

University of Southern Queensland

**The Ontological Evaluation of the Requirements  
Model When Shifting From a Traditional To a  
Component-Based Paradigm in Information  
Systems Re-Engineering**

A dissertation submitted by

Raul Valverde M.Eng M.B.A.

For the award of  
Doctor of Business Administration

2008

## ABSTRACT

The vast majority of present legacy information systems were implemented using the traditional paradigm. The traditional paradigm consists of modeling techniques used by system analysts such as System Flow Charts and Data Flow Diagrams (DFD) to capture, during the analysis phase, the activities within a system.

However, with recent developments, particularly trends towards e-Commerce applications, platform independence, reusability of pre-built components, capacity for reconfiguration and higher reliability, many organizations are realizing they will need to re-engineer their systems into new component based systems that meet these trends given the limitations of legacy systems to adapt to these new technical requirements.

There is a high degree of interest and concern in establishing whether or not a full migration to a more portable and scalable component-based architecture will be able to represent the legacy business requirements in the underlying requirements model of the re-engineered information systems.

As a result, this study poses the question: *Is the resulting component-based requirements model ontological equivalent to the legacy requirements model when shifting paradigms in the re-engineering process?*

After a literature review, the research study is justified given the differences in requirements modeling between component-based and traditional paradigms, which give an indication that the resulting component model might not represent the same business requirements represented in the legacy system requirements model.

The study evaluated the requirements models generated by the component-based and traditional approaches when shifting paradigms in the re-engineering process in order to verify that the re-engineered component-based requirements model was capable of representing the same business requirements of the legacy system. Design science and an ontological evaluation using the Bunge-Wand-Weber (BWW) model were the central research methodologies for this study.

A legacy system was selected as part of the case study and re-engineered by using the component-based paradigm with the help of UML diagrams. The requirements model of the legacy system was recovered using reverse engineering and compared to the component-based requirements model using normalized reference models generated with the help of BWW transformation maps. These maps revealed that the re-engineered requirements models were capable of representing the same business requirements of the legacy system. A set of rules was suggested when re-engineering legacy into component-based information systems to ensure the same representation of legacy system's requirements in the re-engineered requirements model.

Finally, this research included directions of future research that put emphasis on the development of automated software tools for systems re-engineering that could implement the rules suggested in this study and the ontological methodology approach used.

## CERTIFICATION OF DISSERTATION

I certify that the ideas, experimental work, results, analyses, software and conclusions reported in this dissertation are entirely my own effort, except where otherwise acknowledged. I also certify that the work is original and has not been previously submitted for any other award, except where otherwise acknowledged.

---

Signature of Candidate

---

Date

## ENDORSEMENT

---

Signature of Supervisor

---

Date

---

Signature of Supervisor

---

Date

## **ACKNOWLEDGMENTS**

I would like to thank the Computer Institute of Concordia University for their support in my research.

I would like to thank the faculty of the Department of MIS and Decision Sciences at Concordia University for their advice in the preparation of this dissertation.

Special thanks also to my Academic Supervisor Professor Mark Toleman for his patience, guidance and insight throughout this research project.

For my parents, Tirso and Esther and my wife France.

## TABLE OF CONTENTS

<b>CHAPTER 1</b>	<b>INTRODUCTION TO THE RESEARCH STUDY</b>	<b>1</b>
1.1	Background and significance of the study	1
1.2	Research question and objective	2
1.3	Justification for the research	3
1.4	Research Methodology	4
1.5	Structure of dissertation	6
1.6	Summary	7
<b>CHAPTER 2</b>	<b>LITERATURE REVIEW</b>	<b>8</b>
2.1	Introduction	8
2.2	Legacy Systems	8
2.3	Requirements model	10
2.3.1	Business processes	11
2.4	Traditional modeling approach	11
2.5	Component Based Modeling	14
2.5.1	UML Component Based modeling approach	15
2.5.2	Catalysis approach to Component Based Modeling	16
2.5.3	UML vs. Catalysis	19
2.6	Traditional Vs. Component-Based Modeling	21
2.7	System Re-engineering	22
2.7.1	The Deursen Methodology for System Re-engineering	24
2.7.2	The Draco Methodology for System Re-engineering	27
2.7.3	The Jacobson and Lindstrom Methodology for System Re-engineering	30
2.8	Ontologies	32
2.8.1	Bunge-Wand-Weber model	34
2.8.2	Ontological Evaluation of Requirements models	39
2.8.3	BWW mapping for process modeling	43
2.8.4	BWW mapping for component based requirements models	45
2.9	Templates for the transformation of requirements models into ontological models	49
2.9.1	Opdahl and Henderson-Sellers template model for enterprise modeling	49
2.9.2	The Green and Rosemann BWW meta-model	54
2.10	Conclusions	56
<b>CHAPTER 3</b>	<b>RESEARCH METHODOLOGY</b>	<b>58</b>
3.1	Introduction	58
3.2	Research outline for this dissertation	59
3.3	Methodologies applied to the Design Science Framework	60
3.4	Justification of the Case Study Research Methodology	62

3.5	Justification of the Jacobson and Lindstrom Methodology for Information Systems Re-engineering and legacy system requirements model recovery .....	63
3.6	Justification of the Fettke and Loos methodology for Ontological Evaluation of Requirements models .....	64
3.7	Justification of the Rosemann and Green meta-models for normalized reference models .....	64
3.8	Procedures .....	65
3.8.1	Data Collection (Build).....	65
3.8.2	Requirements model Recovery (Build).....	67
3.8.3	Component-based requirements model generation (Build).....	69
3.8.4	Ontological Evaluation (Evaluation) .....	71
3.9	Validity and Reliability .....	72
3.10	Ethical Considerations.....	74
3.11	Conclusions.....	75
<b>CHAPTER 4 RESULTS .....</b>		<b>76</b>
4.1	Case Study Description.....	76
4.1.1	Sub-system selection.....	77
4.1.2	Description of the Offer and Application Sub-System .....	77
4.2	Legacy requirements model recovery .....	78
4.2.1	Case Study's System Architecture.....	79
4.2.2	Use cases.....	84
4.2.3	Legacy requirements model construction .....	87
4.3	Re-engineering of legacy systems .....	94
4.4	Ontological Evaluation.....	104
4.4.1	Developing a transformation mapping.....	105
4.4.2	Identifying ontological modeling deficiencies.....	112
4.4.3	Generation of normalized ontological meta-models .....	115
4.5	Conclusions.....	126
<b>CHAPTER 5 ANALYSIS AND CONCLUSIONS.....</b>		<b>127</b>
5.1	Introduction.....	127
5.2	Conclusions on the research issues.....	128
5.2.1	Conclusion to the research issue 1: "Conflict" .....	128
5.2.2	Conclusion to the research issue 2: "Grammar accommodation" .....	132
5.2.3	Conclusion to the research issue 3: "Complementary" .....	133
5.2.4	Conclusions to research question.....	135
5.3	Implications for theory .....	136
5.4	Contributions of this study .....	137
5.5	Implications to Information Systems Practice .....	138
5.6	Limitations .....	138
5.7	Implications for Methodology.....	139
5.8	Directions for future research .....	140

5.9	Summary .....	140
<b>APPENDIX B</b>	<b>USE CASES .....</b>	<b>155</b>
<b>APPENDIX C</b>	<b>EVENT-RESPONSE TABLES.....</b>	<b>166</b>
<b>APPENDIX D</b>	<b>DFDS FOR LEGACY SYSTEM OF CASE STUDY .....</b>	<b>174</b>
<b>APPENDIX E</b>	<b>UML DIAGRAMS FOR RE-ENGINEERED SYSTEM OF CASE STUDY .....</b>	<b>179</b>
<b>APPENDIX F</b>	<b>BWW MODEL.....</b>	<b>190</b>
<b>APPENDIX G</b>	<b>CONSENT LETTER .....</b>	<b>194</b>
<b>APPENDIX H</b>	<b>INTERVIEW PROTOCOL.....</b>	<b>196</b>

## LIST OF FIGURES

Figure 2.1	Forecasts for numbers of programmers (worldwide) and distribution of their Activities .....	9
Figure 2.2	Sample ERD .....	11
Figure 2.3	A sample data flow diagram.....	13
Figure 2.4	Context and functional hierarchy diagrams.....	14
Figure 2.5	An Example Type Model .....	17
Figure 2.6	An Example Domain Model.....	18
Figure 2.7	An Example Component Specification Architecture .....	19
Figure 2.8	Analysis and transformation of legacy systems .....	26
Figure 2.9	A domain-oriented approach .....	27
Figure 2.10	Strategy of Component-Oriented Software Re-engineering using Transformations.....	28
Figure 2.11	Use Case, Sequence Diagram and Class Diagram of the Reprojected System .....	29
Figure 2.12	Preparing an analysis model.....	31
Figure 2.13	UML class diagram of the instantiation level entry .....	50
Figure 2.14	UML class diagram extended to show the class entity.....	51
Figure 2.15	UML class diagram extended to show the property entry.....	52
Figure 2.16	UML class diagram extended to show the ontological descriptions of properties .....	53
Figure 2.17	UML class diagram extended to show the lifetime entry.....	54
Figure 2.18	The application of Green and Rosemann (1998) metamodel for ontologies comparison.....	55
Figure 3.1	Preparation of the analysis model.....	70
Figure 4.1	Legacy System Architecture.....	80
Figure 4.2	On-line interface processing.....	81
Figure 4.3	Batch processing in the legacy system .....	82
Figure 4.4	Screen layout sample for procedural model .....	83
Figure 4.5	Screen flow structure.....	84
Figure 4.6	DFD notation .....	89
Figure 4.7	Context diagram for the Offer and Application sub-system .....	91
Figure 4.8	Offer Figure 4-8 Offer and Application sub-system DFD .....	92
Figure 4.9	Completed E-R data model .....	93
Figure 4.10	Use case model for the case study's sub-system.....	96
Figure 4.11	Subsystems relationships.....	99
Figure 4.12	Interface relationships .....	99
Figure 4.13	Control Package.....	100
Figure 4.14	Process Offer Head Office Sequence Diagram .....	101
Figure 4.15	Loan generation activity diagram.....	102
Figure 4.16	State diagram of the Income test object .....	103
Figure 4.17	Normalized ontological model for the ERD of the legacy system. ....	117
Figure 4.18	Normalized BWW meta-model for the context diagram of the legacy system .....	118
Figure 4.19	Normalized BWW meta-model DFD for the Process Offer Head Office use case.....	119
Figure 4.20	BWW meta-model for the use case diagram for the re-engineered system.....	120



Figure 4.21	BWW model for the sequence diagram of the Process Offer Head Office use case.....	121
Figure 4.22	BWW meta-model for the class diagram of the database package for the process offer by head office use case for the re-engineered system .....	122
Figure 4.23	BWW meta-model for the package diagram for the Process Offer by Head Office use case for the re-engineered system.....	123
Figure 4.24	BWW meta-model for the state diagram of the ProcessOfferHOfficePanel object .....	124
Figure 4.25	BWW meta-model for the activity diagram of the Process Offer Head Office .....	125

## LIST OF TABLES

Table 2.1	Summary of Evaluation of UML vs. Catalysis .....	20
Table 2.2	Summary of Evaluation of Re-engineering methodologies .....	32
Table 2.3	Constructs of the BWW-model .....	37
Table 2.4	BWW interpretation of UML constructs.....	45
Table 2.5	BWW Representation Model Analysis for Dynamic Aspects of UML .....	47
Table 3.1	Research activities based on the Design Science Framework.....	60
Table 3.2	Methodologies used in this research.....	61
Table 3.3	Research methodologies used for the Design Science Framework..	62
Table 4.1	Offer by regional office Use case.....	85
Table 4.2	Possible events for the Process Application use case.....	88
Table 4.3	Sub-system Actors.....	94
Table 4.4	Possible Classes and interfaces for the Sub-System.....	97
Table 4.5	Mapping of Component Analysis models into Description Elements .....	104
Table 4.6	Mapping between traditional and BWW constructs.....	106
Table 4.7	Mapping between UML diagrams and BWW constructs.....	108
Table 4.8	Elements of meta language for Normalized Ontological Models ..	116
Table 5.1	Mapping of ERD into UML Class diagrams.....	129
Table 5.2	Potential conflict in the representation of data flows in the re- engineered component model.....	131
Table 5.3	Mapping of Context Diagrams onto UML Use Case diagrams .....	131
Table 5.4	Traditional diagrams representation in UML component diagrams	133
Table 5.5	UML constructs that complement traditional requirements models .....	134

# Chapter 1 Introduction to the research study

## 1.1 Background and significance of the study

The vast majority of information systems (IS) were implemented in the early days of computing by using the traditional paradigm and implemented with a structured computer language such as COBOL (Longworth 2003). The traditional paradigm defines the programming process as a sequence of commands leading to the manipulation of data to produce a result (Brookshear 2000).

The traditional paradigm consists of modeling techniques such as System Flow Charts and Data Flow Diagrams (DFD) used by system analysts to capture, during the analysis phase, the activities within a system. The strengths of these models provided a means of identifying input, processes and output. These particular models have been used since the early times of computers and were considered, for the most part, the documentation of legacy systems (Longworth 2003).

However, with recent developments, particularly the trends towards e-Commerce applications and platform independence, many companies are realizing they will have to migrate their systems to new improved systems in order to meet these trends as legacy systems are not capable of coping with these new challenges.

The migration of a legacy system to a new target system is a process of re-engineering that requires its examination and alteration to reconstitute it in a new form (Chikofsky & Cross 1990). This new form may result in the need for a shift in the information systems paradigm serving the architecture and the business domain.

Modern computer languages such as Java, offer many advantages in such a re-engineering process; in particular, good web-application development capabilities, platform independence for applications and security. However, Java is an object-oriented language and changing from a procedural to an object-based language represents a fundamental paradigm shift.

Although object technology has become the vogue for re-engineering information systems, many projects regarded as being object-oriented have failed in recent years due to organizational and technical troubles. Wolfgang (1997) mentions some the problems associated with the object-oriented paradigm as: classes/objects implemented in one programming language cannot interoperate with those implemented in other languages, some object-oriented languages require the same compiler version, composition of objects is typically done on the language level and composition support is missing, that is, visual/interactive tools that allow the plugging together of objects.

On the other hand, ten years ago the component-based paradigm was heralded as the next wave to fulfill the technical troubles that object technology could not deliver (Wolfgang 1997). In addition, the component-based paradigm allows a fast delivery of information systems due to its capacity of reconfiguring and reassembling pre-built business components, easy maintainability and higher quality due to the reusability of pre-tested components (Szyperski 1999).

However, regardless of the systems paradigm used in the development of information systems, requirements models need to be created in order to describe the requirements collected by the system analyst (Jacobson et al. 1994). These models can be constructed by using a grammar that provides a set of constructs and rules to represent these requirements (Wand & Weber 2002).

These requirements models can be used as the blueprint for information systems development, however, they can become quite different depending on whether the project team uses the traditional or the component-based paradigm.

Component-based differs from the traditional analysis as it is an approach to model a system around a set of interacting components containing objects. These objects encapsulate data and behavior while the traditional analysis approach maintains a process-oriented view of systems, providing a decomposition based on processes (namely, data flow diagrams) and provides a view that models process and data.

Information systems under the traditional paradigm were developed when the prevalent systems development life-cycle was the 'waterfall' life-cycle while a component-based paradigm almost necessarily implies an iterative and incremental life-cycle (Satzinger, Jackson & Burd 2002).

Within the context of an information systems paradigm shift, the continuity, robustness and integrity of the business processes and functions of the system are of prime concern when re-engineering legacy systems. This means that the requirements model of the re-engineered information system should represent the same business requirements as the original legacy system in order to preserve this integrity.

Although the traditional and component-based approaches have different grammars for representing requirements models, these models can be compared for equivalency of representation of business requirements (Wand & Weber 1993). An evaluation of requirements models would reveal the limitations of representing the legacy system business requirements in the component-based re-engineered model.

Knowing these limitations would be helpful to assist information systems professionals in the development of automated tools for re-engineering information systems and identification of any business rules in case of conflicts. It will also help to evaluate if the re-engineered systems would be able to satisfy the legacy business requirements.

## **1.2 Research question and objective**

During the life-cycle of the information system, changes can occur that would require a change of scope in the information system. One of these changes is the availability of better technology (Whitten, Bentley & Dittman 2000). Change management of information systems is an ongoing function in the life-cycle of the information systems that deal with these changes so they can be prioritized and implemented at optimal times (Whitten, Bentley & Dittman 2000).

If a better technology becomes available, this could generate a new technical requirement that would require a change in the system. As part of the change management, a decision analysis would need to be performed to assess if the new technology would be feasible in the system (Whitten, Bentley & Dittman 2000). If the analysis reveals that the implementation is not feasible, the information systems will require re-engineering to adapt them to the new technology requirement.

This re-engineering of a legacy information system would require a paradigm shift. However, there is a high degree of interest and concern in establishing whether or not a full migration to a more portable and scalable component-based architecture will be able to represent the legacy business requirements in the underlying requirements model of the re-engineered information systems.

The aim of the research therefore becomes an evaluation of the requirements models of the traditional and component-based information systems when re-engineering information systems in order to verify that both models are ontological equivalent and represent the same business requirements.

The main purpose of this research is to investigate the following research question:  
Is the resulting component-based requirements model ontological equivalent to the legacy requirements model when shifting paradigms in the re-engineering process?

This will require answering the following research questions:

- Research issue 1: Are the compared models in conflict?
- Research issue 2: Can the business component model accommodate all the grammar constructs of the legacy requirements model?
- Research issue 3: Are the compared models complementary?

A substantial information systems evolution can be a major concern of any company considering a paradigm shift as this represents the ability of the new information systems to accommodate the company's essential business processes. In addition, there is the possibility that the component requirements model would not only be able to accommodate all the grammar constructs of the legacy requirements model but complement it with more constructs that were not able to be represented in the original requirements models. In short, this research will focus mainly on the evaluation of the re-engineered requirements models and their capability to represent the same business requirements as the legacy requirements models.

### **1.3 Justification for the research**

The proposed study can be justified on the following grounds:

- Differences in requirements modeling between component and traditional based paradigms
- Possible conflicts when representing business requirements in the re-engineered business component model
- The relevance of this evaluation when constructing re-engineering methodologies

- Gaps in literature on information systems re-engineering that this research project is intended to fill
- The costs/benefits of the paradigm shift
- The high value of legacy systems to business.

Although traditional and component-based paradigms have similarities, both have requirements models that might not be ontological equivalent when re-engineering information systems. As these models capture the business requirements, if the re-engineered requirements model is not ontological equivalent to the legacy requirements model, this can have an impact on the functionality of the final system and its ability to satisfy the original business requirements. If requirements are not met with the new system, this will have an impact on the cost of the information systems as future modifications to meet requirements will be more costly than building a system that meets the requirements in the first place.

Further, if conflicts are found when representing requirements in the re-engineered component requirements model, this could help in the development of re-engineering methodologies as business rules can be created in the case that these conflicts are identified.

Little research has been undertaken on the systematic evaluation of requirements models (Fettke & Loos 2003). Requirements model comparison is also acknowledged as an area that has not been explored as a possible application of ontological evaluation of reference models (Fettke & Loos 2003). This observation therefore highlights the relevance of the research presented in this dissertation.

#### **1.4 Research Methodology**

In order to address the research questions, the design-science research methodology is chosen as a framework for the study given the applied nature of the research. This methodology has a history of providing good results in the evaluation of constructs and models in information systems (Hevner et al. 2004). This is in line with Nunamaker and Chen (1990) who classify design science in IS as applied research that applies knowledge to solve practical problems.

Design science in information systems is defined by March and Smith (1995) as an attempt to create things that serve human purposes, as opposed to natural and social sciences, which try to understand reality. March and Smith (1995) identify build and evaluate as the two main research activities in design science. Build refers to the construction of constructs, models, methods and artifacts demonstrating that they can be constructed. Evaluate refers to the development of criteria and the assessment of the output's performance against those criteria. Parallel to these two research activities in design science, March and Smith (1995) add the natural and social science couple, which are theorize and justify. This refers to the construction of theories that explain how or why something happens. In the case of IT and IS research this is often an explanation of how or why an artifact works within its environment. Justify refers to theory proving and requires the gathering of scientific evidence that supports or refutes the theory (March & Smith 1995).

As part of the design evaluation process in the design-science methodology, the case study methodology is chosen to evaluate the capacity of the re-engineered component requirements model for representing the same requirements as the legacy traditional requirements model (Benbasat, Goldstein & Mead 1987). The research will start with a description of the case-study company, its organizational structure, main business services and client base.

The building part of the research will be done by using re-engineering and reverse engineering methodologies that will help to generate the requirements models required for the research. There are many re-engineering methodologies that help to cope with the problem of transforming legacy systems originally developed with traditional methodologies into component-based systems. However, the Jacobson and Lindstrom (1991) approach for re-engineering of legacy systems was chosen for the following reasons:

- It contemplates cases of a complete change of implementation technique and no change in the functionality, which is the case of this research.
- It does not require the use of source code. In the case study used for this research there is no access to the source code used to develop the system.
- It also covers reverse engineering. This is useful for this research given the need to capture the original requirements model for the legacy system.
- It is relatively simple to use.

Although the original methodology was proposed for object-oriented systems, it can be easily adapted for component-based systems as components can be viewed as a higher level of abstraction based on object-oriented methodology.

In order to capture the requirements model of the legacy system, the researcher will apply reverse engineering as specified in the Jacobson and Lindstrom (1991) methodology.

Once the legacy system and re-engineered requirements models are generated as part of the building part of the research, they will be evaluated based on the ontological evaluation of grammars (Wand & Weber 1993). As part of the evaluation research, an analysis will be done using the Bunge-Wand-Weber model (BWW-model). The BWW model is an ontological theory initially developed by Bunge (1977; 1979) and adapted and extended by Wand and Weber (1989; 1995; 1997).

The use of the BWW-model is justified on two grounds. First, the model is well founded on mathematical concepts. Second, prior research on the evaluation of grammars shows that the BWW model has been used successfully in information systems research (Evermann & Wand 2001b; Green & Rosemann 2000; Opdahl & Henderson-Sellers 2002a; Weber & Zhang 1996).

A methodology by Fettke and Loos (2003) is used to compare both legacy and re-engineered requirements models for equivalency of representation of business requirements. The methodology is justified for the following reasons:

- It provides a mechanism for the comparison of requirements models

- Requirements models can be compared based of their normalized referenced models
- Its simplicity

The ontological normalization of a reference model consists of four steps (Fettke & Loos 2003):

- Developing a transformation mapping,
- Identifying ontological modeling deficiencies,
- Transforming the reference model, and
- Assessing the results.

In the first step of this method, it is necessary to develop a transformation mapping for the grammar used for representing the requirements model. This transformation mapping allows converting the constructs of the grammar used to the constructs of the BWW model. The first step is based on the method for the ontological evaluation of grammars proposed by Wand and Weber (1993).

The transformation mapping consists of two mathematical mappings. First, a representation mapping describes whether and how the constructs of the BWW model are mapped onto the grammatical constructs. Second, the interpretation mapping describes whether and how the grammatical constructs are mapped onto the constructs of the BWW model (Fettke & Loos 2003).

In the third step, the reference model will be transformed to an ontological model. The outcome of this step is an ontologically normalized reference model. The objective of both techniques is to represent the domain of interest in a normalized way by applying specific transformation patterns (Fettke & Loos, 2003). The two models will be compared based on their ontologically normalized models. The result of the comparison will be that the compared models are ontological equivalent, complementary or in conflict. These results will justify the theory that will explain how constructs originally represented in traditional requirements models are represented in the re-engineered component models.

In order to generate these normalized reference models in BWW terms, the Rosemann and Green (2002) BWW meta-models will be used. This meta-model is based on the original Entity Relationship (E-R) specification from Chen (1976) with extensions made by Scheer (1998). This version is called the extended ER-model (eERM).

## **1.5 Structure of dissertation**

In the first stage of the dissertation, a comprehensive literature review will isolate the body of knowledge available and identify any additional information gaps (Perry 1998). Identified information gaps will be documented as open research issues (Yin 1994).

After a literature review in chapter 2, the research methodology will be justified and explained in chapter 3. The results of the case study re-engineering and requirements models ontological evaluation will be presented in chapter 4. Finally, a



summary of the study, contribution to theory and the answer to the research question will be presented in chapter 5.

## **1.6 Summary**

The research conducts an evaluation of the requirements models generated by the component-based and traditional approaches when shifting paradigms in the re-engineering process. This evaluation will reveal if the re-engineered requirements model is capable of representing the same business requirements of the legacy system. If this is identified, it can help to evaluate if the re-engineered component-based system will be able to satisfy the original business requirements.

Further, this research will help in the development of re-engineering methodologies as business rules can be created for conflicts identified in the requirements models. The design science research methodology is used for the study. The case study and ontological evaluation using the Bunge-Wand-Weber model are the central evaluation methodologies for this study. The building part of the research is conducted by re-engineering the case study's legacy system and by recovering the requirements model of the legacy system by using reverse engineering that is compared to the component-based requirements model using the Bunge-Wand-Weber model in order to address the research questions.

## Chapter 2 Literature Review

### 2.1 Introduction

First, an overview of the size of the problem of legacy systems is provided in order to understand the justification of the research problem. Second, the requirements model that represents the business requirements of an information system is covered in the context of this research in order to understand better the research objective. Central to establishing any possible consequences of such a paradigm shift are the fundamental differences in the paradigms themselves. In order to understand these differences, traditional and component-based paradigms are discussed and compared as part of this literature review. Additionally, the concepts of system engineering and re-engineering that are central for the research, are covered in this literature review with the comparison and contrast of several re-engineering methodologies. Finally, the Bunge-Wand-Weber Model (Wand & Weber 1989) is discussed in the context of requirements modeling representation and as a tool for requirements model evaluation.

### 2.2 Legacy Systems

Legacy information systems are the brittle, inflexible and poorly understood, yet stable and mission-critical systems that exist in the vast majority of established organizations (Warrel & Stevens 2003). They present risks to their host organization in their current state (primarily strategic business risks associated with their expense and inflexibility), but attempts to modernize them may also be fraught with a range of difficulties (Warrel & Stevens 2003).

Zou and Kontogiannis (2002) suggest some of the reasons why legacy systems are undesirable when they describe legacy systems as “mission critical software systems that are still in operation, but their quality and expected operational life is constantly deteriorating due to prolonged maintenance and technology updates” (p.1). Seacord et al. (2003) claim “software systems become legacy systems when they begin to resist evolution and modification” (p.5). Cormella-Dorda et al. (2000) neatly sum up the legacy system dilemma by drawing an analogy between the legacy information system and the brain (p.1):

*“In many ways, these information systems are to an enterprise what a brain is to the higher species – a complex, poorly understood mass upon which the organism relies for its very existence.”*

Common to most discussions on legacy systems is an acknowledgement of their inflexibility in terms of functionality and integration with other systems, and a “brittleness” introduced by years of maintenance and enhancements (Warrell & Stevens 2003). Also identified are the high levels of expense incurred in order to maintain the legacy applications and the obsolete hardware required to run them. As the systems age, finding individuals with an understanding of the systems and experience with the technologies involved becomes increasingly difficult (Warrell & Stevens 2003).

Despite these problems, legacy systems have two very important advantages:

- They have been in operation for so long that they are very stable
- They are often crucial to the operations of the business

One of the major problems concerning legacy systems is that they count for the vast majority of the current information systems and the size of these systems tends to increase with the years. Indicators provided by Jones (1991) and Deursen et al. (2000) show that the total volume of all software world-wide is  $7 \times 10^9$  function points. The majority of software is written in old, inflexible languages such as COBOL. For example, 80 percent of the mainframe applications are written in COBOL. Erlikh (2000) indicates that there are more than 10,000 large IBM mainframe sites worldwide with 200 billion lines of legacy code still in use.

