearTextField();
                                                }
                                                else if
(numberOfAvailbleBooks > 0 && numberOfBooks > 0) {

numberOfAvailbleBooks += 1;
                                                        numberOfBooks -=1;

book.update("UPDATE Books SET NumberOfAvailbleBooks =" +
numberOfAvailbleBooks +

",NumberOfBorrowedBooks =" + numberOfBorrowedBooks + " WHERE BookID =" +
data[0]);

member.update("UPDATE Members SET NumberOfBooks =" + numberOfBooks + " WHERE
MemberID =" + data[1]);

borrow.update("DELETE FROM Borrow WHERE BookID =" + data[0] + " AND MemberID
=" + data[1]);
                                                        //for setting the
array of JTextField to null
                                                        clearTextField();
                                                }
```

```
                                                        else

JOptionPane.showMessageDialog(null, "The book is not borrowed", "Warning",
JOptionPane.WARNING_MESSAGE);
                                                }
                                        };
                                        runner.start();
                                }
                                //if there is a missing data, then display Message
Dialog
                                else
                                        JOptionPane.showMessageDialog(null,
"Please, complete the information", "Warning",
JOptionPane.WARNING_MESSAGE);
                        }
                });
                //for adding the action listener for the button to dispose
the frame
                cancelButton.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent ae) {
                                dispose();
                        }
                });
                //for setting the visible to true
                setVisible(true);
                //show the internal frame
                pack();
        }
}
```

Appendix C: List of Publications

1. M. Alawairdhi and H. Yang, “A Business-Logic Based Framework for Evolving Software Systems,” in the 33rd Annual IEEE International Computer Software and Applications Conference (IEEE COMPSA'09), Seattle, Washington, USA, July 2009, pp.300-305.

2. M. Alawairdhi and H. Yang, “A Reengineering Approach for Software Systems Complying with the Utilization of Pervasive Computing Technologies,” in the 3rd International Conference on Complex Distributed Systems (CODS 2009), Leipzig, Germany, March 2009.

3. M. Alawairdhi and H. Yang, “A Reengineering Approach for Software Systems Complying with the Utilization of Pervasive Computing Technologies,” Communications of SIWN, Vol. 6, April 2009, pp. 139-143.

4. M. Alawairdhi, H. Yang, and M. AL-Akhras, "BlueCRM: A New Trend of Customer Relationship Management Systems " in The 12th IEEE International Workshop on Future Trends of Distributed Computing Systems, Kunming, China, 2008, pp. 226-232.

5. M. Alawairdhi and H. Yang,, “Proactive Customer Relationship Management System Using Bluetooth-enabled Devices,” in the EPSRC UK-China Network Workshop, Shandong University, Jinan, China, April 2007. [Presented]

6. M. Alawairdhi, J. Huang and H. Yang, “Collaboration Enabled Smart STRL,” in the 12th Chinese Automation & Computing Society Conference in the UK, Loughborough, England, September 2006.