Moreover, when an industry approaches 50 years of age, it takes more workers to perform maintenance than to build new products (Deursen et al. 2000). Figure 2.1 shows extrapolations for the number of programmers working on new projects, enhancements and repairs. In the current decade, four out of seven programmers are working on enhancement and repair projects. The forecasts predict that by 2020 only one third of all programmers will be working on projects involving the construction of new software. These figures show that maintenance and renovation of existing software, is an activity of major economic importance. Because the total amount of software will only grow, the importance of maintenance and software re-engineering will grow accordingly.

The aim of software re-engineering is to understand, transform and regenerate a legacy system in such a way that its alignment with new business objectives and new technological developments is facilitated (Deursen et al. 2000).

**Figure 2.1 Forecasts for numbers of programmers (worldwide) and distribution of their Activities**

Year	Percentage in New	Percentage in Maintenance
1950	90	10
1960	85	15
1970	65	35
1980	60	40
1990	43	57
2000	40	60
2010	36	64
2020	33	67

(Source: Deursen et al. (2000) p. 2)

Given the size, stability and importance to business operations of legacy systems, it is important to be able to transfer all the requirements that these systems have captured during many years of operation when re-engineering these systems. These

business requirements are captured in requirements models and the re-engineered requirements models should be able to represent the same requirements.

The requirements model is therefore the central part of this dissertation and before more literature review is covered, it is important to understand the nature of requirements models and their use in the information systems context.

### **2.3 Requirements model**

A useful way to define the business domain is by the use of a requirements model. Hoffman (1997) defines a requirements model as the requirements set of a business. It provides the broad view or perspective necessary to identify solutions. The requirements model can be thought of as a representation of how a business functions and works. The model needs to be created in such a way that it can be productively used to simulate the real world business (Hyperion Solutions Corporation 2001). A thorough requirements model should provide a complete end-to-end view of the business processes, from initial ideas and reasons behind the processes to their final implementation. Such a model will hold information about the business goals, geographic locations, organizational structure, parameters, activities, time, cost and resources involved in the business. This information needs to be centrally stored and accessible to all stakeholders in the business. This business documentation will contribute to the understanding of the what, why and how of the functions of a business. Many stakeholders in a business do not have this understanding largely due to the absence of a thorough requirements model (Bloor Research 2001).

The requirements model can help eliminate gaps between the strategic vision of the company and the day-to-day operational execution, between the business and the information systems department, as well as individual duties and organizational requirements (Wreden 1998).

Many legacy systems are poorly documented (Bennett 1995). A poor understanding of the functionality of the legacy system will result in an inaccurate specification of requirements for the target system (Bisbal et al. 1999). Consulting a well maintained requirements model can assist in the understanding of the intended functionality of the legacy system.

During the development of an information system, requirements models are created in order to describe the requirements (Jacobson et al. 1993) collected by the system analyst. These models help to communicate the complexity of the system to the development team members (Satzinger, Jackson & Burd 2002) and to document what was done for the future maintenance or enhancement of it.

These requirements models for a new system created during the analysis phase become quite different depending on whether the project team uses traditional approaches or the more contemporary component-based approach. The primary purpose of the next sections is to review the traditional and component-based modeling techniques, advantages and disadvantages of these approaches and the comparison of the modeling of a real world simplified business process using both techniques.

### 2.3.1 Business processes

The description of the business process, business events and responses is essential in recovering the requirements model (Whitten et. al 2000). One of the most popular and successful approaches for documenting business processes, events and responses is a technique called use cases developed by Dr. Ivar Jacobson (Jacobson et al. 1993). Use cases describe the business process, and document how the business works and the business goals of each interaction with the system. These use cases are then extended to show how the system will support the business goals. Use cases are not just useful to document business processes, they can also be used to generate the target component-based requirements model.

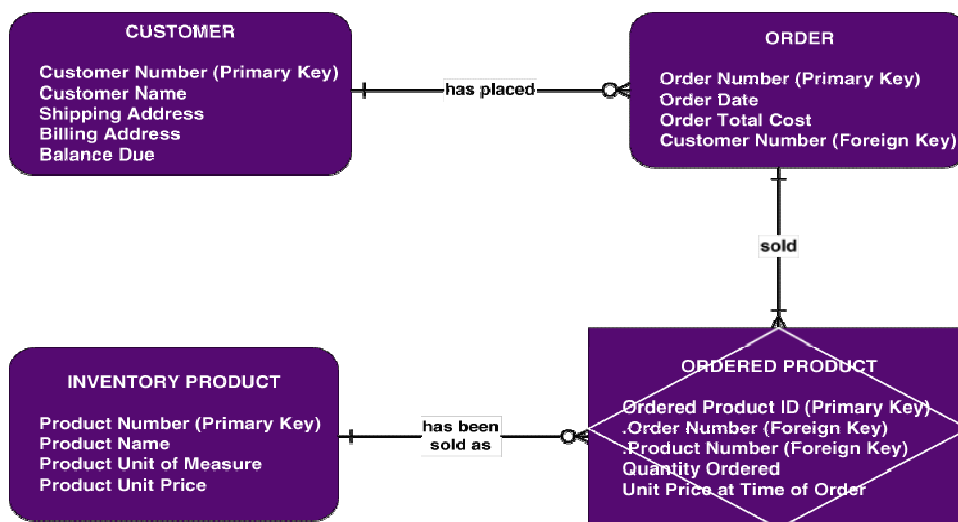
The interactions within the use case should be contained, initiated and seen through to completion by an actor. The use case should further result in achieving a business goal and leaving the system in a stable state (Reed 2002). The nature of a use case is to define the "what" of a system. As such, representing the use cases is essential to the recovery of the legacy system requirements model.

### 2.4 Traditional modeling approach

The traditional methodologies develop two separate models in order to describe information systems: a data model and a process model of the organization's data.

Data models are used for organizing and documenting a system's data (Whitten et. al 2000). The actual model is frequently called an entity relationship diagram (ERD) because it depicts data in terms of the entities and relationships described by the data (Figure 2.2).

Figure 2.2 Sample ERD



(Source: Whitten et. al 2000 p. 258)

An entity is a class of persons, places, objects, events, or concepts about which we need to capture and store data. An attribute is a descriptive property or characteristic of an entity.

A relationship is a natural business association that exists between one or more entities. The relationship may represent an event that links the entities or merely a logical affinity that exists between the entities. Cardinality defines the minimum and maximum number of occurrences of one entity that may be related to a single occurrence of the other entity. Because all relationships are bi-directional, cardinality must be defined in both directions for every relationship.

The traditional approach to information systems development describes activities as processes carried out by people or computers. These processes can be modeled using data flow diagrams (DFD) that help to organize and document the structure and flow of data through a system's processes, and/or the logic, policies, and procedures to be implemented by a system's processes (Whitten et. al 2000). A sample DFD is provided as Figure 2-3.

A process in a DFD is work performed on, or in response to, incoming data flows or conditions (Whitten et. al 2000).

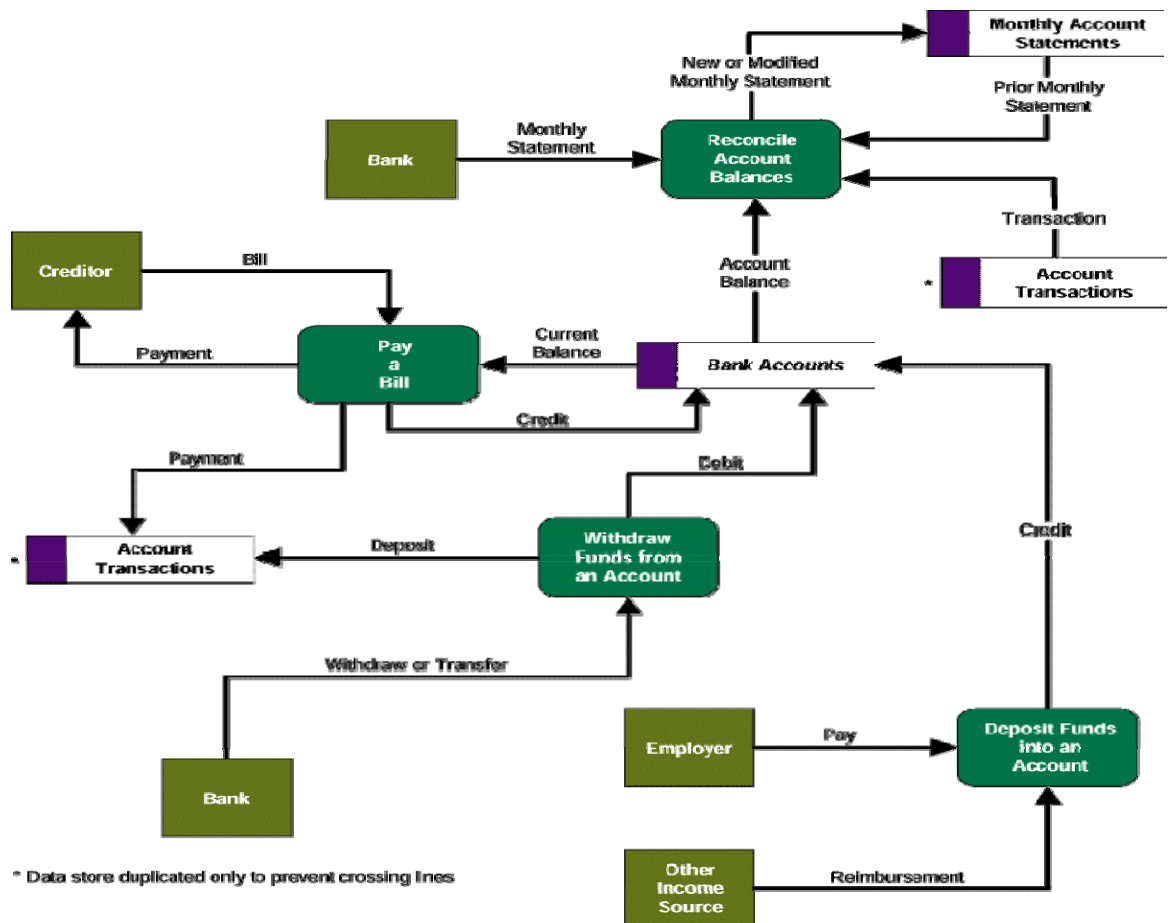
The simplest process model of a system is based on inputs, outputs, and the system itself – viewed as a process. The process symbol defines the boundary of the system. The system is inside the boundary; the environment is outside that boundary. The system exchanges inputs and outputs with its environment.

An external agent defines a person, organization unit, other system, or other organization that lies outside the scope of the project, but which interacts with the system being studied (Whitten et. al 2000). External agents provide the net inputs into a system, and receive net outputs from a system (Satzinger, Jackson & Burd 2002). External agents on a logical data flow diagram may include people, business units, other internal systems or external organizations with which the system must interact.

Whitten et al. (2000) propose several types of DFDs as part of the traditional paradigm. A context diagram defines the scope and boundary for the system and project.

A functional decomposition diagram is required to partition the system into subsystems. It shows the top-down functional decomposition or structure of a system. It provides the beginnings of an outline for drawing data flow diagrams. Figure 2-4 shows a context diagram and a functional decomposition diagram.

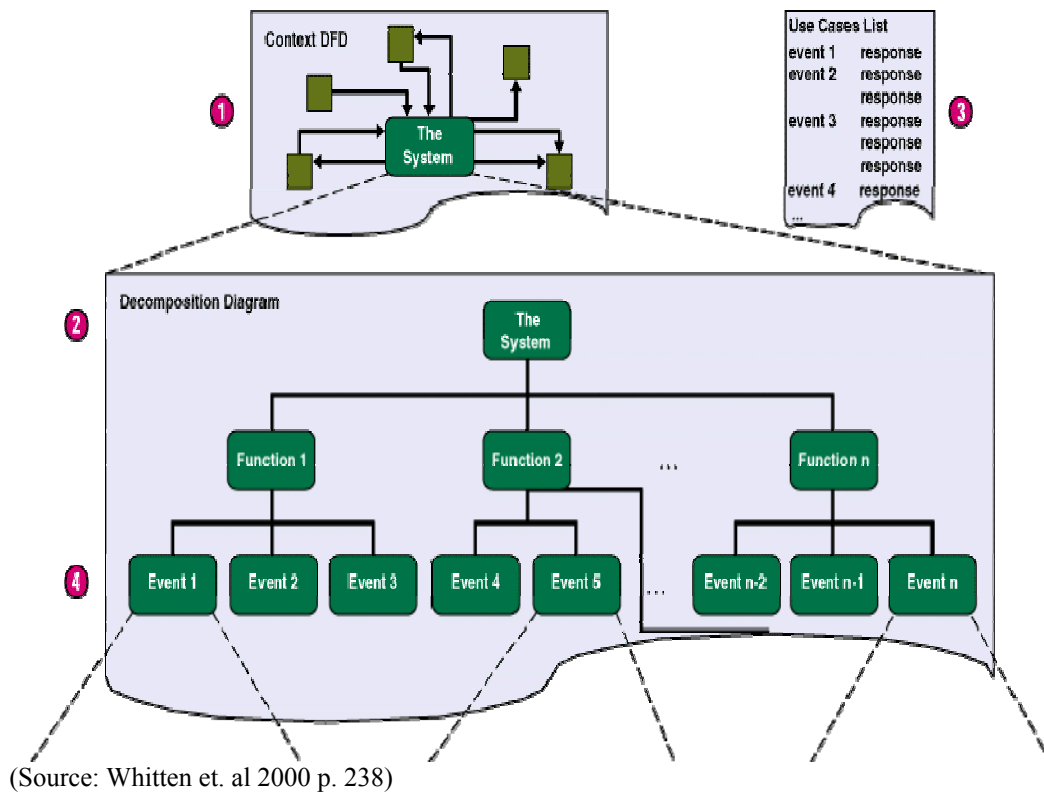
Figure 2.3 A sample data flow diagram



(Source: Whitten et. al 2000 p. 213)

The traditional approach has the advantage of being easy to use and understand. Also, because these models are process oriented, they are suitable for implementation with structured languages such as COBOL, the dominant programming language of legacy systems. The traditional approach was the natural transition to legacy systems implementation. However, current component-based languages such as Microsoft .Net require the description of business processes in terms of components and objects in order to facilitate the implementation and maintenance of information systems. In the next section, the component-based requirements modeling paradigm will be discussed in detail.

**Figure 2.4 Context and functional hierarchy diagrams**



## 2.5 Component Based Modeling

Component-based development focuses on the use of encapsulated and pluggable components as units of deployable code (Szyperski 1998). Before the implementation of component-based software, component models need to be defined in order to specify the behavior, construction and interaction between the different components in the system.

Component models operate at two levels. First, a component model defines how to construct an individual component. For example, Microsoft's Component Object Model (COM) requires each COM component to provide an IUnknown interface. Second, a component model can enforce global behavior on how a set of components in a component-based system will communicate and interact with each other. A component model enables composition by defining an interaction standard that promotes unambiguously specified interfaces (Councill & Heineman 2001). Because components are known almost exclusively by their interfaces these interfaces must be clearly specified, trusted and useful (Digree 1998). The focus of this dissertation is on models that operate at this second level.

Component models are of vital importance when constructing and integrating components with other systems. D'Souza and Wills (1999) observe that "plug-in compatibility" only succeeds if a component can accurately declare its expectations of the other component to which it is connected in its component model.

Houston and Norris (2001) differentiate between the logical and implementation models. The logical representation of a component is concerned with its logical



abstraction, its representation, its relationship with other logical elements, and its assigned responsibilities. The implementation representation of a component defines how its logical representation is implemented in the chosen environment (Houston & Norris 2001).

The logical component model is in fact the requirements model of the system as it captures the business rules of the system without any concern about physical implementation details such as programming languages or deployment issues. Brown (2000, p. 160) mentions that two component-based modeling approaches stand out as offering the most to say about interface-based techniques and their use in building component based systems: the Unified Modeling Language (UML) and Catalysis.

### **2.5.1 UML Component Based modeling approach**

UML is a notation for describing software systems founded on an underlying set of concepts and techniques for developing software-intensive systems. The logical representation of a component is modeled using a UML subsystem, which can be thought of as the design view of a component (Houston & Norris 2001).

In the UML meta-model a subsystem is shown as a subtype of a UML classifier. Therefore, it can realize interfaces as well as have its own operations; these together define the subsystem's specification (Houston & Norris 2001).

A subsystem is modeled as a UML package annotated with the special keyword <<subsystem>>. Packages are containers to organize and manage object model elements, such as classes (Yun-Tung 2001). In comparison to an object-oriented approach, a component package includes definitions of interfaces it requires as well those it provides (Henderson-Sellers 2001).

As for the type of UML diagrams required to prepare a component model, UML does not impose a set of specific diagrams that are required to specify components. However, there are a few recommendations made by several authors for UML diagrams for component modeling. Houston and Norris (2001) propose using the following UML diagrams for specific reasons:

- Class diagrams show the major relationships between internal subsystem elements as well as between other subsystems or packages.
- Statecharts or activity diagrams show important behavioral aspects of the subsystem as a whole.
- Interaction (that is sequence or collaboration) diagrams show how the subsystem elements implement the major interface operations.

A slightly different view is provided by Brown (2000) who justifies the use of the following UML diagrams for component modeling:

- Use case diagrams define how objects outside of a system (called actors) interact with the system's intended functions (called use cases).
- Sequence diagrams show interactions among objects to affect a desired operation or result.

- Class diagrams show the static structure of the system via its interfaces, classes, their internal structure, and their relationship via its interfaces, classes, their interface structure, and their relationship with each other.
- Component diagrams show the organization and dependencies among a set of components. These components are physical elements of a system, including source code, binary code, or executable files.
- Deployment diagrams show the deployment of the physical components to execute on particular nodes in the system considered.

One of the major differences between the Houston and Norris (2001) and Brown (2000) proposed UML diagram set for component modeling is their view of representation of components in UML diagrams. While Houston and Norris (2001) identify logical components as subsystem UML packages in class diagrams, Brown (2000) suggests that components should be represented in component diagrams. Although component diagrams are more suitable for representing component interaction, they clearly represent physical components implemented as source code, binary code or executables. The idea behind requirements models is to model requirements from the logical point of view and free of any physical aspect of the implementation. The idea of subsystems as logical components included in class diagrams seems to be closer to the concept of requirements modeling. The same can be said about deployment diagrams that specify the deployment of physical components.

On the other hand, Houston and Norris (2001) did not include use case diagrams as part of the component model. Use case diagrams are clearly part of the requirements model as they help to visualize the interaction of the system with the external world and this is crucial in representing business requirements. Houston and Norris (2001) included activity diagrams as part of the component representation while Brown's (2000) list did not. Although the object interaction in activity diagrams can be modeled with interaction diagrams, activity diagrams model internal object processes that are not possible to represent in interaction diagrams.

In the next section, the Catalysis approach is discussed as a possible alternative for this study.

### **2.5.2 Catalysis approach to Component Based Modeling**

The Catalysis approach developed by D'Souza and Wills (1999) to component requirements modeling has the following goals:

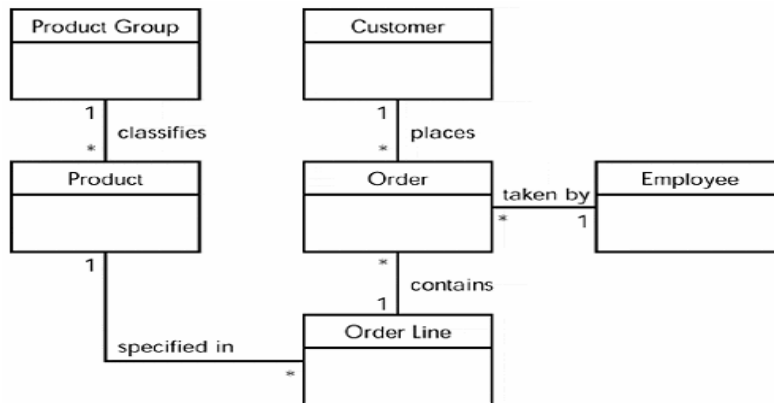
- Systems can be modeled as collections of interacting components.
- System behavior can be analyzed in terms of component interfaces.
- Component specifications can be described independently of the components' implementations.
- A precise, formal notation is available for describing component specifications, sufficient for rigorous analysis of those specifications against a user's needs.

In the Catalysis approach, a user describes the static behavior within a domain as a set of related types within a type model, an abstraction of the UML class concept

(Brown 2000). The structural relationships among types represent the static constraints that exist among elements of the domain (Brown 2000).

This is illustrated in Figure 2-5 where the boxes represent types and the arcs represent static type relationships. Unlike the traditional use of class diagrams for describing the design of a single implementation, in Figure 2-5 the model captures constraints that must be true in any conforming implementation (Brown 2000).

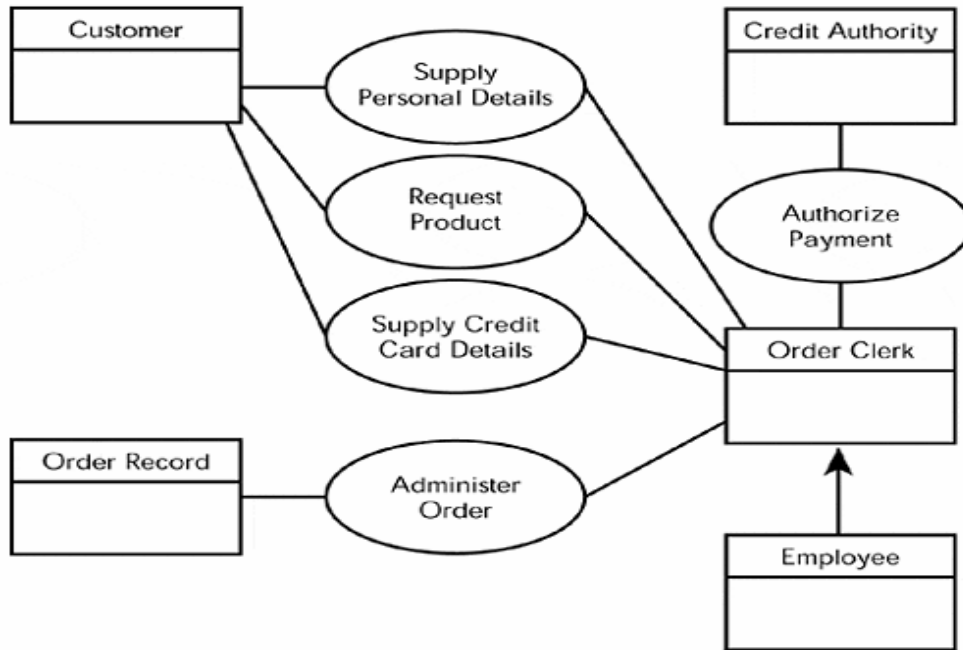
**Figure 2.5 An Example Type Model**



(Source: Brown 2000 p. 174)

The dynamic behavior in a domain can be modeled in Catalysis as interactions among types, and recorded as type collaborations (an abstraction of the UML (class) collaboration concept). Changes of state in a domain occur through interactions among behavior bearing types in that domain. These interactions are represented as collaborations in which types play roles to initiate or respond to requests to carry out actions. This is illustrated in Figure 2-6 where the boxes represent roles, and the ellipses represent joint actions among identified roles (Brown 2000).

Figure 2.6 An Example Domain Model

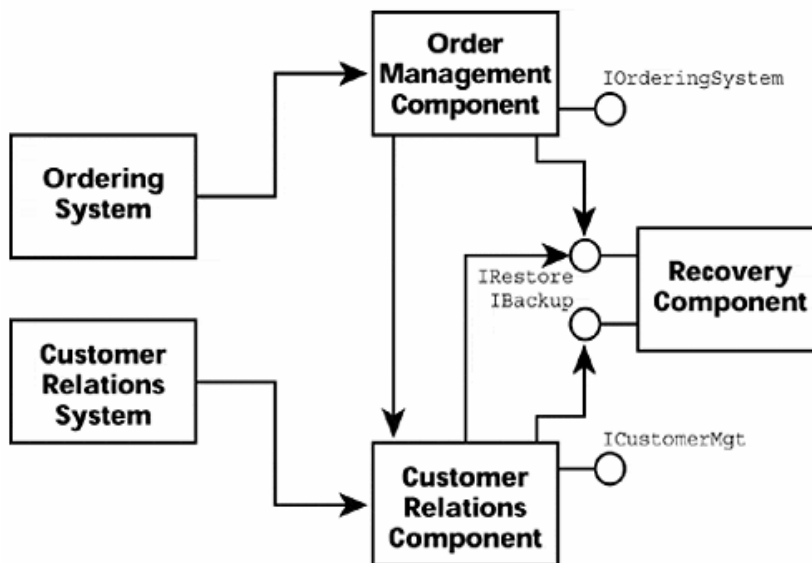


(Source: Brown 2000 p. 175)

For each type in a domain the user continues by describing its features (attributes and operations) in detail. Particularly important are the pre- and post-conditions that define the semantics of each operation by describing the state that must exist before the operation can take place, and the state that will result having executed the operation. Informal definitions of the pre- and post-conditions can be given. In Catalysis an interface type model is used to define the concepts and constraints referenced by the conditions, and any interface-wide constraints.

Components have dependencies based on interactions among their constituent interfaces. The collection of components and their dependencies can be viewed as a component specification architecture for the application. A component specification architecture identifies all the design constraints in that software structure. This is in contrast to the UML form of component dependency diagrams which show physical dependencies among components such as executables, libraries, tables, and files. A component specification diagram is illustrated in Figure 2-7 where the boxes denote component specifications with attached interfaces shown as “lollipops”. Dashed arrows between boxes or lollipops indicate dependencies.

**Figure 2.7 An Example Component Specification Architecture**



(Source: Brown 2000 p.176)

### 2.5.3 UML vs. Catalysis

UML has strengths when modeling component-based systems. The major one is that it allows modeling the logical representation or requirements model of the component-based system. It also gives the ability to document the context in which component interfaces are used, as well as how those interfaces are realized internally.

However, there are difficulties in using or adopting UML's notions of aggregation and composition (Henderson-Sellers & Barbier 1999). A major weakness is that UML does not propose any specific modeling approach for components and the choice of diagrams is left to the designer. Another limitation is the fact that subsystems as they currently stand in UML are not required to completely encapsulate their contents and do not provide any formal enforcement of outgoing interfaces which is required to represent components in the requirements model.

Interface behavior can be difficult to model in UML as it does not support the definition of interface behavior at the required level of precision (Brown 2000). To model interfaces, component specifications, and other key component concepts, the user is required to make use of extension mechanisms in UML, such as stereotypes and naming conventions (Brown 2000).

In spite of UML difficulties with component modeling, Houston and Norris (2001) mentioned that there is a proposal to improve UML and introduce the semantics that will allow UML to be more precise in both external definitions of an interface's responsibilities and internal definition of its realization within a subsystem. However, this is still not included in UML version 2.0.

UML 2.0 proposal defines new user level constructs that will improve UML support for the component-based development, architectural specifications, and advanced

behavioral modeling techniques using interactions, state machines and activity diagrams.

On the other hand, the Catalysis method defines a behavioral approach to component modeling that allows the behavior within a domain to be more accurately described and then partitioned into appropriate components offering access to behavior through well defined interfaces (Brown 2000).

However, there are some problems with the Catalysis approach. Unlike UML, Catalysis requires a detailed description of the attributes and operations of each type. This could be difficult to establish at the requirements model level because this level of detail is normally not so clear when defining business requirements.

In this section, strengths and weakness of the UML and Catalysis support for component modeling were discussed as well as the UML 2.0 proposal that will enhance UML’s ability to model components by improving support for component-based development, architectural specifications, and advanced behavioral modeling techniques.

In conclusion, business component models can be modeled either by using the UML or Catalysis approach. By using the UML methodology, UML subsystems and their associated interfaces are used to model the logical representation of a component. Different types of UML collaboration diagrams are used for documenting the internal business rules of the components. With the Catalysis method, the component requirements models define a behavioral approach to component modeling that allows the behavior within a domain to be more accurately described and then partitioned into appropriate components offering access to behavior through well-defined interfaces. For this research, UML is used given the availability of software tools for this grammar and its ability to represent better requirements models. A summary of the evaluation of UML versus Catalysis is presented in table 2-1.

**Table 2.1 Summary of Evaluation of UML vs. Catalysis**

UML	Catalysis
Allows modeling of requirements of component-based systems	Allows the definition of behaviors of components and interfaces
Simple consistent approach	Difficult to use to model business requirements
Supported by a large variety of commercial software tools	Supported by a limited amount of commercial software tools

## 2.6 Traditional Vs. Component-Based Modeling

The difference between the traditional approach and the component-based approach to systems development is not in the phases of the Systems Development Life Cycle (SDLC), but in the set of models used. The SDLC for each approach has the same phases: planning, analysis, design, and implementation. The activities within the phases are also the same. The traditional approach focuses on identifying and modeling processes, while the component-based approach emphasizes components and their interactions. As the individual tasks within the activities are focused on building models, they will be different. For example, instead of building DFDs, the analysts build UML interaction diagrams (Houston & Norris 2001) or Catalysis type model and domain model diagrams (Brown 2000).

The ERD and the business type model (Catalysis) or class diagram (UML) use different notations but share many concepts. As entities in the ERD, types (Catalysis) and classes (UML) also have attributes, relationships and cardinalities. Although many ideas are shared, the main difference between the two approaches is that the diagrams, or models, used to represent the processes or system activities are quite different.

The traditional and component-based approaches to system development differ in what happens when an event occurs. The traditional approach views a system as a collection of processes, some done by people and some done by computers. Computer processes are much like conventional computer programs – they contain instructions that execute in a sequence. When the process executes, it interacts with data, reading data values and writing other data values back to the data file. The process might also interact with people, such as when an instruction asks the user to input a value or it displays information to the user on the computer screen (Satzinger, Jackson & Burd 2002).

The traditional approach to systems, then, involves processes, data, inputs, and outputs. When modeling what the system does in response to an event, the traditional approach includes processing models that emphasize these elements (Satzinger, Jackson & Burd 2002). In contrast, the component-based approach views a system as a collection of interacting components. Components have interfaces that allow them to interact with each other and with people using the system. Component instances may contain multiple object instances that can communicate between each other by sending messages. There are not conventional computer processes or data files per se. Components carry out the activities and remember the data values. When modeling what the system does in response to an event, the component-based approach includes models that show components, their interface, and their objects' interactions.

Although data models are encapsulated in business component models and ERDs, the difference with the component-based approach is that component models also represent the appropriate behavior that is associated with the data (Carey & Carlson 2001). The traditional approach maintains a process-oriented view of systems, providing a decomposition based on processes (namely, data flow diagrams), whereas the component-based approach decomposes the problem domain based on a classification of entities (types in Catalysis or classes in UML).

The identification of events and things are the starting points in the modeling process for both approaches. The traditional approach takes the event table and creates a set of data flow diagrams (DFDs) based on these events and the data used and produced by the event, including the context diagram, DFD fragments and detailed DFDs (Satzinger, Jackson & Burd 2002). The entity-relationship diagram (ERD) defines the data storage requirements that are included in the DFDs. The component-based approach takes the event table and creates a series of use case diagrams. The type model (Catalysis) or class diagram (UML) and use case diagrams are used to create additional models and component behavior, including sequence diagrams, interface diagrams, and other models.

In the traditional approach, inputs and outputs are shown as data flows on the context diagram, system diagram and primitive diagrams. In the component-based approach, inputs and outputs are defined by messages entering or leaving the system; this message exchange can be seen in the interaction diagram and use case diagram.

Structured approaches were developed when the prevalent systems development life-cycle was the 'waterfall' life-cycle (Satzinger, Jackson & Burd 2002) while the component-based approach almost necessarily implies an iterative and incremental life-cycle (D'Souza & Wills 1999).

Although both traditional and component-based approaches have similarities, the first is poorly suited to the requirements of component-based systems because it offers little in the way of techniques and guidance for defining and using interfaces as key design abstractions (Brown 2000).

Traditional methodologies are not comprehensive enough to accurately model large and complex systems. Also, businesses are increasingly incorporating third-party components as an integral part of solutions as vendors and businesses work in increasing cooperation. This movement has its ultimate expression in the form of the virtual enterprise and underlines the need for a component-based approach. The trend towards the use of third-party software components in the development of Information Systems is supported by the work of Hawthorne and Perry (2005) and Desmet et al. (2003).

Traditional methodologies are notoriously weak in the key area of the software process. A clearly defined process, adaptable to both solution and component development can be difficult to model with traditional methods.

The next part of the review will discuss the re-engineering process in the context of component-based systems.

## **2.7 System Re-engineering**

In this section, the background of systems re-engineering will be covered and the main tendencies will be discussed in order to understand the role of re-engineering in legacy system renovation.



Sommerville (2001) recognizes that one of the most difficult aspects of re-engineering a legacy system is that the legacy systems are often not structured in a way that identification and separation of the basic architectural components is made possible. The user interface logic, business service logic and data access logic are often intermingled. In response to these problems, amongst others, a number of solutions have been proposed. These solutions can be classified into three categories, re-development, wrapping, and migration (Bisbal et al. 1999).

Re-development involves redeveloping a legacy system from scratch using a more modern architecture with tools and databases all operating on a new platform. This approach may be suitable for companies seeking platform independence for their systems. In such a case, the whole application needs to be redesigned to meet new platform independent requirements. The only transferable part of the redevelopment is the requirements model.

For many companies, the re-development of their legacy systems is not an option due to high risk and cost (Bisbal et al. 1999). As a result, many solutions adopt the concept of wrapping. Wrapping involves surrounding the various components of the legacy system such as data, programs and interfaces so that external clients can re-use these trusted components through a middleware layer. The wrapped system acts as a server for external clients to re-use the core components without needing to know how the service is implemented (Bisbal et al. 1999). A popular implementation of wrapping is called screen scraping which involves the replacing of character-based front ends of legacy systems with a client-based Graphical User Interface. Screen scraping does not however solve many of the original problems inherent in legacy systems (Wu et al. 1997).

The third solution involves the migration of a legacy system to an open system. This solution involves the moving of a system to a new platform while retaining the functionality of the system and causing as little disruption to the operational and business environment as possible. The migration is concerned with the cutover from the old system to the target system. Two main methodologies are dominant within legacy system migration, the Chicken Little methodology (Brodie 1995, cited by Wu et al. 1997) and the Butterfly methodology developed by the Milestone project (Wu et al. 1997).

The Chicken Little Methodology aims to allow the legacy and target systems to interoperate (especially in terms of data access and manipulation) during the migration process (Wu et al. 1997). This interoperation is made possible by a series of gateways. A gateway in this context is a software module designed to mediate between the two systems components. In this way, legacy applications are gradually rebuilt onto the target platform (Bisbal et al. 1999). The Butterfly Methodology does not need the two systems to be interoperable during the life of the migration process. Instead, this methodology employs an engineered data migration approach to ensure the development of the target system is totally separate from the migration of the data.

Irrespective of the methodology used, Bisbal et al. (1999) recognize the migration of a system should involve the following phases:

- Justification for the new system
- Understanding of the legacy system
- Development of the target system
- Testing
- Migration

These re-engineering phases form the backbone of many of the case studies presented in the literature documenting migrations. Works include a study by Babiker et al. (1997) who developed a model and method to reengineer non-object-oriented systems into an object-oriented architecture. The model was judged effective when used to migrate a legacy system to an object-oriented system. Part of the model involves the merging of existing business requirements with new requirements. Another example is a study carried out by Serrano et al. (2001). In this study, a re-engineering environment was developed to assist with the migration of a legacy system to a distributed object environment.

Although there have been many attempts to create methodologies to migrate legacy systems into object-oriented systems, very little research has been conducted to develop methodologies that would help information systems professionals to migrate legacy systems into component-based information systems. Although most of the object-oriented re-engineering methodologies can be adapted to component-based, the latter requires the creation of components and interfaces that sometimes are not so evident when re-engineering legacy systems.

Software re-engineering techniques for component-based software must be based both on static and dynamic information (Fevre et al. 2003). Static information must be extracted from a wide range of sources, including source code, but also configuration files, deployment descriptors, etc. Once the problem of information extraction is solved, almost all techniques available in software re-engineering could be reviewed and adapted to the context of component-based software (Fevre et al. 2003).

Although most of the object-oriented re-engineering methodologies can be adapted for component-based systems, specific component-based software re-engineering transformations start from traditional software and produce component-based entities. In this case, much of the work done on traditional software could probably be reused after some adaptation, because only the target of transformation changes. For instance, a large body of work in recent years involved methods to discover/recover "components" (Koschke 2000). These units of functionality could be wrapped into components as defined by component models.

In the next sections, specific examples of different re-engineering methodologies will be covered and analyzed.

### **2.7.1 The Deursen Methodology for System Re-engineering**

One component-based specific re-engineering methodology is that developed by Deursen et al. (1999). This methodology seems to be one of the few that deals with the transformation of legacy systems originally programmed by using the traditional

approach into component-based systems. In this section, a detailed description and discussion of this approach will be covered.

Deursen et al. (2000) propose three technical approaches to software re-engineering:

1. Analysis of legacy sources: the goal is to inspect the sources of the legacy system and extract information to reveal their structure, purpose and architecture. Analysis is non-intrusive as the legacy sources are only inspected and not modified.
2. Transformation of legacy sources: the goal is to systematically restructure and improve the sources of the legacy system. Transformation is intrusive because the legacy sources are modified.
3. Generation: the goal is to identify potential reusable assets in the legacy system by explicitly introducing and structuring domain knowledge in such a way that major parts of the legacy system can be regenerated. Generation is also intrusive because (parts of) the legacy sources are replaced by generated code.

Analysis and transformation of legacy systems are closely related as shown in Figure 2-8. By reference to the figure, it is possible to appreciate that the analysis step gathers information that can be used for transformation. The goal of analysis is to extract information from legacy systems that reveals their structure, purpose, and architecture.

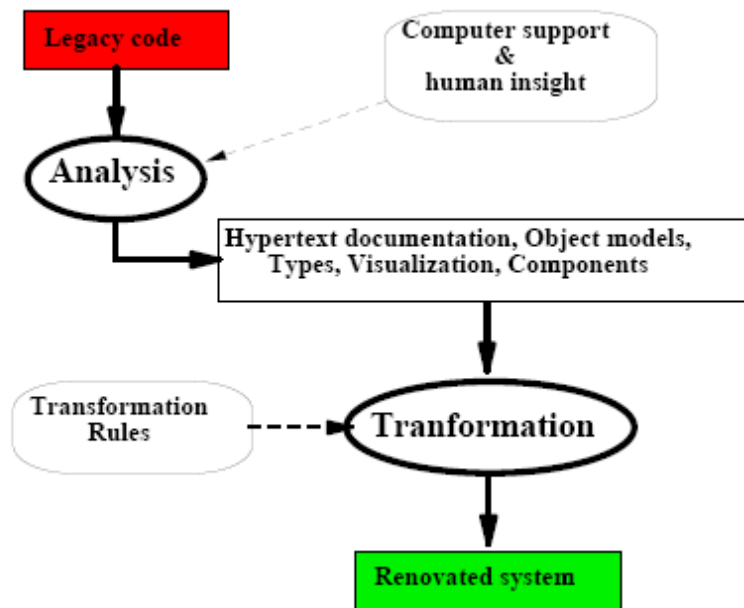
Transformation techniques perform intrusive, systematic modifications of the legacy system in order to enable their maintenance and increase their flexibility (Deursen et al. 2000).

With the insights gained by applying analysis techniques, Figure 2-8 shows that the source code of the legacy system will be transformed in order to achieve the following objectives (Deursen et al. 2000):

- Globally restructure the whole system;
- Restructure the code of individual components;
- Apply uniform comment conventions;
- Eliminate obsolete language features;
- Convert to a new language version;
- Translate to another language.

Generation based on domain engineering is a higher level approach that uses information from the analysis phase and can exploit transformation techniques to achieve its goals.

Figure 2.8 Analysis and transformation of legacy systems

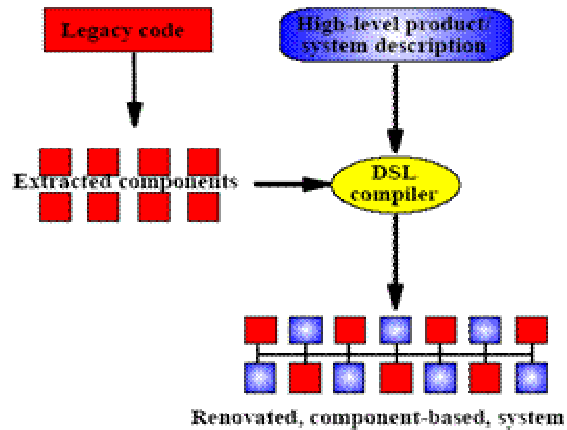


(Source: Deursen et al. 2000 p.12)

The output of transformation is the renovated system; the most sophisticated technical approach to renovation is to decompose the legacy system into components in such a way that these components become reusable across different applications. Customized versions of these components can then be used in different configurations. In order to achieve these goals a deeper understanding of the application domain is needed. Domain engineering attempts to distill domain knowledge from legacy systems (Deursen et al. 2000).

First, the legacy system has to be reorganized to provide the domain knowledge at the proper level of abstraction. Second, a notation tailored towards the domain—a Domain-Specific Language (DSL)—has to be provided to enable the easy composition of components into a workable system. Finally, as shown in Figure 2-9, the DSL will be used as input for a generator (the DSL compiler) (Deursen et al. 2000).

**Figure 2.9 A domain-oriented approach**



(Source: Deursen et al. 2000)

One of the strengths of the Deursen et al. (2000) methodology is the analysis of legacy sources. With the help of this analysis, it is possible to recover the requirements model of a legacy system from its source code. This is valuable when reverse engineering requirements models from information systems as these are normally not kept up to date by legacy system owners. On the other hand, one weakness of the Deursen et al. (2000) methodology is the use of concept and cluster analysis for object identification in legacy source code analysis. These methods besides being difficult to use, can only generate classes if used by a software engineer with knowledge of the application domain and the legacy system. This could be a limitation because the results depend on a sound knowledge of the legacy system.

Another important limitation is its lack of component requirements models. Although the methodology helps to identify components for the target re-engineered system, it does not deal with the generation of the re-engineered system's requirements models required for this research.

An important transformation system that has been used in this area is Draco (Fontanette et al. 2002). Usually, a transformation system restructures a program A into a program B, applying a set of well-defined transformations preserving the semantics of A in B.

According to Prado (1992), it is possible to do the software reconstruction by the direct “load” of language source code to languages of other domains. This methodology will be discussed in more detail in the next section.

### **2.7.2 The Draco Methodology for System Re-engineering**

The strategy of the Draco approach (Fontanette et al. 2002) for component-based software re-engineering using transformations is accomplished in four steps as shown in Figure 2-10:

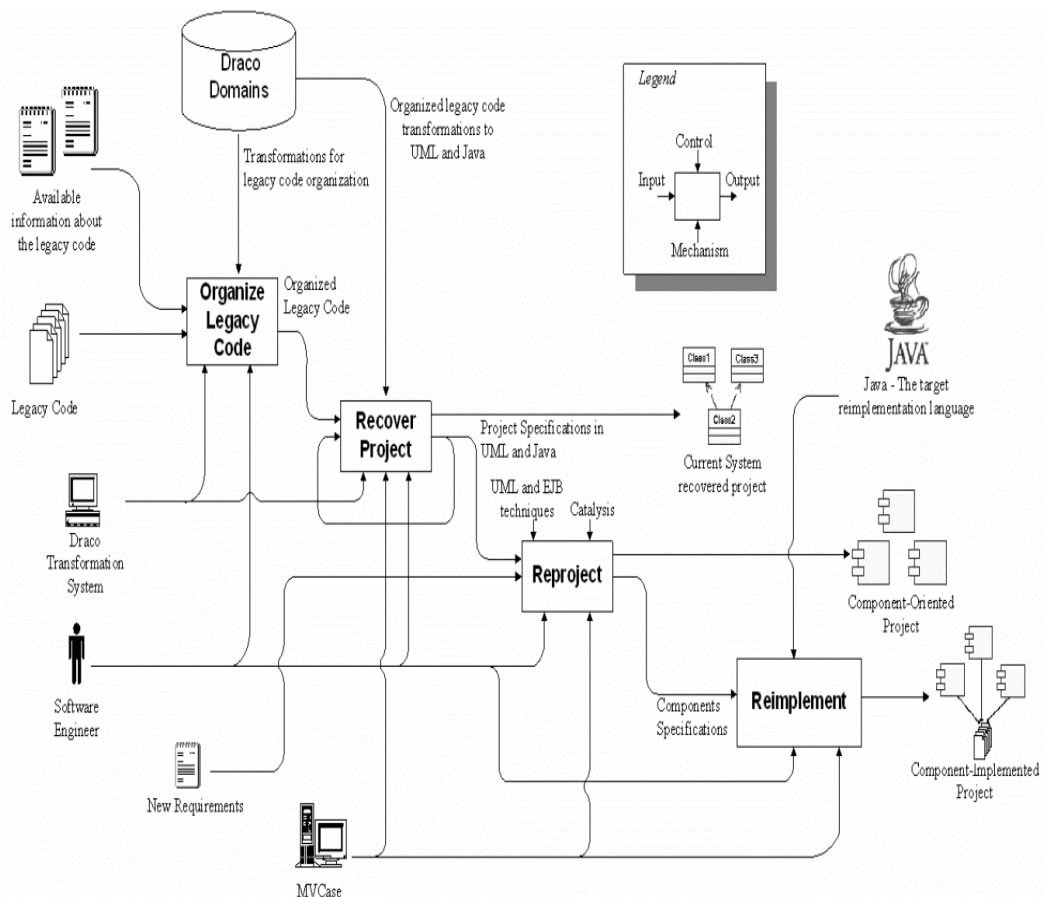
1. Organize Legacy Code

2. Recover Project
3. Reproject
4. Reimplement.

In the first step, Organize Legacy Code, the software engineer with Draco transformation system support organizes the legacy code, obtaining code still in the same language, however organized according to object-oriented principles (Fontanette et al. 2002).

In the second step, Recover Project, the software engineer starts off with the organized source code and again with the Draco support obtains the system MDL (Modeling Domain Language) description (Fontanette et al. 2002). Using the MVCCase tool, an object-oriented CASE tool designed for this specific methodology, the software engineer imports the MDL description to obtain the current system recovered project. This recovered project would reveal the class diagram of the system with the system classes, their respective attributes, methods and relationships (Fontanette et al. 2002).

**Figure 2.10 Strategy of Component-Oriented Software Re-engineering using Transformations**



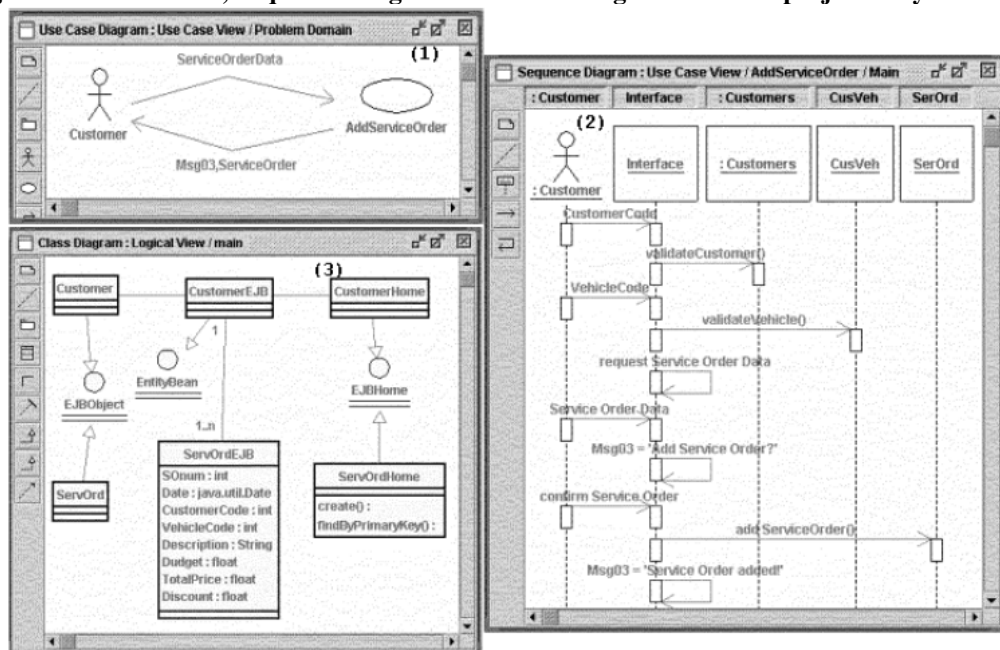
(Source: Fontanette et al. 2002 p.3)

In the third step, Reproject, the software engineer, using the MVCCase tool, does the component-based re-engineering of the recovered current system,. The initial modeling obtained in the previous step is used as a base for the specification and

project of components. The component specification is based on the Java/EJB technology, using Catalysis as a development method. The Problem Domain level, says "what" the system should do to solve the problem. The software engineer identifies the object and action types, gathering them in different views by business areas, and models use case diagrams, indicating the actors' relationships with the system (Fontanette et al. 2002).

The component specification level refines the specifications, from the previous level, emphasizing the component's identification, behavior and responsibilities. New more detailed models are obtained, but without worrying about the implementation. Sequence diagrams, which have as their objective to show the operation execution scenerios along the time, are built. Figure 2-11 shows an example of these component-based specification models.

**Figure 2.11 Use Case, Sequence Diagram and Class Diagram of the Reprojected System**



(Source: Fontanette et al. 2002 p.6)

Finally, in the fourth step of this re-engineering strategy, Reimplement, the MVCCase tool automatically implements the project recovered and specified in UML to the object-oriented language Java. The software engineer uses the MVCCase tool as a mechanism of code generation, starting from class diagrams with their attributes and prototypes of specified methods.

One major advantage of this Draco approach is that it helps to identify the component requirements models necessary for the requirements model evaluation. On the other hand, this methodology does not analyze the legacy source code for requirements model recovery.

Although the Draco approach can be used to generate the required component requirements models for this study, it also requires a mastery of the Draco machine and the MVCCase tool. These tools are available to the researcher as they are accessible via Internet.

However, the methodology is technology dependent on the tools mentioned and not a general approach that can be used with other types of re-engineering CASE tools. This could be seen as a limitation as there is no guarantee that the tools would be supported in the long term. It also limits the possibility of reproducing the research of this dissertation as the availability of these tools is not assured.

Another major limitation of this approach is that the MVCCase tool is based on the Catalysis approach of component-based requirements models and not only on UML. This could be a limitation if the researcher decides to use the UML approach as the framework for this research. Furthermore, the MVCCase tool only supports the Java language and this can be considered as another important limitation as many information systems require the combination of several computer languages. Finally, it needs access to source code and this might not be available for all the re-engineering projects.

### **2.7.3 The Jacobson and Lindstrom Methodology for System Re-engineering**

Jacobson and Lindstrom (1991) proposed a re-engineering methodology which assumes that the legacy requirements models are not available and can be recovered with the help of a reverse engineering methodology. This reverse engineering can be applied by following these steps:

- Develop a concrete graph that describes the components of the system and their interrelationship.
- Prepare an abstract graph showing the behavior and the structure of the system.
- Construct a mapping between the two, i.e. how something in the abstract graph relates to the concrete graph and vice versa.

The abstract graph should be free of implementation details. For example, mechanisms for persistent storage or partitioning into processes should not appear on this graph. The concrete graph must, on the other hand, show these details. The mapping between the two should tell how the ideal world of analysis is implemented by way of the concrete graph (Jacobson & Lindstrom 1991).

This abstract graph is in fact the requirements model. Once the requirements model is reverse engineered from the legacy system, the legacy system can be re-engineered by using the following steps (Jacobson & Lindstrom 1991):

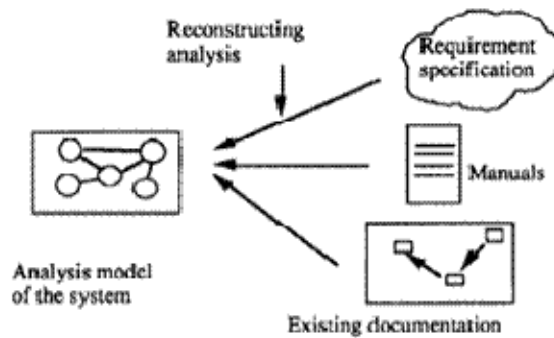
- Prepare an analysis model
- Map each analysis object to the implementation of the old system.

In order to prepare the analysis model step, it is important to assimilate the existing information about the system as illustrated in Figure 2-12. The existing information has many different forms, e.g. requirements specifications, user operating instructions, maintenance manuals, training manuals, design documentation, source code files, and database schema descriptions. These are called description elements (Jacobson & Lindstrom 1991).



From the set of description elements, an analysis model can be prepared. After the analysis model is completed, a map of each analysis object to the implementation of the old system is required. The map must show that all analysis objects and dependencies must be motivated by at least one primitive description element. This can be expressed with is-motivated-by, a mapping from the analysis model to the set of primitive description elements. All the dependencies in the analysis model must be motivated by at least one primitive description element. The use of guidance from experts in the legacy system should also be used to prepare the map as they can help to interpret the existing information and link it to the analysis model.

**Figure 2.12 Preparing an analysis model**



(Source: Jacobson & Lindstrom 1991)

The Jacobson and Lindstrom (1991) re-engineering methodology covers the case of a complete change of implementation technique with no change in functionality. This means that the legacy information systems can be re-engineered without adding any extra requirements. This is relevant for this research as the requirements models from the legacy and re-engineered component-based system need to be compared for equivalency of requirements representation.

The methodology also allows the recovery of the legacy requirements model from a set of description elements. This can be seen as a strength of this research as legacy systems normally do not carry updated documentation that reflects current requirements models and the reverse engineering capability of the methodology allows the recovery of the current requirements model of the legacy systems regardless of the updates that the system has experienced during its life cycle.

The methodology is also not dependent on specific software technologies making it more flexible than other methodologies that are technology dependent. This also allows the study to be reproduced for future research without worrying about specific technologies.

On the other hand, the methodology is based on object-oriented models while the objective of the research is to re-engineer component-based systems. However, it could be easily adapted for component-based systems to make it suitable for this research. The adaptation is not an issue as the component-based model is a higher level of abstraction of the object-oriented model as components are based on objects.

In this section, the re-engineering process was discussed and several methodologies presented. Although most of the existing re-engineering methodologies are object-oriented, they could be easily adapted to component-based provided that the legacy requirements model can be recovered (Fevre et al. 2003). The Deursen et al. (2000) methodology was discussed in the context of component-based systems re-engineering, and although it provides a good way of recovering the legacy requirements model, it does not generate the re-engineered business component models. On the contrary, the Draco methodology (Fontanette et al. 2002) is able to generate the business component models but unable to recover the legacy requirements model. The Jacobson and Lindstrom (1991) methodology was presented as a good fit for this research because it allows the recovery of the legacy requirements model and generation of object-oriented models. Although this was originally designed for object-oriented re-engineering, it can be easily adapted to the component-based paradigm.

Table 2-2 summarizes the evaluation criteria applied to the re-engineering methodologies reviewed and the justification for the selection of the Jacobson and Lindstrom methodology for this research.

**Table 2.2 Summary of Evaluation of Re-engineering methodologies**

<b>Jacobson and Lindstrom</b>	<b>Draco</b>	<b>Deursen</b>
It allows the recovery of the legacy requirements model from a set of description elements.	It helps to identify the component requirements models.	It allows the recovery of the legacy requirements model.
It is not dependent on specific software technologies.	It is dependent on specific software technologies.	Too complex to use because it relies on the analysis of source code.
It allows the study to be reproduced for future research without worrying about specific technologies.	It limits the possibility of reproducing the research since the availability of the software tools required is not assured.	Concentrated in the development of software models and not requirements models.
It supports the UML language.	It supports only the Catalysis grammar.	

In the next section the concept of ontologies will be introduced as this research is based on this concept for the evaluation of requirements models in the re-engineering process.

## **2.8 Ontologies**

Before ontological evaluation is introduced, it is important to define ontology. Ontology is a well-established theoretical domain within philosophy dealing with models of reality. Over the years, many different ontologies have emerged. Mylopoulos (1998) suggests that ontologism can be classified into four categories: static, dynamic, intentional, and social. Each of these categories focuses on different

concepts in the real world. Ontologies that fall into the static category focus on things and their properties. Dynamic ontologies extend static ontologies to focus on such concepts as events and processes, that is, how concepts in the real world change over time. Intentional ontologies attempt to explain abstract concepts such as goals and objectives while social ontologies emphasize the concepts of values and beliefs.

Today however, interest in, and applicability of ontologies, extends to areas far beyond metaphysics. Two general ontologies have been frequently applied for the evaluation of modeling methods in Systems Analysis and Design. First, much work has focused on a set of ontological models known as the BWW (Bunge-Wand-Weber) models. Weber (1997) has taken, and extended, ontology presented by Bunge (1977) and applied it to the modeling of information systems.

Second, Chisholm's ontology (1976) has been used to evaluate a representative range of data modeling languages (grammars) with a view to gain insight into those languages (Milton et al. 2001).

Also, special enterprise and IS ontologies, e.g., the enterprise ontology (Uschold et al. 1998) and the framework of information systems concepts (FRISCO) (Verrijn-Stuart et al. 2001) have been developed for the evaluation of IS modeling methods.

Their fundamental premise is that any Systems Analysis and Design modeling grammar (set of modeling symbols and their construction rules) must be able to represent all things in the real world that might be of interest to users of information systems. Otherwise, the resultant model is incomplete. If the model is incomplete, the analyst/designer will somehow have to augment the model(s) to ensure that the final computerized information system adequately reflects that portion of the real world it is intended to simulate.

The BWW model is not the only ontology available to evaluate information systems as alternatives exist both in the form of general philosophical ontologies such as the Chisholm ontology (1996), or special enterprise and IS ontologies such as the enterprise ontology (Uschold et al. 1998) and FRISCO (Verrijn-Stuart et al. 2001). However, the use the BWW model is justified for two reasons. First, the model is based on concepts that are fundamental to the computer science and information systems domains (Wand & Weber 1993). Second, it has already been used successfully to analyze and evaluate the modeling constructs of many established IS and enterprise modeling languages such as dataflow diagrams, ER models, OML and UML (Evermann & Wand 2001; Green & Rosemann 2000; Opdahl & Henderson-Sellers 2002a; Weber & Zhang 1996) and for the evaluation of enterprise systems (Green et al. 2005) and business component frameworks (Fettke and Loos 2003b).

In addition, the following arguments of Wand and Weber (1993) support its use:

- Better developed and formalized than alternative philosophical ontologies. The BWW ontology has been formalized with a representation model and constructs (Wand & Weber 1988, 1993, 1995).

- Based on concepts that are fundamental to the computer science and information systems domains. Constructs were developed with the intention of representing information and computer systems.
- Productive, in the sense that it has given useful results. Research shows that the model has been useful for the evaluation of information systems methodologies (Wand & Weber 1988, 1993, 1995).

The BWW models consist of the representation model, the state-tracking model, and the decomposition model. The work reported in this dissertation uses the BWW representation model and its constructs. The representation model defines a set of constructs that, at this time, are thought to be necessary and sufficient to describe the structure and behavior of the real world.

In the next section, the BWW model will be discussed in detail and its constructs described.

### **2.8.1 Bunge-Wand-Weber model**

The Bunge-Wand-Weber representation model (Wand & Weber 1988, 1993, 1995) has been used to analyze and evaluate the modeling constructs of many established IS and enterprise modeling languages such as dataflow diagrams, ER models, OML and UML (Opdahl & Henderson-Sellers 2004).

According to Bunge's ontology and the BWW model, there is a world that exists independently of human observers, and it consists of things that possess properties. Examples of BWW things are "atoms, fields, persons, artifacts and social systems" (Opdahl & Henderson-Sellers 2002a, p. 47), whereas "properties of things (e.g., energy), changes in them, and ideas considered in themselves" are non-things (Opdahl & Henderson-Sellers 2002a, p. 47). In particular, concepts are not BWW things. Bunge's ontology and the BWW model also remind us that we only know about things via models of things we create in our minds, and that we ascribe attributes to those models of things to stand for the properties we believe the corresponding things possess. In the BWW model, an attribute (that stands for a BWW property) is represented as a property function of time, which maps the property onto different property values in a property co-domain for different points in time.

The BWW model distinguishes between properties in several different ways. An intrinsic property belongs to only a single thing, whereas a mutual property belongs to two or more things. BWW mutual properties are represented by relationships or similar constructs in many modeling languages.

A whole-part relation is a property that relates an aggregate thing to one of its component things. A resultant property belongs to a BWW aggregate and is derived from one or more properties of its components, whereas an emergent property belongs to a BWW aggregate but not to any of its components. A law property restricts other properties of the same thing.

A BWW law is either a state law or a transition law. An individual property (or property of a particular) is a specific such as "being 25 years old" and "having grey

hair,” whereas the corresponding general properties are “having an age” and “having a hair color.” Bunge (1977) also distinguishes between BWW-properties that are permanent and those that are variable.

BWW properties may be complex because they may have other properties as constituents. A BWW property precedes a second BWW-property if and only if:

- Either (a) the second property is complex (or compound) and the first property is one of its constituents,
- Or (b) a BWW law states that all BWW things that possess the second property must also possess the first.

According to (a), “having a ZIP-code” precedes “having a postal address” because every postal address includes a ZIP-code and, according to (b), “being a human being” precedes “being married”.

Things with a property in common form BWW classes. A class contains all the things, and only those things, that possess one or more characteristic properties for the class. In other words, every BWW class is defined by a nonempty set of characteristic properties of the things in the class. The most general BWW class is the class of all things, which is defined by the universal property of being able to associate with other things (Bunge, 1977). Because characteristic properties may be complex, it is sometimes possible to say that a BWW class is defined by a group of characteristic BWW properties.

One BWW class may be defined by a group of characteristic properties that is contained in a larger group of properties that defines a second class. We then say that the second BWW class is a subclass of the first.

A BWW thing has time-dependent states that are determined by the values of the thing’s property functions over time. A change of BWW state in a thing is an event, hence a BWW event can be described as a pair of BWW states. Consecutive BWW events form complex events, or processes if they occur in the same thing. The sequence of consecutive BWW states undergone by a thing (or, alternatively, the sequence of consecutive BWW events) is called its history. A BWW thing acts on a second thing if and only if the BWW history of the second thing would have been different had the first thing not existed. The first thing is called an active thing. Two BWW things are coupled if and only if (at least) one of them acts on the other. BWW couplings are caused by certain BWW mutual properties that are said to be binding. A BWW aggregate whose BWW components are coupled is a system.

Systems are things that are made up of other things that satisfy two conditions. First, every thing in the system must be coupled to at least one other thing. Second, it must not be possible to divide the things that make up the system into two subsets such that the history of one subset of things is independent of the other subset of things (in other words, the subsets are not coupled) (Wand & Weber 1995).

The composition of a system is the set of things that are in the system. The environment of a system is the set of things that are not in the system’s composition

but interact with (are coupled to) at least one other thing in the system's composition (Wand & Weber 1995).

The structure of a system is the set of internal couplings (between things in the composition of the system) and external couplings (between things in the composition of the system and things in the environment of the system) (Wand & Weber 1995).

A subsystem is a system that satisfies the following conditions ((Wand & Weber, 1995):

Its composition is a subset of another system's composition. In other words, all things in the subsystem are also things in another system.

Its environment is a subset of the environment of the other system joined with the difference between the composition of the other system and composition of the subsystem.

In other words, first we take the things that are in the environment of the system. To these we add the things that are in the composition of the system but not in the composition of the subsystem. The things in the environment of the subsystem will be a subset of this newly formed set of things. Its structure is a subset of the other system's structure. In other words, all internal couplings and all external couplings in the subsystem are also internal couplings and external couplings of the other system.

The input of a thing is the set of state changes (events) to a thing that have arisen by virtue of the actions of things in its environment. In the same way, we define the output of a thing as the set of all events that occur to things in the environment of the thing by virtue of the action of the thing. In other words, if we identify those events that have occurred to things in the environment of a thing only because they are coupled to the thing, we have the output of the thing (Wand & Weber 1995).

A set of things may be a subtype of a type (subclass of a class) if the things possess the property of the type plus at least one other property that is not possessed by all instances of the type. Subtypes may also be disjoint and have a rigid property which is essential for its type (Wand & Weber 1995).

The hereditary properties of a thing are properties that also belong to things in the thing's composition. The emergent properties of a thing are properties that are not properties of any of its components (Weber & Zhang 1996). Emergent properties are always functions of other properties, although often we cannot clearly articulate the nature of the relationship that exists. Simple emergent properties are aggregates.

A special relationship exists between systems and emergent properties. All systems must have an emergent property of some kind. The only reason for our conceiving a set of things as a system is that the composite thing (system) possesses at least one emergent property that is of interest to us for some purpose.

A thing is called a composite thing if it is composed of (made up of) things other than itself (has proper parts). Things in the composite are part-of the composite.

Table 2-3 summarizes the main constructs of the BWW model.

**Table 2.3** Constructs of the BWW-model

<b>Ontological Construct</b>	<b>Description</b>
THING	The elementary unit in the ontological model. The real world is made up of things. A composite thing may be made up of other things (composite or primitive).
PROPERTY	Things possess properties. A property is modeled via a function that maps the thing into some value. A property of a composite thing that belongs to a component thing is called a hereditary property. Otherwise it is called an emergent property. A property that is inherently a property of an individual thing is called an intrinsic property. A property that is meaningful only in the context of two or more things is called a mutual or relational property.
STATE	The vector of values for all property functions of a thing.
CONCEIVABLE STATE SPACE	The set of all states that the thing might ever assume.
STATE LAW	Restricts the values of the property functions of a thing to a subset that is deemed lawful because of natural laws or human laws.
EVENT	A change of state of a thing. It is effected via a transformation (see below).
EVENT SPACE	The set of all possible events that can occur in the thing.
TRANSFORMATION	A mapping from a domain comprising states to a Co-domain comprising states.
PROCESS	An intrinsically ordered sequence of events on, or state of, a thing.
LAWFUL TRANSFORMATION	Defines which events in a thing are lawful.
HISTORY	The chronologically ordered states that a thing

Ontological Construct	Description
	traverses.
ACTS ON	A thing acts on another thing if its existence affects the history of the other thing.
COUPLING	A thing acts on another thing if its existence affects the history of the other thing. The two things are said to be coupled or interact.
SYSTEM	A set of things is a system if, for any bi-partitioning of the set, couplings exist among things in the two subsets.
SYSTEM COMPOSITION	The things in the system.
SYSTEM ENVIRONMENT	Things that are not in the system but interact with things in the system.
SYSTEM STRUCTURE	The set of couplings that exist among things in the system and things in the environment of the system.
SUBSYSTEM	A system whose composition and structure are subsets of the composition and structure of another system.
SYSTEM DECOMPOSITION	A set of subsystems such that every component in the system is either one of the subsystems in the decomposition or is included in the composition of one of the subsystems in the decomposition.
LEVEL STRUCTURE	Defines a partial order over the subsystems in a decomposition to show which subsystems are components of other subsystems or the system itself.
STABLE STATE	A state in which a thing, subsystem or system will remain unless forced to change by virtue of the action of a thing in the environment (an external event).
UNSTABLE STATE	A state that will be changed into another state by virtue of the action of transformation in the system.
EXTERNAL EVENT	An event that arises in a thing, subsystem or system by virtue of the action of some thing in the environment on the thing, subsystem or system. The before-state of an external event is always stable. The after-state may be stable or unstable.



Ontological Construct	Description
INTERNAL EVENT	An event that arises in a thing, subsystem, or system by virtue of lawful transformations in the thing, subsystem, or system. The before-state of an internal event is always unstable. The after state may be stable or unstable.
WELL DEFINED EVENT	An event in which the subsequent state can always be predicted given the prior state is known.
POORLY DEFINED EVENT	An event in which the subsequent state cannot be predicted given the prior state is known.
CLASS	A set of things that possess a common property.
KIND	A set of things that possess two or more common properties.

(Source: (Wand & Weber 1993; Weber & Zhang 1996))

In this section, the BWW model was discussed as an ontological tool to evaluate information systems models. The BWW model has been used in the past to evaluate representation grammars in information systems and it can also be used as an ontological framework for this study.

### 2.8.2 Ontological Evaluation of Requirements models

A major component of this dissertation is the evaluation of legacy and re-engineered component systems requirements models in order to verify that both represent the same business requirements. In the past, Mišić and Zhao (2000) and Schütte (1998) developed a framework for the evaluation of requirements models. Although they could have been used to compare the legacy and re-engineered requirements models, the problem with these feature-based evaluation approaches is that the development and selection of a specific feature set is often a subjective issue that is not based on sound theory.

Fettke and Loos (2003) proposed an approach to the ontological evaluation of requirements models based on the Bunge-Wand-Weber ontology theory. The main idea of this approach is the ontological normalization of a requirements model. An ontological normalization is comparable with the normalization of a database schema. The objective of both techniques is to represent the domain of interest in a normalized way by applying specific transformation patterns. Normalization of a database schema aims at eliminating problems of information representation and processing in database management systems (e.g. avoiding data redundancies, problems of lost update, dirty read etc.). In contrast, the ontological normalization aims to achieve a unified representation of facts represented by a requirements model with respect to the structure of reality.

The ontological normalization of a reference model consists of four steps (Fettke & Loos 2003):

1. Developing a transformation mapping,
2. Identifying ontological modeling deficiencies,
3. Transforming the reference model, and
4. Assessing the results.

In the first step of this method, it is necessary to develop a transformation mapping for the grammar used for representing the requirements model. This transformation mapping allows conversion of the constructs of the used grammar to the constructs of the BWW model (Wand and Weber 1993). The transformation mapping consists of two mathematical mappings. First, a representation mapping describes whether and how the constructs of the BWW model are mapped onto the grammatical constructs. Second, the interpretation mapping describes whether and how the grammatical constructs are mapped onto the constructs of the BWW model (Fettke & Loos 2003).

With respect to both mappings, four ontological deficiencies can be distinguished (Fettke & Loos 2003).

- Incompleteness: A grammar is incomplete if the representation mapping is not defined in total. Otherwise a grammar is complete.
- Redundancy: A grammar is redundant if the representation mapping is ambiguous.
- Excess: A grammatical construct is excessive if it cannot be mapped onto an ontological construct. A grammar is excessive if at least one of its constructs is excessive.
- Overload: A grammatical construct is overloaded if it can be mapped onto more than one ontological construct. A grammar is overloaded if at least one of its constructs is overloaded. This can be seen as a modeling deficiency and will be explained below in more detail.

To prepare the ontological normalization of the requirements model, all ontological deficiencies of the requirements models have to be identified. This is the objective of the second step. The second step is based on the former constructed transformation mapping. It is possible that one ontological deficiency is resolvable in various ways or even not resolvable at all. Hence, it is useful to separate the identification of ontological modeling deficiencies from the transforming step of the requirements model (the next step) (Fettke & Loos 2003).

To identify the ontological deficiencies of the requirements model all constructs of the requirements model must be reviewed. Each construct of the requirements model must be examined with respect to whether the construct is used correctly regarding the interpretation mapping. One of the following situations can arise (Fettke & Loos 2003):

- Adequacy: The grammatical construct is ontologically adequate.
- Excess: Construct excess is a modeling deficiency in general and needs special handling in the transformation step. Construct excess occurs if

implementation specific aspects are represented in the requirements model, e.g. the technical concepts of message passing or polymorphism cannot be represented with ontological constructs.

- **Overload:** Construct overload is a modeling deficiency in general and needs special handling in the transformation step. This construct should be marked as overloaded in the requirements model. For instance, using UML, UML object can represent a BWW thing (UML object “Mr. Miller” is an instance of the UML class customer) or a BWW class (UML objects “a class journal”, “b class journal” etc. are instances of the UML class “journal categories”). So, the construct UML object is ontological overloaded.

The described identification step of modeling deficiencies relies on the interpretation mapping. In addition, the representation mapping supports an indirect means to identify modeling deficiencies. Based on the representation mapping it can be decided whether the used grammar is incomplete or redundant. An incomplete grammar leads to the trend that specific facts of reality cannot be adequately represented in the requirements model. (Fettke & Loos 2003).

In the third step, the requirements model will be transformed to an ontological model. The outcome of this step is an ontologically normalized requirements model. More formally, an ontologically normalized requirements model is a mapping from the constructs of the requirements model to the constructs of an ontological model. While mapping a construct of the requirements model onto an ontological construct, four cases can arise (Fettke & Loos 2003):

- **Adequacy:** The construct of the requirements model is marked as adequate. It is possible to map this construct in a straightforward way onto a construct of the ontological model.
- **Inadequacy:** The construct of the requirements model is marked as inadequate. It is necessary to interpret the representation in the reference model in a sensible manner. The result of this interpretation may be that it is possible to represent this construct by a specific construct of the ontological model.
- **Excess:** The construct of the requirements model cannot be mapped onto a construct of the ontological model with respect to the interpretation mapping. Nevertheless it should be examined whether it is possible to represent this construct by a specific construct of the ontological model.
- **Overload:** The construct of the reference model can be mapped onto several constructs of the ontological model with respect to interpretation mapping. It is necessary to decide which interpretation mapping is preferable regarding the representation in the reference model. The result of this decision may be that it is possible to represent this construct by exactly one construct of the ontological model.

The resolution of the ontological deficiencies of constructs should be guided by the intention of these constructs. This step relies on the interpretation of the subject performing the evaluation. The result of this transformation is an ontological model representing the requirements model in an ontologically normalized way. The ontologically normalized model is assessed regarding different aspects in the next step (Fettke & Loos 2003).

In the last step, the requirements model can be evaluated regarding the results of the three mentioned steps above (Fettke & Loos 2003):

- Assessing the transformation mapping in general,
- Assessing the ontological deficiencies of constructs in particular, and
- Assessing the ontologically normalized reference model.

First, the transformation mapping can be assessed in general. Based on the representation and interpretation mappings it is possible to determine the ontological clarity and adequacy of the used grammar. This assessment gives an idea as to whether the used grammar is suitable to represent the facts of reality with regard to the intended application in general (Fettke & Loos 2003).

Second, the ontological deficiencies of constructs of the reference model can be assessed in particular. While the ontological deficiencies of excess and overload have their roots in the definition of the grammar, the cause of an ontologically inadequate construct of the reference model is the specific application of a grammatical construct employed by the person who developed the model. Note that an ontologically adequate construct of the reference model is not ontological equivalent to a correct modeling (in a syntactical meaning). Instead, the high usage of inadequate constructs may be a sign of representing many implementation aspects in the reference model (Fettke & Loos 2003).

Third, the ontologically normalized reference model can be assessed. In this case, two different evaluation aspects are reasonable (Fettke & Loos 2003 ):

- Isolated assessment: Different metrics can be used for an isolated assessment of the ontological model.
- Comparative assessment: Comparative evaluations of reference models can be undertaken if further ontological models of the application domain are given. In this manner, it is possible to evaluate a reference model with respect to its completeness. Such an evaluation is possible only with respect to another ontological model.

The Fettke and Loos (2003) methodology can be a useful research tool given its capacity of comparing requirements models based on their normalized reference models. These normalized requirements models are the ontological representation of the requirements models in the comparison. If two requirements models are ontological equivalent, their normalized reference models should be ontological equivalent.

However, the transformation of the requirements models into ontologically normalized requirements models can be quite difficult as transformation mapping can be challenging without an appropriate methodology. Although Fettke and Loos (2003) identify this mapping as a step in the ontological business evaluation, they do not provide mappings of grammar constructs for UML or traditional models. Furthermore, the details about the implementation of the normalized reference models are not presented in their published research.

In the next sections of this chapter, the major attempts to map information systems' traditional and UML grammars into BWW constructs are presented and analyzed in the context of this research.

### **2.8.3 BWW mapping for process modeling**

Traditional requirements models are constructed by the use of Data Flows. There are two types of modeling in DFDs: 1) Physical DFDs – where the diagram describes the physical components of the information system and 2) Logical DFDs – that describe the meaning or the 'what' of the components of the information systems (Wand & Weber 1989).

As this research deals only with the requirements models, only logical DFDs will be analyzed. Wand and Weber (1989) developed an interpretation of DFDs to BWW constructs. According to their interpretation, data stores represent state information, and data flows represent external and internal events. Properties of real things may be represented by data elements described in data dictionaries but not in data flows and data stores.

There is no explicit representation of the states of the real system in a DFD. Rather, the possible and allowed states of the information systems are defined implicitly in terms of possible and allowed values of the data elements described in the data dictionary and therefore not represented in the DFDs (Wand & Weber 1989).

External events of the information system are represented by data flows coming from a source while internal events are represented by internal data flows that are generated because the system responds to an external event. Data linked to a process, a process linked to another process and a process linked to an external agent may be interpreted as coupling (Wand & Weber 1989).

A DFD represents a proper system if and only if there is a path between every pair of processes. If this is not the case, then the DFD represents two or more disconnected information systems. External agents and data stores are represented by things and they form part of the environment in the BWW model (Wand & Weber 1989).

In DFDs, decomposition involves breaking a process "bubble" into a number of sub-processes. DFDs conform to the BWW model notion of a good decomposition (Wand & Weber 1989).

Another process modeling interpretation comes from the work of Green and Rosemann (1999). This work presents the mapping between one of the most successful grammars for process modeling, that is, the event-driven process chains (EPC) and the BWW model. This grammar is embedded in the Architecture of Integrated Information Systems (ARIS) (Scheer 1998).

In the process view of ARIS, the thing as an elementary ontological construct is not a part of the original meta-model of event-driven process chains. Because a function type within an event driven process chain can be seen as the transformation of a business relevant object, an EPC function type can be interpreted to represent a

property in general of that object. Attribute types in EPC represent attributes in the BWV representation model. The ontological construct class, however, is not represented in the EPC-grammar. As opposed to grammars that depict the structure of a system (e.g., the Entity Relationship (E-R) model), process modeling languages focus on the behavioral aspects of what is being modeled. Consequently, the ontological constructs state, transformation, and event are most relevant. Transformations are represented by function types in the event-driven process chains while states are depicted as event types. Accordingly, the triple, 'event type – function type – event type', in an EPC represents the ontological construct event, and usually internal events that are well defined. The homonym between the EPC event type and the ontological event requires careful attention during the analysis. Similarly, a state law can be represented by the triple, 'function type – connector – event type', while a lawful transformation can be represented by the pattern, 'event type – connector – function type'. An external event may be represented by the start event type at the beginning of an EPC while the final stable state (of an object) may be represented by the end event type at the bottom of an EPC (Green & Rosemann 1999).

Although Green and Rosemann (2000) developed a transformation map of ARIS to BWV constructs that could be used for ontological evaluations of legacy requirements models, most legacy systems were not built based on the ARIS framework but by using DFDs that differ from the ARIS framework for process modeling. In the case of the ARIS framework, EPCs are used to model business processes while DFDs are used by the traditional approach for the same representation.

However, the work of Green and Rosemann (2000) can be used to complement and compare some of the work of Wand and Weber (1989) as both approaches use ERDs and context diagrams.

Entity-relationship diagrams (ERD) are interpreted by both Wand and Weber (1989) and Green and Rosemann (2000). Although Wand and Weber's (1989) interpretation that entities and relationships can be viewed as representing things of a real system, the interpretation of Green and Rosemann (2000) of the entity representing a class seems more accurate as entities can represent multiple instances of things. Properties are represented directly in the entity relationship diagram via the notion of attributes in both interpretations. Coupling between things can be represented by the relationships between entities (Wand & Weber 1989).

Another limitation of Wand and Weber's approach is the lack of interpretation of functional decomposition diagrams that are interpreted by Green and Rosemann's (2000) work. Also Wand and Weber (1989) did not include the transformation construct mapping as this was added to the BWV model after the publication of their analysis.

Limitations of process modeling are acknowledged by Rosemann et al. (2005) and Green and Rosemann (2000). Functional decomposition diagrams are ontologically redundant when compared to the combination of DFDs, ERDs, and context diagrams.

The result of the interpretation of Wand and Weber (1989) and Rosemann et al. (2005) is that no ontological representations exist for conceivable state, state space, lawful state space, conceivable event space, lawful transformation or lawful event space BWW constructs.

Accordingly, problems may be encountered in capturing all the potentially important business rules of the situation.

Also, no representations exist for stable state, unstable state, well defined event and poorly-defined event. Again, the usefulness of traditional diagrams for defining the scope and boundaries of the system being analyzed is undermined.

#### 2.8.4. BWW mapping for component based requirements models

Component-based models can be represented by using several approaches including UML and Catalysis. Although specific component-based frameworks have been proposed for the specification of business components (Ackermann et al. 2002), not all of them are ontologically capable of representing business requirements. Fettke and Loos (2004) showed that this last mentioned framework was ontologically incomplete and weak for representing business components.

In spite of the difficulties of different grammars for the representation of the business requirements, the UML grammar has been used as a standard for the specification of requirements models for many years and has been evaluated and mapped into BWW constructs by several authors. Perhaps the most complete analysis was by Opdahl and Henderson-Sellers (2002b) as they included in their research a complete map of UML to BWW constructs as seen in table 2-4.

**Table 2.4 BWW interpretation of UML constructs**

UML construct	Interpretation
UML object	BWW thing
UML active object	BWW thing that acts on another thing
UML swimlane	BWW thing that acts on another things
UML actor	BWW thing that acts on the proposed system thing
UML object lifeline	A segment of a BWW history
UML type	BWW natural kind
UML supertype	BWW natural kind that has a subkind
UML subtype	Subkind
UML generalization	Natural kind/sub kind relationship
UML actor class	BWW natural kind of things that act on the proposed system thing
UML active class	BWW natural kind of things that act on another things
UML property	BWW intrinsic property
UML attribute-of a class	BWW characteristic intrinsic property
UML multiplicity	BWW characteristic state law
UML data type	BWW co-domain of a property function
UML operation	BWW transformation
UML precondition	Subtype of BWW intrinsic state law

UML construct	Interpretation
UML postcondition	Subtype of BWW intrinsic transformation law
UML responsibility (of class)	Subtype of BWW complex law property
UML link	BWW eventual property of two or more things
UML association	BWW characteristic mutual property
UML-link object	BWW composite thing
UML association class	BWW natural thing
UML communication association	BWW characteristic binding mutual property
UML aggregate	BWW composite thing
UML aggregate class	BWW natural kind of composite things
UML aggregation	BWW whole part relation
UML composition	BWW whole part relation
UML container	Subtype of BWW thing
UML physical system	BWW system composition
UML state	BWW state
UML object flow state	Subtype of BWW state of a BWW thing
UML event	BWW event
UML sender	BWW thing that acts on another thing
UML-focus on control	Sequence of BWW unstable states in a thing
UML-use case instance	BWW process in the proposed system thing
UML use case class	A group of BWW processes in the proposed system thing
UML-extend	Subtype of BWW binding mutual property
UML-include	Subtype of BWW binding mutual property
UML-scenario	BWW process in the proposed system thing
UML timing mark	Element in the domain of any BWW property
UML time event	Subtype of BWW event

(Source: Opdahl and Henderson-Sellers 2002b p.51)

Although this work presents UML as a strong grammar from the ontological point of view, UML has been criticized by Irwin and Turk (2005) as ontologically incomplete. Irwin and Turk (2005) argue that UML is incomplete with respect to representing the system structure or decomposition. There are no clearly-defined constructs for representing systems at different levels of detail such that no information is lost between levels (Irwin & Turk 2005). Also, the definitions of actor, use case, association, and generalization are ontologically overloaded and the <<include>> and <<extend>> constructs overlap with other UML constructs, such as aggregation (Irwin & Turk 2005).

Although Opdahl and Henderson-Sellers (2002b) presented an analysis of all UML constructs, Dussart et al. (2004) conducted an ontological analysis of UML from the diagram perspective. They prepared a study of three UML diagrams used for the specification of component based systems: the activity, the state and the sequence diagrams. In their study, a mapping was created to map UML constructs into the BWW model. The results of this mapping are shown in Table 2-5.



**Table 2.5 BWW Representation Model Analysis for Dynamic Aspects of UML**

<b>Ontological construct</b>	<b>Activity Diagram</b>	<b>State Diagram</b>	<b>Sequence diagram</b>	<b>Other Views</b>
Thing	Object Swimlane	Object	Object	
Property	Activity Swimlane			
Class				Class (Diagram)
Kind				Generalization (Class diagram)
State	State of Object	State		
Conceivable State Space		State Machine		
State Law		State→transition→State		
Lawful State Space		Substates		
Process	Activity Diagram Activity			
Event	Activity	Trigger		
Conceivable Event Space		All triggers		
Transformation	Activity			
Lawful transformation	Guard conditions on transitions			
Lawful event state				

Ontological construct	Activity Diagram	State Diagram	Sequence diagram	Other Views
History		Shallow history state construct		
Acts on				
Coupling			Messages	
System			Sequence diagram	Package with <<System>>
System composition			Object	
Subsystem				Package with <<subsystem>>
System Decomposition				Composition
Level structure				Generalizations
External Event		<<Stereotype>>		
Stable state		Final State		
Unstable state		Initial State		
Internal Event		<<Stereotype>>		
Well defined Event		Trigger		
Poorly defined Event				

(Source: Dussart et al. 2004 p.85)

The work of Dussart et al. (2004) revealed that the activity diagram overlaps ontologically with the state diagram. As both state diagrams and activity diagrams represent the behavior of objects (Evermann & Wand 2001) such overlap seems to be justifiable. As part of the analysis of ontological completeness, Dussart et al. (2004) showed that there are several constructs that cannot find representation in the BWW model: lawful event space, acts on and poorly defined event.

Although UML could be considered incomplete, this incompleteness has been minimized to only four constructs that are not necessarily essential to workflow modeling (Dussart et al. 2004), and this could confirm a conclusion by Green and Rosemann (1999) who raised the question of a possible over-engineering of the BWW model and a need for a contextual individualization of the model.

In this section the ontological analysis of UML made by several authors has been introduced and their maps to BWW constructs presented. Although the UML has been shown as ontologically incomplete, it has enough constructs to represent business information systems requirements and therefore a strong grammar to generate component requirements models.

## **2.9 Templates for the transformation of requirements models into ontological models**

In the third step of the Fettke and Loos (2003) methodology for ontological evaluation of requirements models, requirements models need to be transformed to ontological models in order to compare them for equivalence. Although Fettke and Loos (2003) mentioned this as part of the methodology, they did not mention how to accomplish this.

This is a major issue when it comes to the implementation of this methodology. However, there are at least two template models that can help with this problem. The first one is the template for defining enterprise modeling constructs proposed by Opdahl and Henderson-Sellers (2004) and the second is the Green and Rosemann (1999) BWW meta-model for the description of ontological models in BWW construct terms. This first template will be explained in the section below.

### **2.9.1 Opdahl and Henderson-Sellers template model for enterprise modeling**

The main idea behind the Opdahl and Henderson-Sellers (2004) template is to provide a standard way of defining enterprise and IS modeling constructs in terms of the BWW model, in order to make the definitions cohesive and, thus, learnable, understandable and as directly comparable to one another as possible. When all construct definitions are directly comparable, it becomes easier to translate models from one language to another (Opdahl & Henderson-Sellers 2004). This could be used to transform legacy systems and re-engineered systems models into BWW models for ontological evaluation of business requirements equivalency.

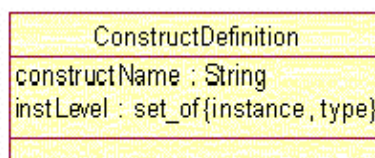
The template is used to define each modeling construct separately by filling in four types of top-level entries, some of which have sub-entries (Opdahl & Henderson-Sellers 2004):

- The instantiation level entry type is used to define whether the modeling construct represents the enterprise at the type level, at the instance level or at either level. This is the simplest type of top-level entry.
- The class entry type is used to define which class of things (or classes of things) in the enterprise that the modeling construct may represent.
- The property entry type is used to define which property (or properties) in the enterprise the construct may represent. It may be repeated and may have several subentries.
- The lifetime entry type is used to define whether the modeling construct represents events in, states of, processes in or the whole lifetime of one or more things.
- Each type of top-level entry can be represented separately by using constructs from UML. The first and simplest entry type is used to define the instantiation level of a modeling construct. The construct is at the type level if it represents BWW classes (or their characteristic properties, etc.) and it is at the instance level if it represents BWW things (and/or their properties, states, events, histories, etc.) (Opdahl & Henderson-Sellers 2004).

Figure 2-13 shows the first part of a UML class diagram for the template, according to which a ConstructDefinition has a constructName and an instLevel has attributes. In the UML, when multiplicities are not shown for attributes, the default is one to one, so each ConstructDefinition has exactly one instLevel attribute (Opdahl & Henderson-Sellers 2004).

The second type of entry is used to define which class of things the modeling construct may represent. For a modeling construct at the type level, this means that the construct may only represent subclasses of the specified class. For a modeling construct at the instance level, this means that the construct may only represent things that belong to the specified class (Opdahl & Henderson-Sellers 2004).

**Figure 2.13 UML class diagram of the instantiation level entry**

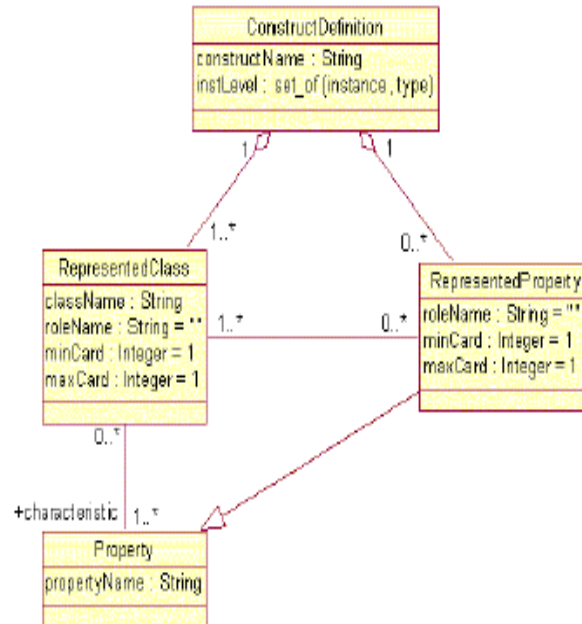


(Source: Opdahl & Henderson-Sellers 2004)

Figure 2-14 extends the UML class diagram to show that a ConstructDefinition consists of one or more RepresentedClasses, each of which is defined, according to the BWW model, by one or more CharacteristicProperties (Opdahl & Henderson-Sellers 2004). According to Figure 2-14, the template allows repeated class entries for modeling constructs that may represent several different classes of things (at the

type level) or several things of different classes (at the instance level) (Opdahl & Henderson-Sellers 2004).

**Figure 2.14 UML class diagram extended to show the class entity**

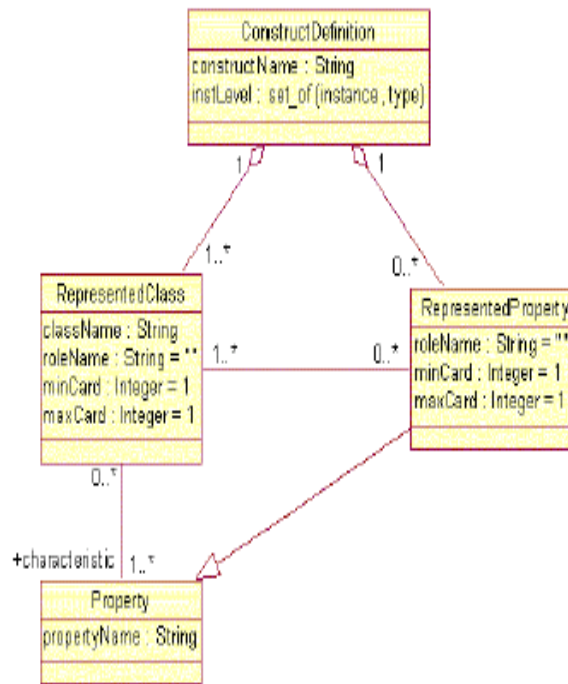


(Source: Opdahl & Henderson-Sellers 2004)

The third type of entry is used to define which properties of things the modeling construct may represent. Sometimes different modeling constructs may represent the same class of things but not the same properties of those things (Opdahl & Henderson-Sellers 2004).

Figure 2-15 extends the UML class diagram to show that a `ConstructDefinition` also consists of zero or more `RepresentedProperties`, which specialize the `Properties` that characterize `RepresentedClasses`.

**Figure 2.15 UML class diagram extended to show the property entry**



(Source: Opdahl & Henderson-Sellers 2004)

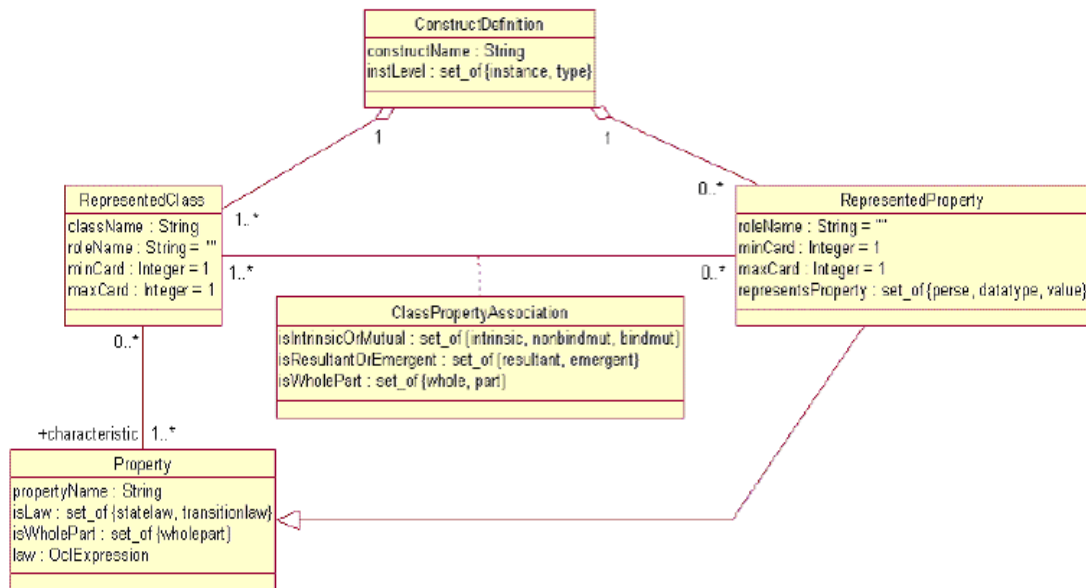
The BWW model has concepts that describe properties in even greater detail and that are also used in the template. Figure 2-16 extends the UML class diagram to show the additional attributes of Properties and RepresentedProperties.

Most importantly, according to the BWW model, a `RepresentedProperty` has an attribute that defines whether the modeling construct represents (a) the property per se, i.e., the BWW property itself; (b) the property datatype, i.e., the BWW property co-domain; (c) a property value, i.e., a value in the BWW property co-domain; or (d) some combination of these.

A `Property` has an attribute that defines whether the modeling construct represents a non-law, a state law or a transition law according to the BWW model. A `Property` that is a law is described by an `oclExpression`. A `Property` also has an attribute that defines whether the modeling construct represents a wholepart relation or not according to the BWW model.

Figure 2-16 also shows the additional attributes of the `ClassPropertyAssociation` class. The first of these defines whether the `RepresentedProperty` is intrinsic, nonbinding mutual or binding mutual with respect to a particular `RepresentedClass`.

**Figure 2.16 UML class diagram extended to show the ontological descriptions of properties**



(Source: Opdahl & Henderson-Sellers 2004)

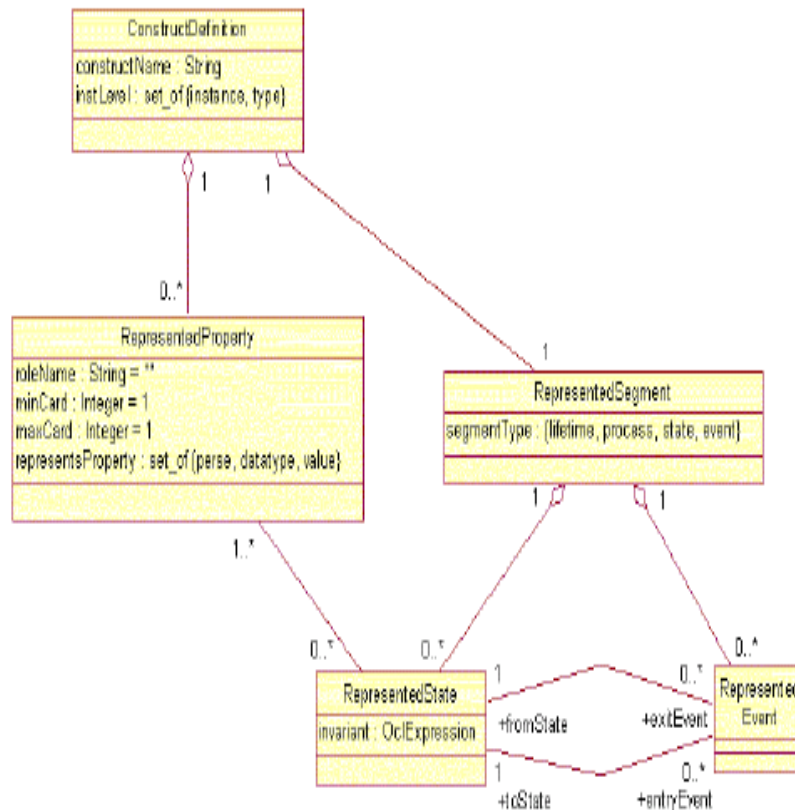
The fourth type of entry is used to define which part of the lifetime of a thing that the modeling construct may represent. Sometimes different modeling constructs may represent the same class of things and the same properties of those things but different segments of the lifetimes of those things. For example, one construct may represent an event, another a state and a third a process, although all three constructs represent the same property of the same thing. This becomes obvious when we see that constructs that are as different as UML-state and UML-event have identical instantiation level, class and property entries. Both constructs represent the type level, may represent any subclass of the class of ChangingThings and may represent any non-law properties of those subclasses. However, they are distinguished by their lifetime entries (Opdahl & Henderson-Sellers 2004).

Figure 2-17 extends the UML class diagram to show RepresentedSegments of the lifetimes of things and classes. A ConstructDefinition has exactly one RepresentedSegment, which is either the whole lifetime of the thing or class, a process, a state or an event. RepresentedSegments that are states or events must also have a RepresentedState and/or a RepresentedEvent as parts. A RepresentedState is described by an oclExpression that involves RepresentedProperties. A RepresentedEvent is defined in terms of its from- and toStates. BWV processes are represented as chains of RepresentedStates and -Events (Opdahl & Henderson-Sellers 2004).

Although the Opdahl and Henderson-Sellers (2004) template does not account for all the BWV-concepts presented in other papers (e.g., Wand & Weber, 1988, 1993, 1995), it accounts for all the basic concepts, so that modeling constructs defined in terms of the template should also be implicitly related to the rest of the BWV model. However, the need for further research in order to extend the template with

more BWV concepts is acknowledged as part of Opdahl and Henderson-Sellers (2004) research.

**Figure 2.17 UML class diagram extended to show the lifetime entry**



(Source: Opdahl & Henderson-Sellers 2004)

Another alternative to generate normalized reference models in BWV terms is the Green and Rosemann (1999) BWV meta model. This will be discussed in detail in the next section.

### 2.9.2 The Green and Rosemann BWV meta-model

The Green and Rosemann (2000) BWV meta-model is based on the original E-R specification from Chen (1976) with extensions made by Scheer (1998). This version is called the extended ER-model (eERM). The meta-model identifies 28 main constructs in the BWV model (Green & Rosemann 2000).

The Green and Rosemann meta-model has been already used for the comparison of ontologies (Davies et al. 2002) and it has been proposed as a tool to generate normalized reference models (Rosemann & Green 1999).

Davies et al. (2002) proposed a set of information objects that can be useful when comparing reference models built with the Green and Rosemann meta-model (1999):

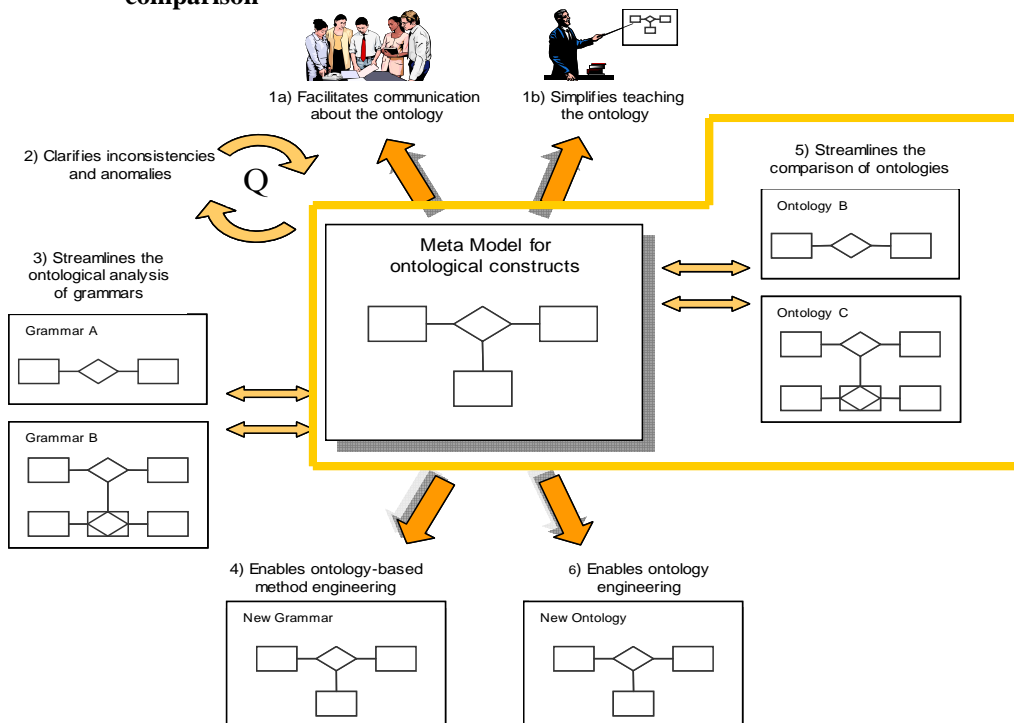


- Entity types: The comparison of the number and kind of entity types provides the most essential information for the comparison of meta-models. Within a given degree of abstraction, the width of an ontology increases with the number of entity types in the meta-model.
- Relationship types: Another metric concerning the integration within an ontology is the number of relationship types. The structural density of an ontology increases with the number of relationship types if the number and kind of entity types stay the same.
- Beyond entity and relationship types, the comparison of cardinalities and attributes typically provides further information.

Independent from entity types and relationship types, three different situations can be distinguished when comparing meta-models (Davies et al. 2002):

- Between two corresponding elements in two ontologies there might be a 1-1 relationship. This case describes ontological equivalence.
- It might also be the case that one element in an ontology is further specified by two or more elements in the other ontology.
- Finally, it might be the case that one element in one ontology does not have any correspondence in the other ontology at all.
- The meta-model has also been used to model the ARIS process modeling grammar (Rosemann & Green 2002) so there is previous work that could be used to build upon for this dissertation.

**Figure 2.18 The application of Green and Rosemann (1998) metamodel for ontologies comparison**



(Source: Davies et al. 2002)

The Green and Rosemann (1998) meta-model seems to be a better fit for this dissertation. Although the Opdahl and Henderson-Sellers (2004) model is able to represent IS models in BWB terms, it does not account for all the BWB-concepts

(Opdahl & Henderson-Sellers 2004) while the Green and Rosemann (1998) meta-model identifies 28 main constructs in the BWW model (Green & Rosemann 2000). Also, the Green and Rosemann meta-model (1998) has been used to generate normalized reference models (Rosemann & Green 1999) and the methodology to compare Green and Rosemann (1998) meta-models for ontological equivalence has been developed by Davies et al. (2002), making it an attractive option as this foundation is required for this dissertation research (Figure 2-18).

## **2.10 Conclusions**

The literature review started with the concept of legacy systems and their importance to business given their size, stability and importance to business operations. As these legacy systems are crucial to the operation of business, it is important to be able to transfer all the requirements that these systems have captured during many years of operation when re-engineering these systems in order to adapt them to component based architectures that can cope with more modern technologies.

The definition of a requirements model and its role to represent the business requirements of an information system were covered as part of this literature review.

The traditional approach of requirements modeling and component-based modeling were covered and the two major approaches (UML and Catalysis) for component model generation were compared and contrasted. Component-based modeling was also compared and contrasted with the traditional approach of requirements modeling.

Component-based re-engineering was explained and three re-engineering methodologies were reviewed and compared. The Deursen et al. (2000) re-engineering methodology was highlighted as a strong way of recovering legacy systems requirements models but weak to generate component-based models. On the other hand, the Draco methodology (Fontanette et al. 2003) is able to generate the component-based models necessary for this research by using case tools but unable to recover legacy requirements models. Finally, the Jacobson and Lindstrom (1991) approach for re-engineering of legacy systems was proposed as a good fit for this research because it includes cases of a complete change of implementation technique and no change in the functionality and covers reverse engineering. Although this last one was originally designed for object-oriented re-engineering, it could be easily adapted to component-based (Fevre et al. 2003).

The BWW model was detailed and its relevance to information systems was explained. The Fettke and Loos (2003) approach to the ontological evaluation of reference models was described and its relevance to compare requirements models was explained. Mappings for process modeling and UML models to BWW constructs were introduced and analyzed.

One of the biggest weaknesses of the Fettke and Loos (2003) methodology is its lack of methodology for transformation of requirements models into BWW models. Although this is essential to compare two requirements models for equivalency of representation, Fettke and Loos (2003) did not include this in any detail.

Two major approaches for transforming information systems models into BWW models were reviewed and contrasted. Although the Opdahl and Henderson-Sellers (2004) template could be used as a platform for model comparison in the Fettke and Loos (2003) methodology, the Green and Rosemann (1998) meta-model is a better fit for this research given the previous research work that shows its use for requirements model comparison (Davis et al 2002).

The presented literature review described all the necessary elements to conduct the research for this dissertation. Also, this review justifies this study given the size of the legacy systems in the present economy that require re-engineering in order to adapt to modern needs. The literature review also reveals a gap of research dealing with ontological evaluation of re-engineered systems: although many researchers have found many uses of ontological evaluation in the modern information systems, very little research has been conducted on the use of ontologies for evaluation of re-engineered systems.

In the next section, the research methodologies used for this study are introduced and justified. In addition, issues with validity and reliability are also discussed.

## Chapter 3 Research Methodology

### 3.1 Introduction

This research is applied in nature. Instead of building new theoretical constructs, this research has the objective of using existing component-based constructs in order to compare their capability of representing the same requirements in component-based requirements models when re-engineering legacy systems modeled with traditional constructs. Very few research frameworks for applied research in information systems have been developed in the past. However, design science, a scientific research method applied to information systems (March & Smith 1995) can be used in developing an evaluation of requirements models framework.

Applied to this dissertation, design science means designing an evaluation of requirements models' frameworks that helps IS specialists in the verification of representation of the business requirements in re-engineered component-based requirements models originally represented in legacy requirements models.

March and Smith (1995) define design science as an attempt to create things that serve human purposes, as opposed to natural and social sciences, which try to understand reality (Au 2001).

March and Smith outline a design science framework with two axes, namely research activities and research outputs. Research outputs cover constructs, models, methods and instantiations. Research activities comprise building, evaluating, theorizing on and justifying artifacts.

Constructs or concepts form the vocabulary of a domain. They constitute a conceptualization used to describe problems within a domain. A model is a set of propositions or statements expressing relationships among constructs. In design activities, models represent situations as problem and solution statements. A method is a set of steps (an algorithm or guideline) used to perform a task. Methods are based on a set of underlying constructs (language) and a representation (model) of the solution space. An instantiation is the realization of an artifact in its environment. Instantiations operationalize constructs, models and methods.

Concerning research activities, March and Smith (1995) identify build and evaluate as the two main issues in design science. Build refers to the development of constructs, models, methods and artifacts demonstrating that they can be constructed. Evaluate refers to the development of criteria and the assessment of the output's performance against those criteria. Parallel to these two research activities in design science, March and Smith add the natural and social science couple, which are theorize and justify. This refers to the construction of theories that explain how or why something happens. In the case of IT and IS research this is often an explanation of how or why an artifact works within its environment. Justify refers to theory proving and requires the gathering of scientific evidence that supports or refutes the theory (March & Smith 1995).

The use of the design science research framework is justified for this research for the following reasons:

- It provides a framework that can be used for information systems applied research. This is in line with Nunamaker, et al. (1990) who classify design science in IS as applied research that applies knowledge to solve practical problems.
- It provides a framework for evaluation of models. The objective of this research is the evaluation of the capacity of component-based requirements models to represent business requirements of legacy requirements models. This framework seems to be aligned with this objective.
- The framework can be used to extend the scope of this research. Although the objective of this research is not to create new theory based on the findings, the framework provides that possibility and could be used for future research.

### **3.2 Research outline for this dissertation**

The research in this dissertation is based on the design science framework detailed above and essentially covers the build and some evaluate research activities and has a research output of constructs and models. Instantiations are not covered as the scope of this research is limited to requirements models. Requirements models do not include any implementation details that can be used for instantiation.

As illustrated in Table 3-1, March and Smith (1995) propose a four by four framework that produces sixteen cells describing viable research efforts. The different cells have different objectives with different appropriate research methods. A research project can cover multiple cells, but does not necessarily have to cover them all.

The build part of the framework will be used as part of this research as requirements models need to be created for ontological evaluation. The main activity of this research will be the evaluation as it will allow us to identify metrics to compare the performance of constructs and models.

Table 3-1 illustrates which cells at the intersection of research activities and research outputs of March and Smith's (1995) framework are covered by this dissertation. Each cell/intersection contains a specific research objective of the overall research. The build column covers the recovery of a requirements model for a legacy system and the generation of a re-engineered component-based requirements models. Construct building is not required as existing constructs for both traditional and component-based will be used.

The evaluate column includes evaluating the completeness of the component-based constructs (UML) in terms of ontological deficiencies that the constructs could have to model traditional constructs. Requirements models need to be evaluated in order to measure the component-based requirements model capacity of representing the same requirements as the legacy requirements model.

**Table 3.1 Research activities based on the Design Science Framework**

	<b>Build</b>	<b>Evaluate</b>	<b>Theorize</b>	<b>Justify</b>
Constructs	Not required	Identifying ontological modeling deficiencies of component-based constructs in terms of traditional construct representation	Not required	Not required
Model	Recover the legacy requirements model of the case study  Generate the re-engineered component-based requirements model for the legacy system	Evaluate the capacity of the re-engineered component-based for representing the same business requirements embedded in the legacy requirements model	Not required	Not required
Method	Not required	Not required	Not required	Not required
Instantiation	Not required	Not required	Not required	Not required

### **3.3 Methodologies applied to the Design Science Framework**

In the previous section we explained the research objectives in the different cells of March and Smith's (1995) framework covered by this dissertation. March and Smith warn that every cell and research objective may call for a different methodology. This makes it necessary to identify an adequate method for each specific research objective, resulting in an overall method mix. To achieve this, several methodologies were identified as part of the literature review. These methods are identified in Table 3-2.

**Table 3.2 Methodologies used in this research**

<b>Methodology</b>	<b>Definition</b>
Case Study	Study of a single phenomenon (e.g., an application, a technology, a decision) in an organization over a logical time frame.
Jacobson & Lindstrom (1991)	Methodology for information systems re-engineering and legacy system requirements model recovery
Fettke & Loos (2003)	Methodology for ontological evaluation of requirements models
Interviews	Research in which information is obtained by asking respondents questions directly.
Direct observation	This occurs when a field visit is conducted during the case study
Secondary Data	A study that utilizes existing organizational and business data, e.g., document, diagrams, etc.
Rosemann & Green (2002)	Meta-models methodology for Normalized Reference Models generation and comparison

Table 3-3 illustrates which of the retained methodologies was applied to each cell and accordingly to which research objective.

The research starts with a description of the case study company, its organizational structure, main business services and client base. Further, the output of this research involves three main parts: requirements model recovery, system re-engineering and ontological evaluation.

The requirements model recovery of the case study is one of the major challenges in the research because most of the legacy systems have very poor documentation in terms of requirements models and technical design. In order to address this problem, the researcher captured the requirements model of the legacy system by applying a reverse engineering approach as specified in the Jacobson and Lindstrom (1991) methodology. This methodology uses data collection methods including interviews, direct observation and secondary data.

Once the requirements models from the legacy system are recovered, the system was re-engineered using the Jacobson and Lindstrom (1991) approach for re-engineering of legacy systems. The output of this step is the re-engineered component-based requirements model.

After the legacy system and re-engineered requirements models are generated, they are evaluated based on their ontological evaluation of grammars (Wand & Weber 1993). An ontological normalization for the original and re-engineered requirements models is generated. The two models are evaluated using the Fettke and Loos (2003) methodology based on their ontologically normalized models generated using the Rosemann and Green (2000) methodology. The resulting comparison is that the compared models are ontological equivalent, complementary or in conflict (Fettke & Loos 2003).

In the following section, the justification and contribution for each of the chosen methodologies is explained in detail.

**Table 3.3 Research methodologies used for the Design Science Framework**

	<b>Build</b>	<b>Evaluate</b>
Constructs		Fettke & Loos
Model	Case Study	Case Study
	Interviews	Fettke & Loos
	Secondary Data	Rosemann & Green
	Direct Observation	
	Jacobson & Lindstrom	

### **3.4 Justification of the Case Study Research Methodology**

Case study methodology is appropriate for this research given its “exploratory nature” (Benbasat et al. 1987) and because it is the most commonly applied qualitative, positivist, method in information systems research (Orlikowski & Baroudi 1991; Alavi & Carlson 1992).

Also, case studies are “especially useful in situations in which a complicated series of variables interact to produce the problem” (Kinneer & Taylor 1996 p. 176). Information systems re-engineering is complex and it does not have a predefined set of controlled variables that can be identified for all the possible cases. This makes quantitative research almost impossible and case study research more viable as a research methodology because the investigator may not specify the set of independent and dependent variables in advance (Benbasat et al. 1987).

A case study enables the researcher to deeply study information systems in the real environment of the study object rather than in a simulated environment. Another characteristic of case study research is that data is collected by multiple means (Benbasat et al. 1987). This is relevant for this research because data is collected from user documentation, observation and open interviews with the maintainers. This multiple data source also makes the study more reliable.

Only one or few entities (person, group or organization) are examined in a case study (Benbasat et al. 1987). This makes case study methodology a good option for information systems re-engineering research as it simplifies the research and allows the researcher to concentrate on one organization. By concentrating on only one organization, the researcher can study the complexity of the re-engineering process intensively.

Finally, the focus of case study research is on contemporary events (Benbasat et al. 1987) and systems re-engineering is current and expected to grow rapidly.



In summary, the case study methodology is appropriate for this research because it is suitable for exploratory research, allows multiple sources of data collection to improve the validity of the study and allows the natural study of information system re-engineering.

### **3.5 Justification of the Jacobson and Lindstrom Methodology for Information Systems Re-engineering and legacy system requirements model recovery**

One of the major challenges of this research is to reconstruct the legacy requirements model so it can be compared with the re-engineered requirements model. The Jacobson and Lindstrom (1991) methodology is relevant for this research as it covers the reverse engineering techniques necessary to recover the original requirements model for the legacy system.

Although other re-engineering methodologies such as the Deursen et al. (2000) methodology allow the recovery of the legacy requirements model based on the analysis of source code, this can represent a problem for information systems that consist not only of software developed in-house but also involve the integration of off-the-shelf software packages and custom software development. Furthermore, the source code analysis for legacy requirements model recovery is a complex process that requires a sound knowledge of the legacy system that sometimes is not available in the organization. One of the main reasons for the selection of the Jacobson and Lindstrom (1991) methodology is that it does not rely exclusively on the analysis of source code but on other description elements such as the requirements specifications, user operating instructions, maintenance manuals, training manuals, design documentation and database schema descriptions. This makes the recovered requirements model more reliable given the different sources required for its generation.

One of the constraints of this research is to compare requirements models from the legacy and re-engineered system for the case of no change in functionality as this allows the comparison of ontological equivalent models that should reflect exactly the same requirements. The Jacobson and Lindstrom (1991) methodology considers the case of no change in functionality, which is the case of this research.

Reduction of complexity of the research is another reason for using the Jacobson and Lindstrom (1991) methodology. Most of the reverse engineering and re-engineering methodologies rely on complex methods such as cluster analysis and source code analysis while the research methodology chosen here is straightforward and does not require complex calculations and analysis tools.

There is also documented evidence that this methodology has worked well for several industries such as the telecommunications, aerospace and defense industries (Jacobson & Lindstrom 1991).

In summary, the Jacobson and Lindstrom (1991) approach for re-engineering of legacy systems is selected because it contemplates cases of a complete change of

implementation technique and no change in the functionality, it does not rely solely on the source code, it covers reverse engineering and it is relatively simple to use.

### **3.6 Justification of the Fettke and Loos methodology for Ontological Evaluation of Requirements models**

The objective of the research is to evaluate the requirements models of the legacy and re-engineered information systems for equivalency of representation of business requirements. Several research methodologies for evaluation of requirements models have been developed in the past for this purpose. Fettke and Loos (2003) classified research methods for requirements model evaluation into analytical and empirical approaches. Analytical approaches are based on logical conclusions while empirical approaches are based on experiences. Both approaches can be differentiated by quality criteria, these criteria can be either ad hoc or theory driven. Theory-driven quality criteria are derived from and founded on a specific reference theory (Vessey et al. 2004) whereas ad hoc quality criteria are introduced for the purpose of the evaluation approach without referring to a specific theory.

Although approaches to evaluate requirements models such as those of Mišić and Zhao (2000) and Schütte (1998) could have been used to compare the legacy and re-engineered requirements models, the problem with these feature-based evaluation approaches is that the development and selection of a specific feature set is often a subjective issue as they are driven by the ambiguous ad hoc quality criteria. On the other hand, the Fettke and Loos (2003) approach is based on the Bunge-Wand-Weber ontology theory and therefore a much better choice for requirements model evaluation given that this theory is well founded on mathematical concepts and has shown promising results for research on the evaluation of grammars (Evermann & Wand 2001b; Green & Rosemann 2000; Opdahl & Henderson-Sellers 2002a; Weber & Zhang 1996).

In addition, the Fettke and Loos (2003) approach was selected because it allows comparing requirements models that are represented with different grammars which is the main objective of this research.

In summary, the use of the Fettke and Loos (2003) approach is justified because it is BWB theory driven and a simple analytical approach that provides a mechanism for comparison of requirements models.

### **3.7 Justification of the Rosemann and Green meta-models for normalized reference models**

As part of the Fettke and Loos (2003) methodology, the requirements model needs to be transformed to an ontological model. The outcome of this step is an ontologically normalized reference model. In order to generate these normalized reference models in BWB terms, the Rosemann and Green (2000) BWB meta-models will be used. This meta-model is based on the original E-R specification from Chen (1976) with extensions made by Scheer (1998). The use of the Rosemann and Green (2000) meta-models is justified for the following reasons:

Since Chen (1976) introduced the original E-R approach, it has undergone intensive discussions and further developments. It is realistic therefore to expect that solutions for special methodological problems that could occur during the process of designing the meta-model are already available in most cases.

Within the communities of computer science and information systems many potential meta-languages are available. The E-R approach is widely accepted as a de facto standard for modeling. Evidence for this situation is that the annual international conference on E-R modeling has been organized for the past 20 years. Several meta-models based on the E-R approach are already available. Among others, Scheer (2000) uses an E-R-based meta-language to explain his Architecture of Integrated Information Systems (ARIS).

### **3.8 Procedures**

There are two types of research procedures: build and evaluation. Build procedures are required to accomplish the build objectives of the design research framework while the evaluation procedures accomplish the evaluation objectives.

#### **3.8.1 Data Collection (Build)**

Data gathering is an important part of this research as it is required to commence the building part of the research. Stake (1995) and Yin (1994) have identified seven sources of empirical evidence in case studies:

1. Documents: Written material sources.
2. Archival records: Archival documents can be service records, organizational records, and lists of names, survey data, and other such records.
3. Interviews: An interview can be used for three purposes: as an exploratory device to help identify variables and relations, as the main instrument of the research and as a supplement to other methods (Kerlinger 1986).
4. Questionnaires: These are structured questions written and supplied to a large number of respondents, commonly spread over a large geographical area for consideration in advance.
5. Direct observation: This occurs when a field visit is conducted during the case study.
6. Participant-observation: Participant-observation turns the researcher into an active participant in the events being studied.
7. Physical artefacts: Physical artefacts can be tools, instruments, or some other physical evidence that may be collected during the study as part of the field visit.

For this research, interviews, observation techniques, physical artifacts and review of system documents were used for the case study. The most common methods of collecting data within the case study approach are through observation and interviews (Bell 1992). The benefits that both observation and interviews serve in the assimilation of qualitative information is noted by Gilbert (1993) and expanded upon here.

The use of observation as a method of data collection is well documented (Bell 1992, Benbasat et al. 1987, Stake 1995) and works well in case research (Yin 1994). Before observation can be used in research, three minimum conditions set out by Tull and Hawkins (1993) need to be met:

1. The data has to be available for observation
2. The behaviour has to be repetitive, frequent, or otherwise predictable
3. An event has to cover a reasonably short time span.

According to Jorgensen (1989), observation is appropriate for studies of almost every human existence. Through observation, it is possible to describe what goes on, who or what is involved, when and where things happen, how they occur, and why things happen as they do in particular situations (Jorgensen 1989). A great deal of time is spent on paying attention, watching and listening carefully (Neuman 1994). The observer uses all the senses, noticing what is seen, heard, smelled, tasted and touched (Neuman 1994; Spradley 1979).

According to Neuman (1994), there are four possible research stances for the participant observer:

1. Complete participant: the researcher operates under conditions of secret observation and full participation.
2. Complete observer: the researcher is behind a one-way mirror or in an invisible role that permits undetected and unnoticed observation and eavesdropping.
3. Participant as observer: the researcher and members are aware of the research role, but the researcher is an intimate friend who is a pseudomember.
4. Observer as participant: the researcher is a known, overt observer from the beginning, who has more limited or formal contact with members.

The case selected is a legacy systems which uses a centralized mainframe platform. Such a platform is representative of the platforms on which many legacy-based systems operate. A software house produces a home loan software system implemented at a number of banks and insurance groups. The system operates on the Unisys A-Series mainframes. The case-study's system is a customised version of the home loan system product implemented at a mid-sized home loan bank in the Netherlands that specializes in the marketing, sales and administration of its own home loan products. This bank is the case study company for this research. The case study system was implemented in the mid 1980's and is maintained by a team of two software developers. The system has been modified many times in order to reflect changes in the business processes of the organization; however, there is no formal documentation of these changes.

The technique used to interview maintainers was open-ended interviews. The use of this technique is justified for this research by two main reasons. First, the goal is to elicit the respondent's views and experiences in his or her own terms, rather than to collect data that are simply a choice among pre-established response categories

(Anderson et al. 1994). Secondly, the interview is not bound to a rigid interview format or set of questions that would be difficult to establish given the nature of the research and will limit the results (Anderson et al. 1994).

System documentation was collected in order to perform the reverse engineering analysis required to recover the requirements models (Jacobson & Lindstrom 1991). Jacobson and Lindstrom (1991) suggests that the legacy information system can be described by using different elements as requirements specifications, user operating instructions, maintenance manuals, training manuals, design documentation, source code files, and database schema descriptions (Jacobson & Lindstrom 1991).

For this case study, the following documents were collected to describe the information system:

1. Architecture documentation: The diagrams are included in Appendix A and include Sub-system flow (geographic) diagram, Sub-system process flow diagram, System architecture for the procedural model diagram, batch process program flow diagram, and screen diagrams. These diagrams have not been updated since the original implementation of the legacy system that started in the mid 1980's and ended in 1987. These diagrams were developed by the software development firm that customized the bank's legacy system.
2. Database schema: The logical data model of the legacy system was collected and used to generate the data model for this dissertation. This document was created by the software development firm that implemented the legacy system.
3. Manuals: User manuals for the legacy system. These manuals have not been updated to reflect modifications.

### **3.8.2 Requirements model Recovery (Build)**

The reverse engineering methodology, as specified in Jacobson and Lindstrom (1991) was applied to capture the requirements model of the legacy system. The following steps were used:

- Develop a concrete graph that describes the components of the system and their interrelationship.
- Develop an abstract graph showing the behavior and the structure of the system.
- Develop a mapping between the two, i.e. how something in the abstract graph relates to the concrete graph and vice versa.

The abstract graph should be free of implementation details. For example, mechanisms for persistent storage or partitioning into processes should not appear on this graph. The concrete graph must, on the other hand, show these details. The mapping between the two should explain how the abstract graph is implemented by way of the concrete graph (Jacobson & Lindstrom 1991).

As described in section 3.8.1, the components of the legacy system that were collected to describe the system were:

- Database schemas
- User manuals
- Architecture documentation
- Observation of the system
- Open Interviews with users and technical experts

Use cases were used to develop the concrete graph for reverse engineering. Use cases are an excellent tool for reverse engineering as they provide a sequence of user interactions with the system (Jacobson & Lindstrom 1991). Their purpose is to define a typical way of using the system and to describe the business process, which document how the business works and what the business goals are of each interaction with the system. In the context of reverse engineering, it is possible to explore an old system with use cases (Jacobson & Lindstrom 1991).

The use cases developed show the interrelationship between manuals, documentation, interviews, source code and researcher's observation of the system. The abstract graph described in the Jacobson and Lindstrom (1991) methodology is in fact an example of a legacy requirements model. For the legacy system in this study, the requirements model was represented in terms of data flow diagrams, a context model and entity relationship diagrams.

The description of the business process, business events and responses is essential in generating a requirements model (Whitten et al. 2001). The use cases used to construct the concrete graph, document the business processes, events and responses required to construct this legacy abstract graph. In order to generate the DFDs required to construct the legacy requirements model, business events to which the system must respond and appropriate responses were identified with the help of the use cases. Essentially, there are three types of events (Whitten et al. 2000):

1. External events: are so named because they are initiated by external agents. When these events happen, an input data flow occurs for the system in the DFD.
2. Temporal events: trigger processes on the basis of time. When these events happen, an input called control flow occurs.
3. State events: trigger processes based on a system change from one state or condition to another.

Information systems usually respond to external or temporal events. State events are usually associated with real time systems (Whitten et al. 2000).

Once these events were identified, DFDs were drawn using Microsoft Visio 2000 with the help of the list of mapping transformations suggested by Whitten et al. (2000). This mapping shows how the concrete graph represented by the use case can be mapped into the abstract graph represented by the DFD. The list of recommendations is:

- The actor in the use case that initiated the event will become the external agent.
- The event identified in the use case will be handled by a process in the DFD.
- The input or trigger in the use case will become the data or control flow in the DFD.
- All outputs and responses in the use case will become data flows in the DFD.

DFD models were reviewed with the software developers in charge of the maintenance of the case-study legacy system in order to verify its accuracy. The data model of the legacy requirements model was generated by identifying the data stores in the DFD, examining the use cases and database schemas, and documented using an entity relationship diagram.

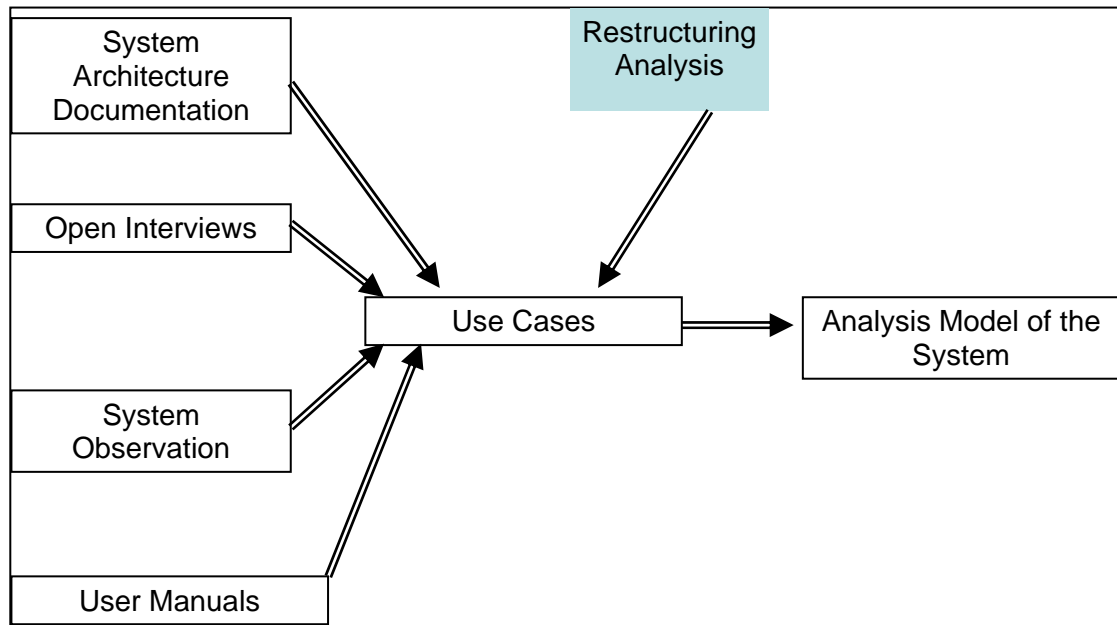
### **3.8.3 Component-based requirements model generation (Build)**

Once the requirements model was reverse engineered from the legacy system, the legacy system was re-engineered for a complete change in implementation technique but no change in functionality by using the following steps (Jacobson & Lindstrom 1991):

- Prepare an analysis model.
- Map each analysis object to the implementation of the old system.

In the first step, an analysis model was prepared with the help of the use cases prepared in the reverse engineering process. These use cases already contain the information that was assimilated from the manual, system architecture documentation, open interviews and research observations described as description elements in the Jacobson and Lindstrom (1991) methodology shown in Figure 3-1. Only the analysis model of the re-engineering process was required as this research's primary objective is the comparison of requirements models and it is not concerned about the full implementation of the information systems.

**Figure 3.1 Preparation of the analysis model**



(Adapted from Jacobson & Lindstrom (1991))

An analysis model only contains the logical aspect and is free of physical implementation details. The logical representation of a component is concerned with its logical abstraction, its relationship with other logical elements, and its assigned responsibilities. The logical representation of a component-based system was modeled using the following UML diagrams (Houston & Norris 2001):

- Use case diagrams
- Class diagrams
- Sequence diagram
- State diagrams.

Actors were identified from the use cases and use case diagrams were constructed to identify the system scope and boundaries. The requirements model should be free of physical implementation details; for the case of components, their logical representation is modeled using UML subsystems and identified inside the use case diagrams as proposed by Houston and Norris (2001). Class diagrams are prepared using the criteria for finding objects that are described in the object-oriented method described by Jacobson (1987). This step is accomplished by reviewing each use case to find nouns that correspond to business entities or events (Jacobson 1987). Not all the nouns in the use cases represent valid business objects. The list of nouns is cleaned by removing nouns that represent synonyms, nouns outside of the scope of the system, nouns that are roles without unique behavior or are external roles, unclear nouns that need focus or nouns that are really actions or attributes (Whitten et al. 2000). Once objects are identified, their relationships were modeled as part of the class diagrams and interfaces were identified.

All subsystems are said to have a state that is the value of its attributes at one point in time (Whitten et al. 2000). The change in state is triggered by an event. A state diagram depicting the different state a subsystem can have, the events that cause the



subsystem to change state over time, and the rules that govern the subsystem's transition between states is generated for all the subsystems that have identifiable states and complex behavior. Once state diagrams are generated, sequence diagrams are completed in order to show how the subsystem's elements implement the major interface operations (Houston & Norris 2001).

After the UML diagrams that represent the component analysis model are completed, a mapping of each analysis object to the implementation of the old system is conducted as suggested by Jacobson and Lindstrom (1991). The map showed that all analysis objects and dependencies were motivated by at least one primitive description element.

### **3.8.4 Ontological Evaluation (Evaluation)**

Once the legacy requirements model is recovered and the component business analysis model represented with the use of UML diagrams, the Fettke and Loos (2003) methodology is used to evaluate these models for equivalency of representation of business requirements.

As part of this evaluation, the ontological normalization of the legacy and re-engineered component requirements models is generated. The ontological normalization of a reference model consisted of four steps (Fettke & Loos 2003):

1. Developing a transformation mapping,
2. Identifying ontological modeling deficiencies,
3. Transforming the requirements models, and
4. Assessing the results.

In the first step of this method, a transformation mapping for the traditional and component-based (UML) diagrams used for representing the requirements models is developed. This transformation mapping allowed converting the constructs of the traditional and component based (UML) diagrams to the constructs of the BWW model. The first step is based on the method for the ontological evaluation of grammars proposed by Wand and Weber (1993).

The transformation mapping consisted of two mathematical mappings. First, a representation mapping described whether and how the constructs of the BWW model are mapped onto the traditional and component based (UML) constructs. Second, the interpretation mapping described whether and how the traditional and component based (UML) constructs are mapped onto the constructs of the BWW-model (Fettke & Loos 2003).

All ontological deficiencies of the requirements models are identified as part of the second step of the generation of the normalized ontological requirements models. To identify the ontological deficiencies of the recovered requirements model and re-engineered component-based requirements model, all constructs of the models are reviewed. Each construct of the requirements models analyzed is examined with respect to whether the construct is used correctly regarding the interpretation mapping.

Deficiencies are classified as (Fettke & Loos 2003):

- **Adequacy:** The grammatical construct is ontologically adequate. Nevertheless an ontological deficiency can emerge by applying the grammatical construct to build the reference model. Therefore it must be examined whether the construct of the reference model is used correctly with respect to the interpretation mapping. The construct of the reference model is used adequately if it is used correctly with respect to the interpretation mapping.
- **Excess:** Construct excess is a modeling deficiency in general and needs a special handling in the transformation step. Therefore, this construct should be marked as excessive in the reference model.
- **Overload:** Construct overload is a modeling deficiency in general and needs a special handling in the transformation step. Therefore, this construct should be marked as overloaded in the reference model.

Based on the representation mapping it is decided whether the traditional and component-based grammar are incomplete or redundant. An incomplete grammar suggests that specific facts of reality cannot be adequately represented in the requirements model.

In the third step, the requirements models are transformed to ontological models. The outcome of this step is two ontologically normalized requirements models. The objective of both techniques is to represent the domain of interest in a normalized way by applying specific transformation patterns (Fettke & Loos 2003).

The two models are compared based on their ontologically normalized models. The result of this comparison is an analysis that revealed if the compared models are ontological equivalent, complementary or in conflict. In order to generate these normalized reference models in BWW terms, the Rosemann & Green (2002) BWW meta-models are used.

### **3.9 Validity and Reliability**

Peräkylä (1997 p. 206) states, "the issues of reliability and validity are important, because in them the objectivity of research is at stake". Enhancing objectivity ensures the accuracy of the results and findings from research.

Reliability is the extent to which a procedure will produce the same results under constant conditions (Bell 1992, Kirk & Miller 1986). In the case of this study, the reliability of the research results entailed whether or not the same findings would occur if the study was repeated in the same manner.

Great care has been taken in the planning, implementing and analysis stages to ensure reliability. Benbaset et al. (1987) state that a clear description of the data sources and the manner in which they contribute to the overall findings of a study is an important aspect of the reliability and validity of the results. For this reason, a clear description of the data sources and methods used to gather those sources is provided in section 3.8.1.

Data collected using interviews is open to problems such as interview bias, misdirected prompting and issues of question wording. With regard to the results from observation techniques, some of the issues that typically affect the reliability of observation results are potential recorder bias and obtrusive influence (Benbaset et al. 1987).

Audio tapes were used to increase reliability, interviews were audio recorded. Peräkylä (1997) states that using tapes and transcripts eliminates many of the problems associated with the recording of qualitative information, specifically field notes and the limited public access to them. However, Peräkylä (1997) warns researchers of several important factors affecting the reliability of tape recordings and transcripts. These include:

- The decision of how much to record
- The technical quality of the recordings
- The adequacy of transcripts
- The inclusion of vocal expression in initial transcripts.

These issues were considered and it was decided to record and transcribe all interviews completely thus providing a wide scope and full database of information from which to extract information. The two interviews conducted were fully transcribed in detail, noting and including the many aspects of body language of interviewees. This was in accordance to Peräkylä (1997) who notes that:

A rich transcript is a resource of analysis: at the time of transcribing, the researcher cannot know which of the details will turn out to be important for the analysis. (Peräkylä 1997, p. 207).

The transcript was sent back to the interviewees for confirmation. With regard to the results from observation techniques, the issues of reliability are somewhat easier to assess than data collected via interviews. Because much of what was observed was inanimate and static (such as technological deployments, report printing, data entry) the issues that typically affect the reliability of observation results such as potential recorder bias and obtrusive influence did not apply. Subsequently, these observations have high reliability.

Validity describes whether an item measures or describes what it is supposed to measure or describe (Bell 1992). It is a much more complex concept than reliability and there are many variations and sub-divisions to which researchers can investigate in attempts at ensuring validity of their results. Bell (1992) states that researchers involved in smaller projects without complex testing or measurements need not investigate the concept of validity too thoroughly but should examine results and methods critically. Noting this, a brief examination of two aspects of validity is provided.

To increase validity and to ensure accuracy, follow-up e-mail was used to discuss and clarify topics of discussion. This ensures that what was stated in the research is factual and accurate.

Another aspect of validity relates to the generalisability of research findings. The results of this research were produced from a single case study. The generalisability of the conclusions can be compromised due to this limitation.

Exemplary case study design ensures that the procedures used are well documented and can be repeated with the same results (Soy 1996). According to Burns (1994), research can be considered internally valid if the author demonstrates that the changes indicated by the analysis of a problem situation constitute an improvement to it. Internal validity normally applies to explanatory and causal studies, but not to a descriptive or exploratory study such as this one (Pervan 1996). The researcher attempts to ensure internal validity for the case study by:

- Conducting the interviews by the same researcher to avoid variations in administration of the instrument.
- Assuring the respondents of anonymity and confidentiality of the data to ensure that data gathered were accurate and unbiased.
- Retaining original data such as interview recordings, interview transcripts, and field notes.
- Allowing the respondents to choose time and place of interviews and interview time.
- Allowing the respondents to choose comfortable and familiar surroundings for interview.
- Using triangulation of sources and methods (e.g. open interview, observation, documents, source code).

According to Gable (1994) and Jick (1979), triangulation involves the use of multiple techniques within a given method to collect and interpret data. It also increases the reliability of the data and the process of gathering it as well as serving to corroborate the data gathered from other sources (Tellis 1997a; 1997b). Exclusive reliance on one method may bias or distort the researcher's picture of the particular slice of reality the researcher is investigating (Burns 1994). Bias may be in the form of perceptual deceptions or distortions (Remenyi & Williams 1996). Although it cannot be eradicated, bias can be minimized by the use of techniques such as triangulation (Remenyi & Williams 1996).

For this research, triangulation was performed by collecting data from open interviews, manuals, architecture documents, database schemas and observation techniques.

### **3.10 Ethical Considerations**

Ethical issues to consider when carrying out this research are noted in this section.

Software code is a considerable investment that must be protected against theft and plagiarism. There was an agreement between the researcher and the owner of the information system of the case study that the source code should not be made public, therefore the source code will not be included in this dissertation.

The researcher had access to the information systems database. This database contains many sensitive customer details such as credit records, addresses and other

personal information. The personal information contained in this data raises privacy issues, and any existing, applicable privacy legislation must be adhered to, such as Canada's Privacy Act 1988.

Different software packages have been used in the preparation of this dissertation. All the software used in the preparation of this work was properly licensed in order to conform to copyright laws.

### **3.11 Conclusions**

To address the research question, the design science framework was chosen and justified as the main methodology framework for this research. This framework requires two main activities: build and evaluate, which require the use of several methodologies. The case study methodology was chosen and justified as the main methodology to build and evaluate requirements models. The Jacobson and Lindstrom (1991) approach for re-engineering of legacy systems was chosen and justified as the methodology to build the original requirements model of the legacy system under research and for the building of the component-based re-engineered requirements models.

The methodology by Fettke and Loos (2003) was selected and justified to evaluate the requirements models generated by the reverse engineering (legacy requirements model) and those generated by the re-engineering process (component model). Traditional and component-based constructs were also evaluated under the same methodology. The Rosemann and Green (2002) meta-models were justified as the primary tool to represent the normalized requirements models in BWV construct terms. The comparison of the normalized legacy and re-engineered requirements models revealed if they represent the same requirements and helps to answer the research question of this study.

Requirements models are built with the help of data collected using interviews, observation techniques and review of information systems documents. Different methods to enhance the validity and reliability of the study are discussed and ethical considerations mentioned in order to protect the privacy of the information collected and to conform to copyright laws.

## Chapter 4 Results

### 4.1 Case Study Description

The case-study system selected is a Home Loan information system developed by a consultant company in the Netherlands. The system was customized for a mid-sized home loan bank based in the Netherlands that specializes in the marketing, sales and administration of its own home loan products. The information system was designed for use on Unisys A-Series mainframes.

The selection of this case study is justified for the following reasons. First, the proposed legacy system uses a centralized mainframe platform. Such a platform is representative of the platform on which many legacy-based systems operate. Secondly, the system contains both an on-line processing component, as well as a batch-processing component. This is a characteristic of many legacy systems, especially in the financial sector, that require an on-line entry and maintenance component and a job-scheduling component to handle high volume processing in non-peak times. Lastly, the system was developed by using the structured programming approach.

The bank fulfils a number of functions, specifically:

- Marketing and sales of its home loan products.
- The offer and acceptance of client applications via an on-line system. The client comprises both internal sales staff as well as external agents.
- The complete administration of the home loans.
- The complete administration of their in-house insurance policies.

The bank is responsible for the system enhancements, development and maintenance of the Home Loan and Insurance information System (HLIS). The HLIS was developed in the mid eighties and has been in service in the bank since then. HLIS is a contract system wherein all business property loans as well as private home loans are managed and administered. This system has its own application/offer system ensuring that an interface between a separate offer and loan system is not required. In addition to the loan account administration, all related insurance policies are managed and administered within the system. The system ensures absolute correlation between the account and the insurance administration.

The information system consists of two main business systems, namely a batch processing system and a transaction-based processing system. Most of the systems within the bank have their data stored and managed within the same shared database.

There are a number of sub-systems defined within the HLIS system:

- The loan authorization system
- The product system
- The insurance administration system
- The offer and application system

- The loan administration system
- The relation administration system (links offers with applications).

#### **4.1.1 Sub-system selection**

Due to the large scale and complexity of the system, the research focused on one sub-system representative of the main types of business processes. The sub-system focused on for this research was the Offer and Application sub-system. The Offer and Application sub-system was selected for the following reasons:

- It includes an on-line user interactive component, a procedural business flow component and a batch-processing component. This makes it a representative sub-system because these are the three main components of the system's architecture.
- The sub-system can be analyzed independently from the rest of the system as there is a clear entry point into the sub-system, which starts with an on-line application process and a clear delivery point to the rest of the system with the creation of a loan structure. This makes it ideal for the study because the researcher does not require understanding the details of the other parts of the system for the analysis.
- This sub-system is small enough to be researched within the scope of the research project.
- The researcher has access to the area where the sub-system operates. This can be seen as an advantage because the researcher can observe the sub-system's users during working hours for the preparation of the uses cases required for the re-engineering process.
- The sub-system interacts with the loan administration personnel. This makes it a representative sub-system because all the other sub-systems also interact with these actors.

#### **4.1.2 Description of the Offer and Application Sub-System**

All applications and subsequent offers for home loan products are handled by the sales department of the bank (either central or at branch offices) and independent home loan agents. The sales personnel and the loan administration personnel are the main system users.

An on-line application is made between the customer and the agent or bank sales person. In discussion with the client, a customer chooses a particular home loan product. A product can be thought of as a predefined template or blueprint containing all relevant information for a loan structure to be "manufactured" at the time of closure of the offer stage. The product gives the loan its structure and is the basis for all related entity couplings. Within the product, there can be a number of optional sub-products.

There are two distinct phases to the home loan application process. The potential client firstly needs to apply for a home loan. To facilitate the application process, the system offers the following functions:

- The entry of a new application

- Changes to application details
- Inquiry of existing application details
- Cancellation of the application
- The entry of client details
- The entry of property details.

Once an application has been completed, the bank makes the potential client an offer or number of offers. The number of offers depends on the amount of changes the client makes to the application details during the session with the sales advisor. To facilitate the offer process, the system offers the following functions:

- The entry of a new offer
- Changes to offer details
- Inquiry of existing offer details
- Copying, changing and deletion of client relation details
- The invoking of the calculation module
- Changes to the status of the offer.

The offer can progress through any of the following states from its inception:

- Registration of offer by regional office handling sale.
- Offer registration authorized by regional office.
- Offer registration declined by regional office.
- Offer document produced by regional office.
- Offer document accepted by regional office.
- Offer complete for processing by head office.
- Declined by client at regional office.
- Ready for sending offer back to regional office.
- Registration of offer by head office.
- Authorized by head office.
- Declined by head office.
- Offer document produced by head office.
- Offer document accepted by head office.
- Accepted by client.
- Offer definite.

## **4.2 Legacy requirements model recovery**

As the final goal of this dissertation is to compare the traditional and re-engineered component requirements models, it is necessary to recover the original requirements model from the legacy system.

The technique to be used for this purpose is the one proposed by Jacobson and Lindstrom (1991). The legacy information system can be described by using different elements including requirements specifications, user operating instructions, maintenance manuals, training manuals, design documentation, source code files, and database schema descriptions. The elements selected to describe this system should represent the true system, e.g. Manual.



For this case study, the following elements were collected to describe the information system:

- Architecture diagram
- Database schema
- Training manuals
- Interviews with developers
- Database scripts
- System observation

#### **4.2.1 Case Study's System Architecture**

Open interviews were conducted with the maintainers of the legacy systems in order to find out how the system was developed, what are the functions of the system and the type of documentation used for the system development. A consent letter to interview the two developers in charge of the case-study system's maintenance was mailed to the head office administrator of the bank before the interview in order to confirm their participation in the study. A copy of the consent letter and form is included in appendix G of this dissertation. Once consent was given, two interviews were conducted of one hour duration each. The interview protocol is included in appendix H and includes questions used to collect information related to the system's documentation, architecture and operation.

As part of the requirements model recovery, the system architecture of the legacy system of the case study was recovered by examining transcripts of the interviews, documentation and manuals collected from the system. A sample of these documents is in Appendix A.

Information systems architecture can be mapped to five layers (Whitten et al. 2000):

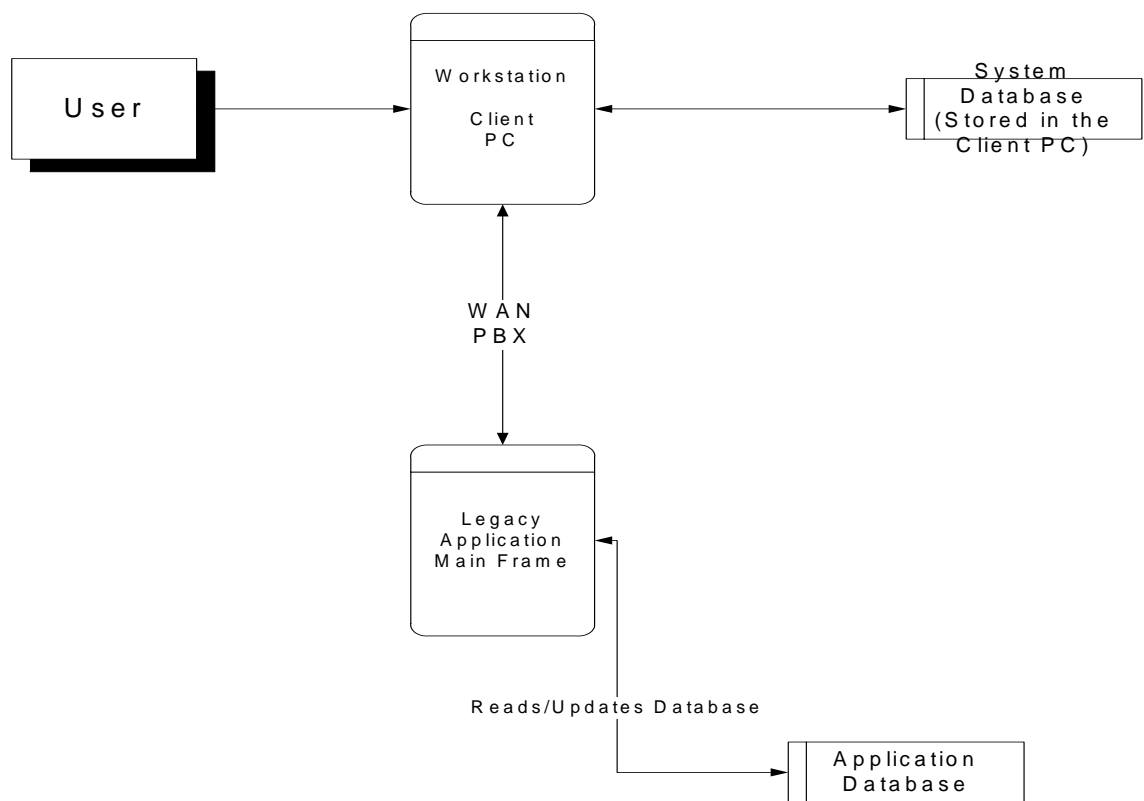
1. Presentation layer: the actual user interface, the presentation of inputs and outputs to the user.
2. Presentation logic layer: Any processing that must be done to generate the presentation. Examples include editing input data and formatting output data.
3. Application logic layer: Includes all the logic and processing required to support the actual business applications and rules. Examples include credit checking, calculations, data analysis, and the like.
4. Data manipulation layer: Includes all the commands and logic required to store and retrieve data to and from database.
5. Data Layer: Is the actual data in a database

The HLIS system is a centralized system. The system uses a central, multi-user computer (mainframe) to host all the data, data manipulation and application logic layers of the information system. The users interact with the host computer via workstations.

The program running on the user's workstations was written in the Pascal language. This software handles presentation and presentation logic processing. Local controls

such as checking compulsory filling of fields is done by the workstation via scripted controls and referenced in a local system database. The workstation also handles a small portion of the data and data manipulation layers as it maintains a local system database that is used to configure the Pascal program running on the workstation. Entry data and program calls are rerouted as a record to the mainframe via a logical format interface program. The record consists of both data as well as the required transaction code to be executed in the mainframe system. Transaction codes are linked on the mainframe to actual program object files written in COBOL. The mainframe programs are responsible for the application logic layer that includes the business rules. The mainframe hosts a DB2 relational database and the data manipulation layer that allows the server to access and update the system's database as required. The system architecture is represented in figure 4-1.

**Figure 4.1 Legacy System Architecture**



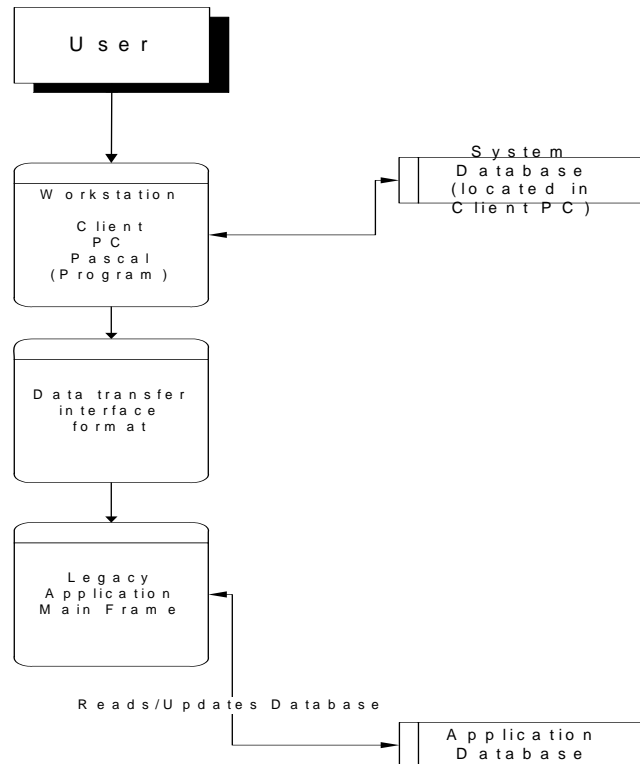
#### 4.2.1.1 Interface Architecture

The interface architecture defines the technologies used for input, output and inter-system connectivity (Whitten et al. 2000). The legacy system uses batch and on-line input/output processing.

In the on-line process, the user enters the input data in the workstation using a keyboard and the system processes the information in real-time. A good example of this process is the registration (creation) of an offer. Once the sales agent has completed the required offer entry screens and entered on the last screen, the workstation (Pascal) calls the applicable mainframe transaction to create an offer record in the offer dataset. A mainframe transaction generates offer numbers

automatically. Any exceptions found in the mainframe program generate an error response from the program back through the data transfer interface to the workstation. The error response is in the form of an error code, the message description of which is held in the definition dataset in the local database installed in the client PC and accessed by the workstation software. The error message is displayed at the bottom of the workstation screen. Figure 4-2 illustrates this process.

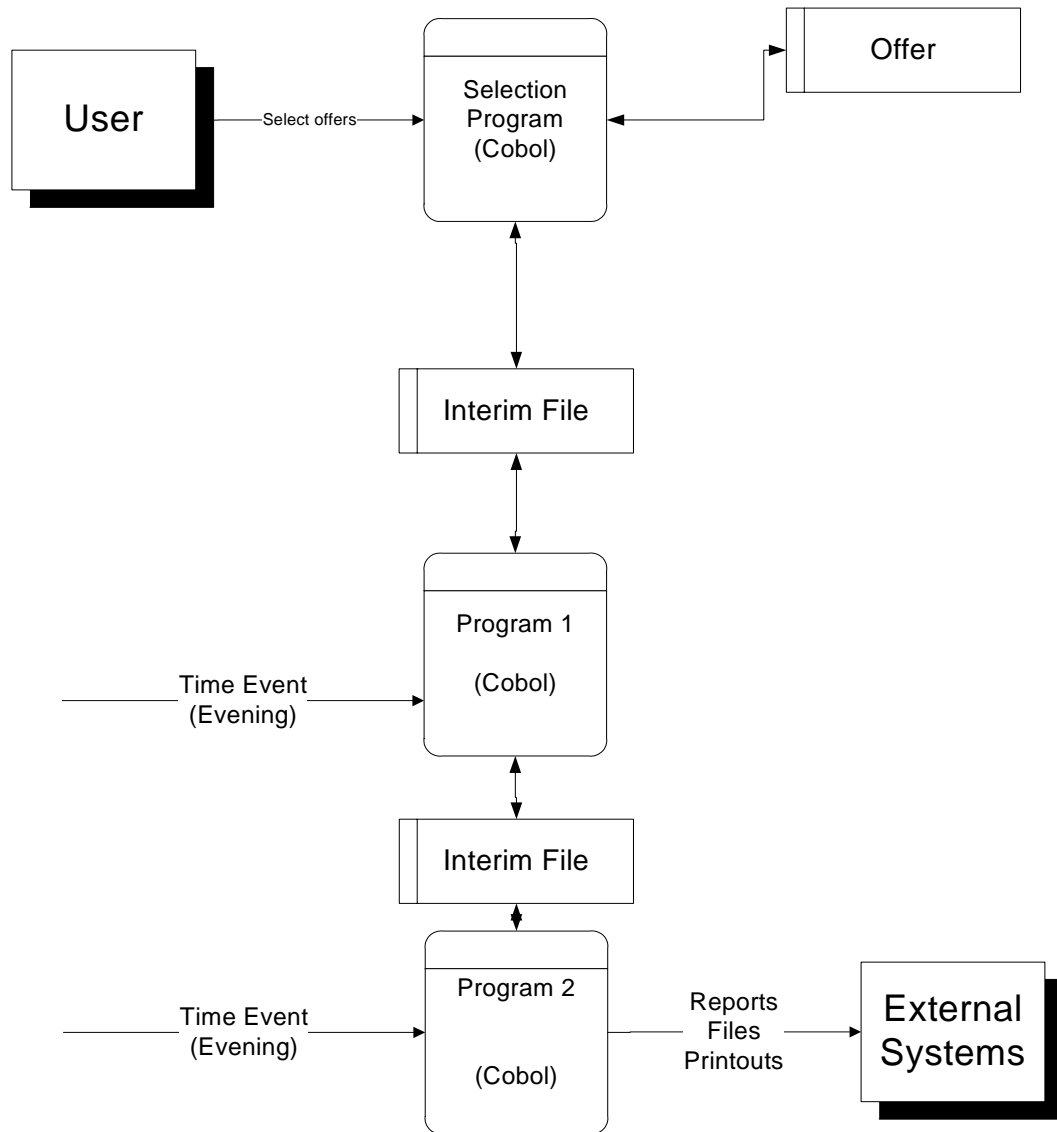
**Figure 4.2 On-line interface processing**



In batch-processing, transactions are accumulated into batches for periodic processing (Whitten et al. 2000). The loan generation process is a good example of a batch process and is illustrated in Figure 4-3. This process is scheduled as a batch job for the evening batch run at head office. The selection program selects all offers with a status of accepted by head office and writes offer details to an interim file to serve as input for program 1, described below.

Program 1 is responsible for the creation of the loan account and sub-accounts and connects people to loans. It changes the status of the offer to “offer definite” and writes relevant offer and loan data to an interim file to be used by program 2. Program 2 is responsible for generating print files for letters to clients and data files for interfaces for external systems.

**Figure 4.3 Batch processing in the legacy system**



*4.2.1.2 User Interface architecture*

The HLIS system uses a menu-driven interface. Figure 4-4 represents a sample screen layout for the current on-line system.

**Figure 4.4** Screen layout sample for procedural model

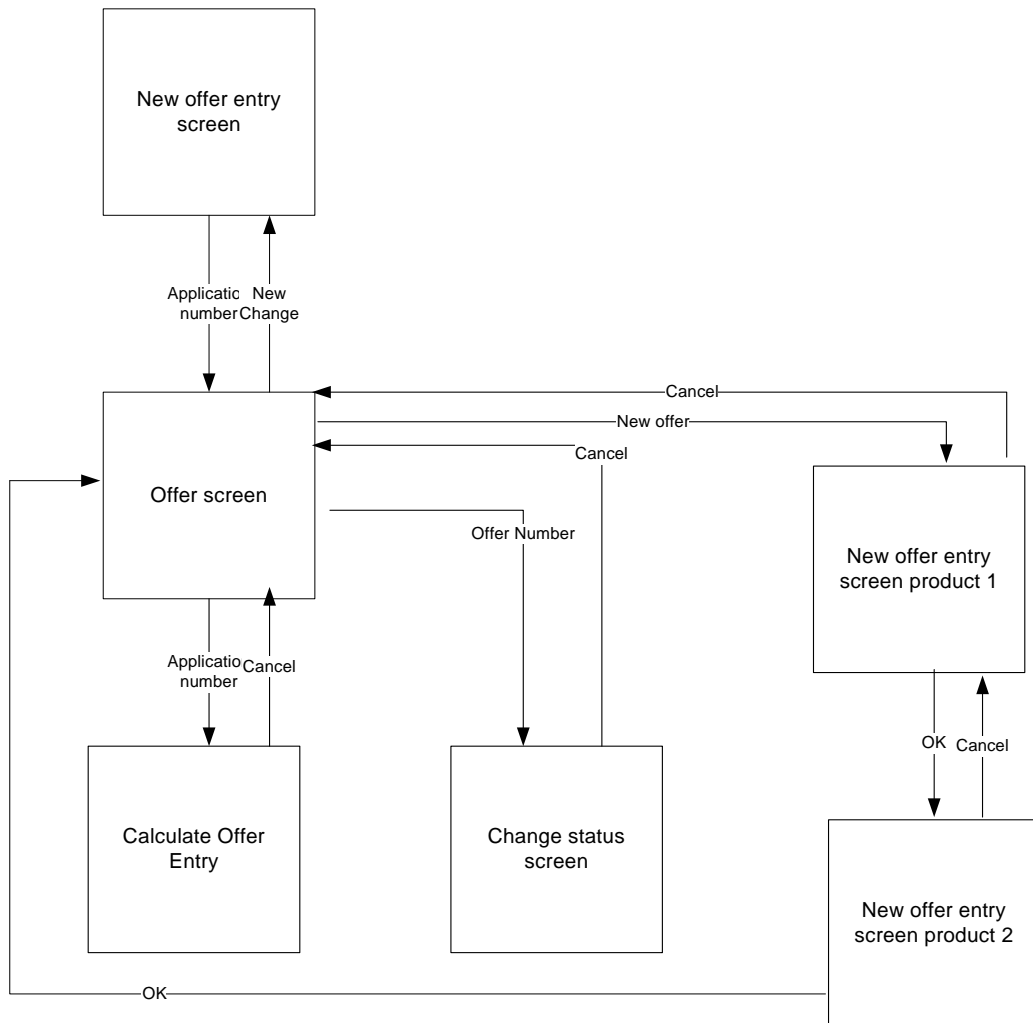


Navigation between options is done by use of arrow keys and enter key on selected option to advance to next logical screen. Return to previous logical screen is via the escape key. Navigation between data entry fields is done with the tab key. Mouse functionality is not enabled. All transaction confirmation messages and error messages resulting from controls are displayed in red in the bottom panel of the screen.

When filling in fields, a user may choose from a drop down list box by using the F1 key. An example is when the user chooses a product in the new offer entry screen. A product code can be typed in directly to the entry field or the user can press F1 to receive a list box of products and choose from there.

In order to document the user interface, a state transition diagram is generated. A state transition diagram is used to depict the sequence and variations of screens that can occur when the system user uses the terminal (Whitten et al. 2000). Also, the state transition diagram will be used to generate use cases to document the interaction with the system (Deursen et al. 2000). Figure 4-5 illustrates the state transition diagram of the legacy system. Appendix A has a complete set of screen shots for all the output screens of the system.

**Figure 4.5 Screen flow structure**



#### 4.2.2 Use cases

The following use cases have been documented to describe the Offer and Application Sub System:

- Process Application
- Process Offer Regional Office
- Process Offer Head Office
- Maintain Offer
- Loan Generation

The use cases were developed by using observation, interviews and documentation. First, the researcher visited the sales department for the head and regional offices of the case study information system's site during five different days and observed users operating the system as a complete observer during several hours. The justification for this role was to avoid intrusion in the normal operation of the information systems and learn by observing the users in the natural environment of the information systems. The notes generated from these observations were used to

document the normal flow of operation, actors, frequency of use and business rules for use cases

Two interviews were conducted with the developers in order understand the operation and identify the available documentation for the system. Interviews helped to validate and correct business processes documented during the observation phase. The developers identified several user manuals that were used to complete the use cases documents for this study.

The researcher reviewed user manuals to identify actors, normal flows, triggers, alternative flows, preconditions, postconditions and business rules for the use cases of this study. The results of the observation, interviews and manuals were documented in preliminary use cases. These preliminary uses were validated by reviewing them with the developers for errors, omissions, inconsistencies and problems.

The use cases that received the most attention in the research were the process offer by regional office case, the maintain offer case and the loan generation case. These three use cases were selected for focus as representative of the required criteria of investigating a procedural business flow, a more event-based flow and a batch processing flow respectively.

Table 4-1 defines information that pertains to this particular use case. Each piece of information is important in understanding the purpose behind the use case.

**Table 4.1 Offer by regional office Use case**

Use Case ID:	1
Use Case Name:	Process Offer by regional office
Actors:	Sales agent, Applicant, Administrator, Debtor and Insurer
Description:	This use-case satisfies all of the goals of setting up and processing a new offer to the status of production by regional office. This applies for both existing as well as new applicants. All aspects of the offer process are covered, from initial registration to the production of the offer at the regional office.
Trigger(s):	All events dealing with new and existing applicants applying for a home loan through the usual sales channels. Once the application is completed, the applicant requests an offer from the Sales Agent. An on-line offer form is loaded.
Preconditions:	The sales agent must be logged onto the system.
Postconditions:	Offer attains a status of produced at regional office.
Normal Flow:	Sales agent processes an offer and generates a yet to be authorized offer document all in a single session. 1. Relevant application data is made available in workstation offer entry screen once Sales agent enters application number in the offer entry screen. 2. Changes (if applicable) to relation and/or loan and/or property details are made. 3. A product or product combination is captured. All entry

	<p>requirements within the product or product combination structure are filled These include: the completion of all sub-product requirements and options such as linking savings, accounts and linking depot accounts, fixed interest rate duration and distribution of capital loan amount over product structure.</p> <ol style="list-style-type: none"> <li>4. Loan repayment options are captured including: external bank account number, method and frequency of payments amount of deposit to be paid.</li> <li>5. Sales agent attempts to upgrade offer status to Registered. Once all local and mainframe based controls have been processed, the offer is upgraded by the system to status registered.</li> <li>6. The computational module is called from the workstation by the sales agent in order to calculate: Loan - interest payments percentages and amount based on duration of fixed interest, reduction on capital amount over time, penalties for early loan settlement or non-scheduled payments against capital loan.</li> <li>7. On completion of the computational module, the offer is placed by the system as status ready for printing.</li> <li>8. The sales agent selects print from the offer entry screen. The offer document is printed.</li> <li>9. Once the applicant accepts the offer, the offer attains status of offer complete for transfer to head office.</li> </ol>
Alternative Flows:	<p>Sales agent processes an offer in a number of sessions with the applicant. A range of offers is made under a single application.</p> <p>1 – 8Completed over time and from last registered offer status A range of offers is made under a single application.</p> <p>1 – 8Repeated for every required offer variation. Each new offer receives an incremented series number.</p>
Exceptions:	
Includes:	
Priority:	1
Frequency of Use:	10/hour
Business Rules:	<p>A maximum of four sub-accounts are allowed within a contract.</p> <p>External account details (the payer account) must be linked to a bank registered with the home loan bank.</p> <p>An internal saving account must be linked to the sub-account representing the product.</p>
Special Requirements:	
Assumptions:	
Notes and Issues:	



Details of the remaining use cases are provided in Appendix B.

### **4.2.3 Legacy requirements model construction**

In order to construct a requirements model for the legacy system, the following elements generated as part of the reverse engineering activity are used:

- Description of the subsystem
- System architecture
- Interface architecture
- Use cases (business processes description)

These elements identify how the components of the legacy system relate to each other and are used to recover the requirements model in terms of data flow and entity relationship diagrams.

In order to generate the DFDs required to construct the legacy requirements model, business events to which the system must respond and appropriate responses were identified with the help of the use cases. Essentially, there are three types of events (Whitten et al. 2000) :

- External events: are so named because they are initiated by external agents. When these events happen, an input data flow occurs for the system in the DFD.
- Temporal events: trigger processes on the basis of time, or something that merely happens. When these events happen, an input called control flow occurs.
- State events: Trigger processes based on a system change from one state or condition to another.

Information systems usually respond to external or temporal events. State events are usually associated with real time systems (Whitten et al. 2000).

The possible events for Process Application use case are represented in Table 4-2. These events were identified from the use case description of Table 4-1. The descriptions of possible events for the remaining use cases for the Offer and Application Sub System are described in Appendix C.

**Table 4.2 Possible events for the Process Application use case**

<b>Actor</b>	<b>Event</b>	<b>Trigger</b>	<b>Response</b>
Applicant	Completes loan application	New application	A paper application is generated and put on the file cabinet
Sales Agent	Enters applicant and sales details for new application	New paper Application	Create new records for the application, applicant and other applicants relations tables in the database
Sales Agent	Enters property details for new application	Application and sales details are entered for the new application	Update the application table with the property details in the application table.
Sales Agent	Enter the loan requirement details	Property details are entered for the new application	Update the application table with the loan requirement details
Sales Agent	Income test request	Loan requirements are entered for the new application	Income test results (Fail or accepted)
Applicant	Application is rejected	Income test failed	Reject Application
Sales Agent	Selects new product	Income test is accepted	Update the application table with the product selection
Sales Agent	Invokes credit test module	Product is selected	Sends a credit request to the bank
Bank	Reviews credit request	Credit request from application	Application is rejected or allowed to continue
Sales Agent	Application document is generated	Credit test is accepted by the bank	The system generate an application document, updates the application table with the acceptance status and informs the sales agent that application has been accepted
Applicant	Signs document	Application is generated by the system	The signed application is stored in the customer file cabinet
Sales Agent	Request application acceptance status	Application is signed by the customer	System changes application status to accepted in the application table

Once these events were identified, data flow diagrams and context diagrams were drawn with the help of the list of transformations suggested by Whitten et al. (2000). The list of recommendations is:

- The actor that initiated the event will become the external agent
- The event will be handled by a process.
- The input or trigger will become the data or control flow
- All outputs and responses will become data flows

The diagrams generated out of the analysis of the events tables were:

- Context diagram
- DFDs

The notation used to describe the DFD and context diagram is expressed in Figure 4-6.

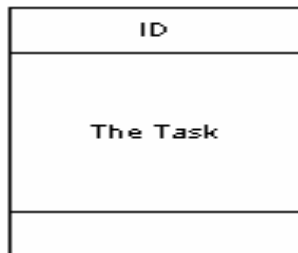
**Figure 4.6 DFD notation**



External agent



Data flow



Business Process



Data Store

The context diagram for the Office and Application sub system is shown in Figure 4-7. This context diagram was generated by identifying the actors, triggers and responses from Table 4-2 and Appendices C-1, C-2, C-3 and C-4. Each identified actor is represented as an external agent, each response as a data flow going out the sub-system and each trigger as a data flow going in the sub system in the context diagram. The DFD at the first level was generated by identifying the responses, triggers and actors for each use case in the Office and Application sub-system identified in Table 4-2 and Appendices C-1, C-2, C-3 and C-4. Each use case of the Office and Application sub-system is represented as a process, each trigger as a data flow going in the processes, each actor as an external agent and each response a data flow going out the processes in the DFD in Figure 4-8.

DFDs for each of the use cases of the Office and Application sub-system are presented in Appendix D of this dissertation and were generated based on the identification of actors, triggers, events, and responses from Table 4-2 and Appendices C-1, C-2, C-3 and C-4. Each DFD at the second level represents actors as external agents, triggers as data flows going into the processes, responses as data flows going out the processes and events as the processes that handle them.

**Figure 4.7 Context diagram for the Offer and Application sub-system**

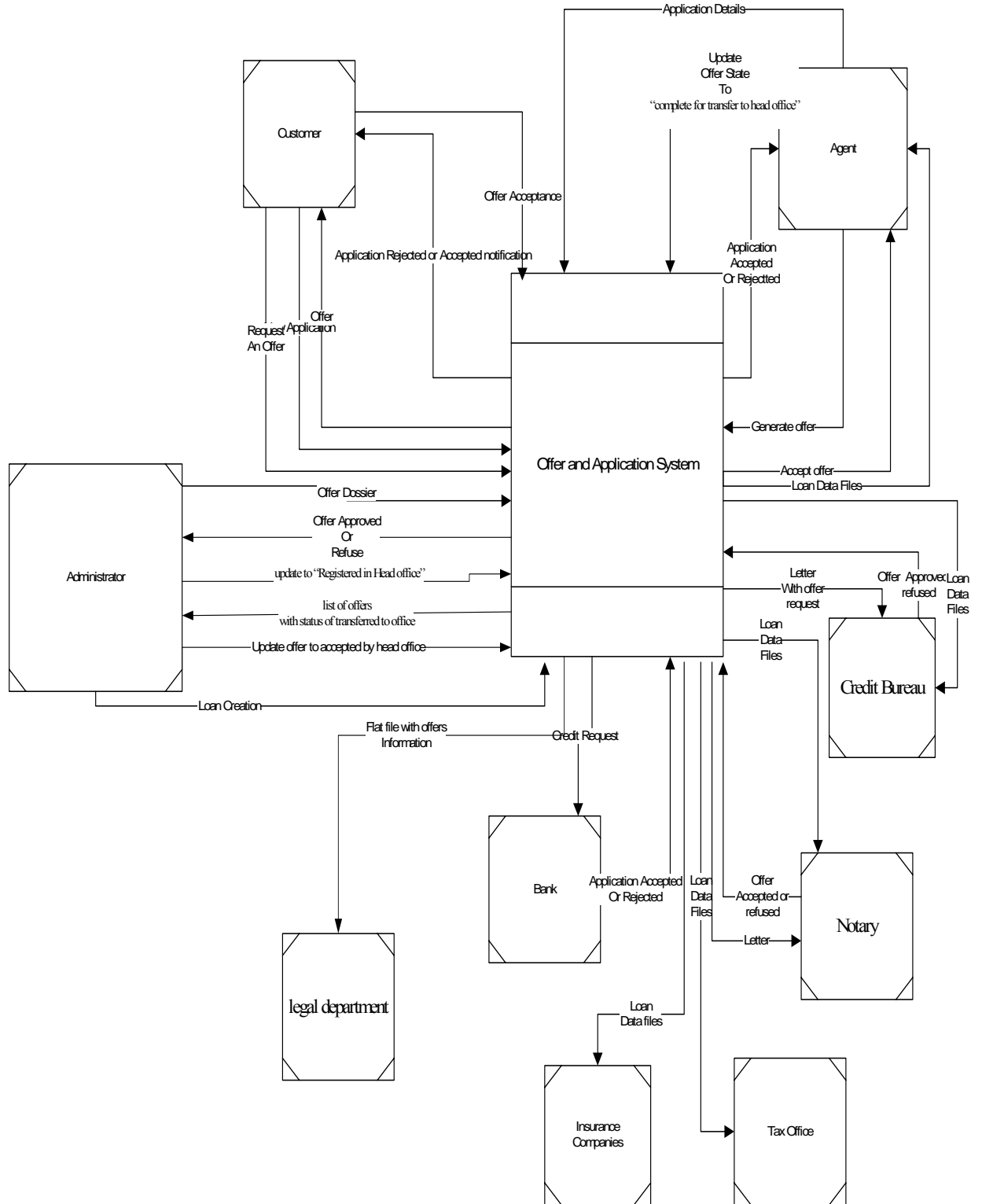
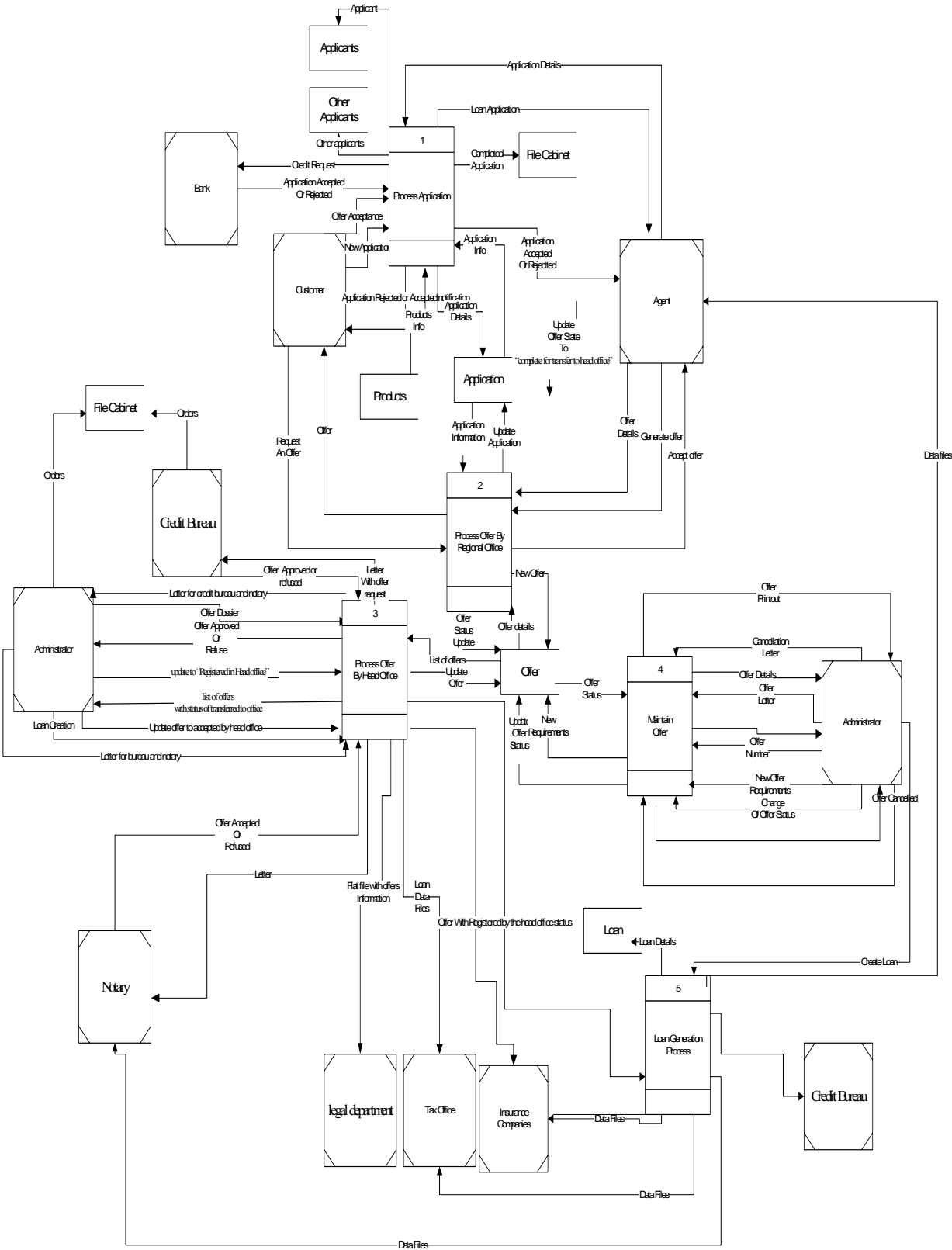
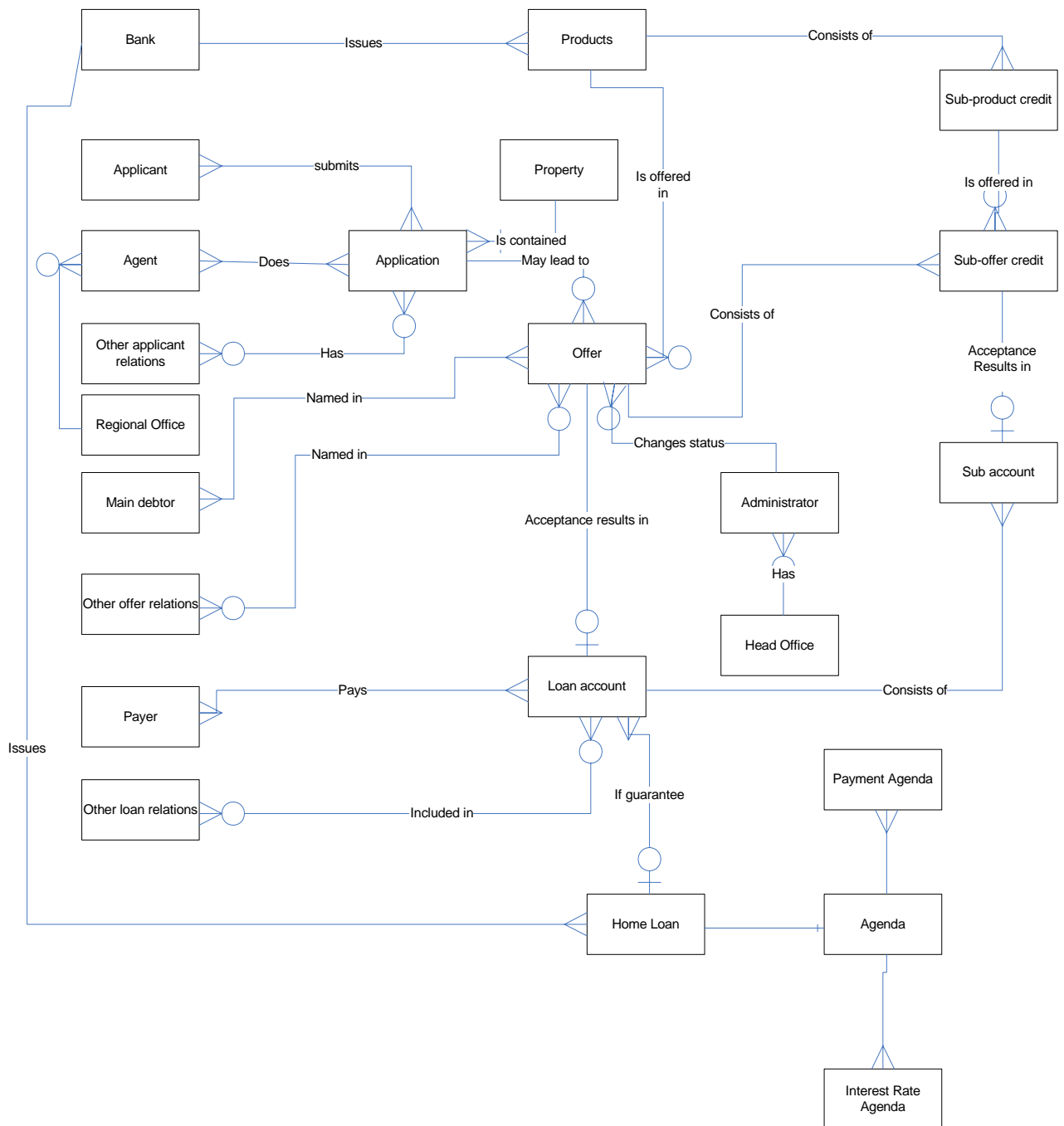


Figure 4.8 Offer Figure 4-8 Offer and Application sub-system DFD



The data model found in the system architecture documentation was examined and the researcher found that it was incomplete because many of the entities that were presented in the database script source code collected from the system were not included in the model. This was expected as the system has been modified many times since the date of the documentation. The model was completed in order to reflect its current state and it is presented in Figure 4-9.

**Figure 4.9 Completed E-R data model**



### 4.3 Re-engineering of legacy systems

As part of the Jacobson and Lindstrom (1991) methodology, the component-based requirements model was developed with the help of the use cases prepared for the recovery of the legacy requirements models. These use cases contain the information assimilated from the source code, manual, system architecture documentation, open interviews and research observations described as description elements in the Jacobson and Lindstrom (1991) methodology. These description elements were used to build the component-based model. The following UML diagrams were generated as suggested by Houston and Norris (2001) as the required diagrams to model component requirements models.:

- Use case diagrams
- Class diagrams
- Sequence diagram
- State diagrams
- Activity diagrams.

The UML diagrams followed the UML 2.0 standard (OMG 2003). In keeping with the UML approach to model development, it is essential to define actors for the system. Reed (2002) defines an actor as a stimulator of the system and an initiator of an event. Actors can also be passive recipients of stimuli from the system. Actors are mostly thought of as human beings but can also be other interfaced systems that receive input from the system or provide input into the system or hardware devices (Reed 2002). The actors identified with the help of the use cases for the sub-system under research are presented in Table 4-3.

**Table 4.3 Sub-system Actors**

Actor	Definition
System	Creates loan structure
Applicant	Applies for a home loan product
Sales Agent	Facilitates sale of product
Home Loan Administrator (regional)	Handles Offer and loan admin. At regional level
Home Loan Administrator (head office)	Handles Offer and loan admin. At head office level
Debtor	Responsible for loan debt
Bank	Loan provider and administrator
Notary	Provider of deed of sale, bond documentation
Business	Updates product model
Insured	Covered by policy
Insurer	Pays policy premium
Insurance Company	Underwrites policy
Administration management	Requests and receives reports
Tax department	Requests loan and policy details
Credit Bureau	Issues credit assessment of debtor
Loan system	External system handling all loan administration



Actor	Definition
Insurance system	External system handling insurance administration

Use case models capture the functional requirements of an information system by focusing on usage situations, the tasks that users want to accomplish with an information system. In this context, a use case model is a conceptual model that articulates the required behavior of a system in non-technical, implementation-independent terms. The use case model shows how the actor interfaces to the information systems. The use case model of the case study's sub-system is shown in Figure 4-10. The interaction of actors with the use cases identified for the system is represented in this model.

There are functions that are shared or can be reused by other use cases. These are represented by the include UML construct. This construct represents a relationship between two use cases that shows that an instance of one use case will also contain the behavior specified by another use case (the included use case) (OMG 2003).

In order to generate the class diagram, it was important to identify the objects of the system. This step was accomplished by reviewing each use case to find nouns that correspond to business entities or events (Jacobson 1987). The result of this search is documented in Table 4-4 depicting all the classes and types identified in the use cases.



**Table 4.4 Possible Classes and interfaces for the Sub-System**

Class	Type
ProcessApplicationPanel	Boundary
CreditApplicationPanel	Boundary
ProcessOfferRegionalPanel	Boundary
ProcessOfferHOfficePanel	Boundary
CreditCheckInterface	Boundary
MaintainOfferPanel	Boundary
InquireOfferPanel	Boundary
LoanGenerationPanel	Boundary
Credit Test	Control
Process Application	Control
Process Offer regional	Control
Process Offer head office	Control
Maintain Offer	Control
Loan Generation	Control
Architecture Infrastructure	Control
Print	Control
File	Control
Application	Entity
Relation	Entity
Applicant	Entity
Agent	Entity
Loan	Entity
Bank	Entity
Debtor	Entity
Property	Entity
Product	Entity
Sub-product	Entity
Regional office	Entity
Head office	Entity
Administrator	Entity
Offer	Entity
Cabinet	Entity
Account	Entity
Sub Account	Entity
Service Centre	Entity
Agenda	Entity
Payment Agenda	Entity
Interest Rate Agenda	Entity
Income Test	Control
Computation Module	Control

The Application and Offer sub-system is represented as a package with the stereotype of <<system>>, as seen in Figure 4-11. The system was broken into multiple subsystems as seen in the same picture. Subsystems, like systems, are

stereotyped packages with the stereotype of <<subsystem>> and are a grouping of model elements that are part of the overall system.

A subsystem can be modeled as a UML package and can communicate with other packages by using interfaces. Figure 4-12 shows the different UML packages of the case study and their interfaces and possible interactions. UML packages manage object model elements, such as classes (Yun-Tung 2001). The process sub-system UML package that encapsulates the control type classes is represented in Figure 4-13. The UML package can be interfaced by using the IProcess interface that is indicated in the same diagram. The class diagrams for the Data and Interface subsystems are included as appendices E-1 and E-2.

Detailed object interactions can be modeled with the sequence diagram. The purpose of the sequence diagram is to represent the interaction between object instances within the system. They provide the sequence of messages passing between objects over time (Sparx Systems 2001). A sequence diagram can represent each use case. Figure 4-14 is the sequence diagram for the Process Offer Head Office use case. The diagram begins with objects organized into columns to differentiate the sequencing stages. The rectangular blocks located on the dashed vertical lines are focus-of-control rectangles indicating that the object above is in control of that messaging sequence. For example, the instantiated offer class (object) has control over many messaging sequences in the Process Offer Head Office use case. The sequence diagrams for the rest of the use cases of the case study are in appendices E-3, E-4, E-5 and E-6.

Activity diagrams can be divided into object swimlanes that determine which object is responsible for which activity. For example, the activity diagram for the Loan Generation use case is depicted in Figure 4-15. In this figure, it is possible to identify two different swimlanes that are handled by the administrator and system objects. In the diagram, a single transition comes out of each activity, connecting it to the next activity. A transition may fork into two or more parallel activities as can be seen after the creation of the loan structure in the example of figure 4-15 where three activities are performed in parallel. The activity diagram of the Maintain Offer use case is included in Appendix E-7 and the Process Offer by Head Office use case is in Appendix E-8.

A state diagram models the life cycle of a single object. It depicts the different states an object can have, the events that cause the object to change state over time, and the rules that govern the object's transition between states (Whitten et al. 2000). Figure 4-16 depicts the state diagram of the test income object identified in the class diagram. Appendix E-9 shows the state diagram for the offer object with its different states and transitions. Activity diagrams and state diagrams are related. While a state diagram focuses attention on an object undergoing a process (or on a process as an object), an activity diagram focuses on the flow of activities involved in a single process. The activity diagram shows how those activities depend on one another.

Figure 4.11 Subsystems relationships

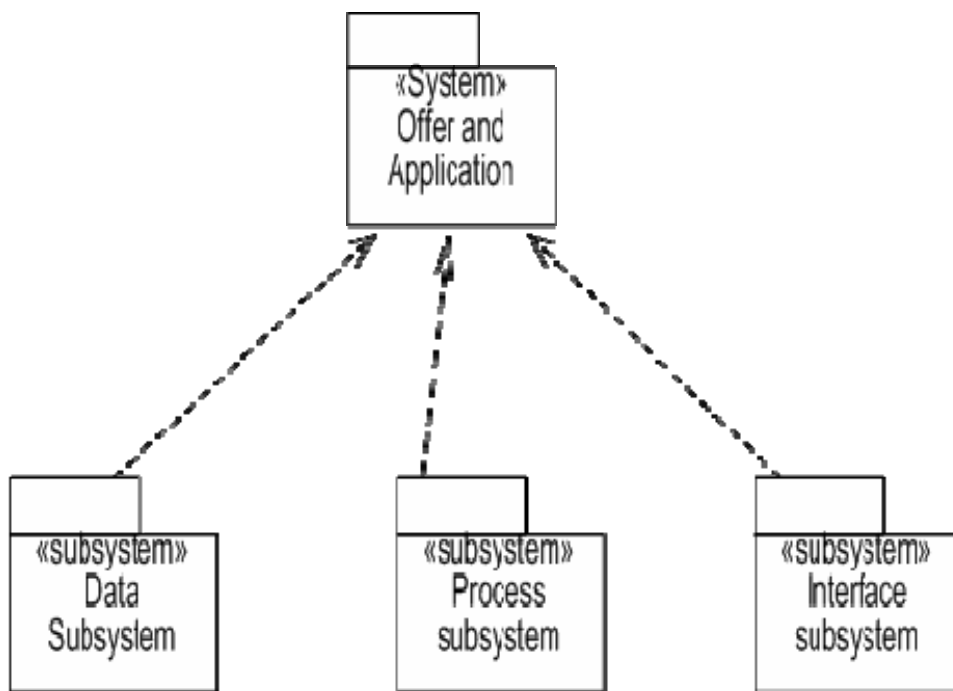


Figure 4.12 Interface relationships

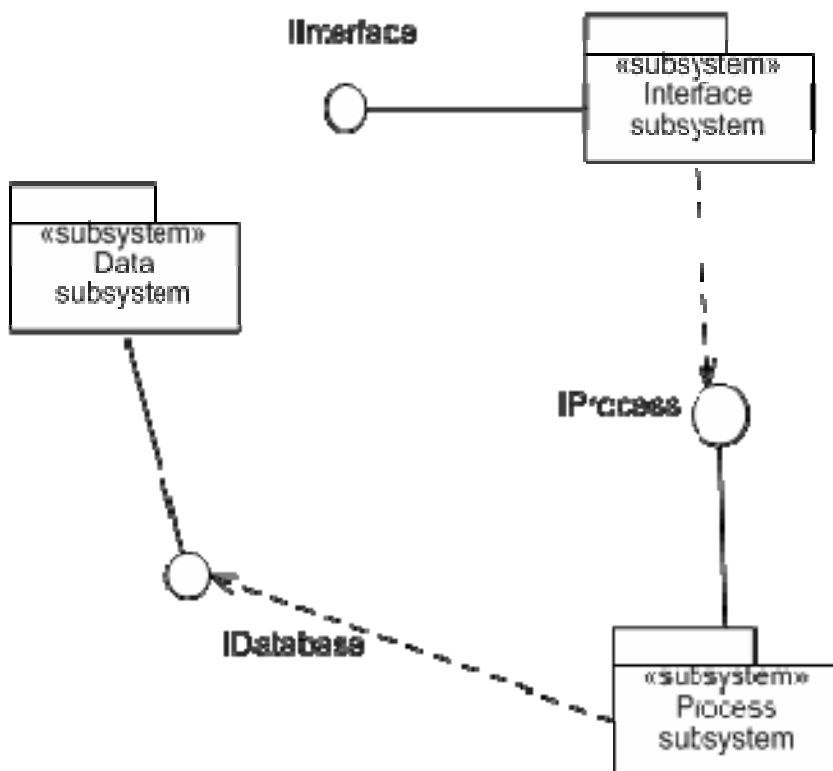
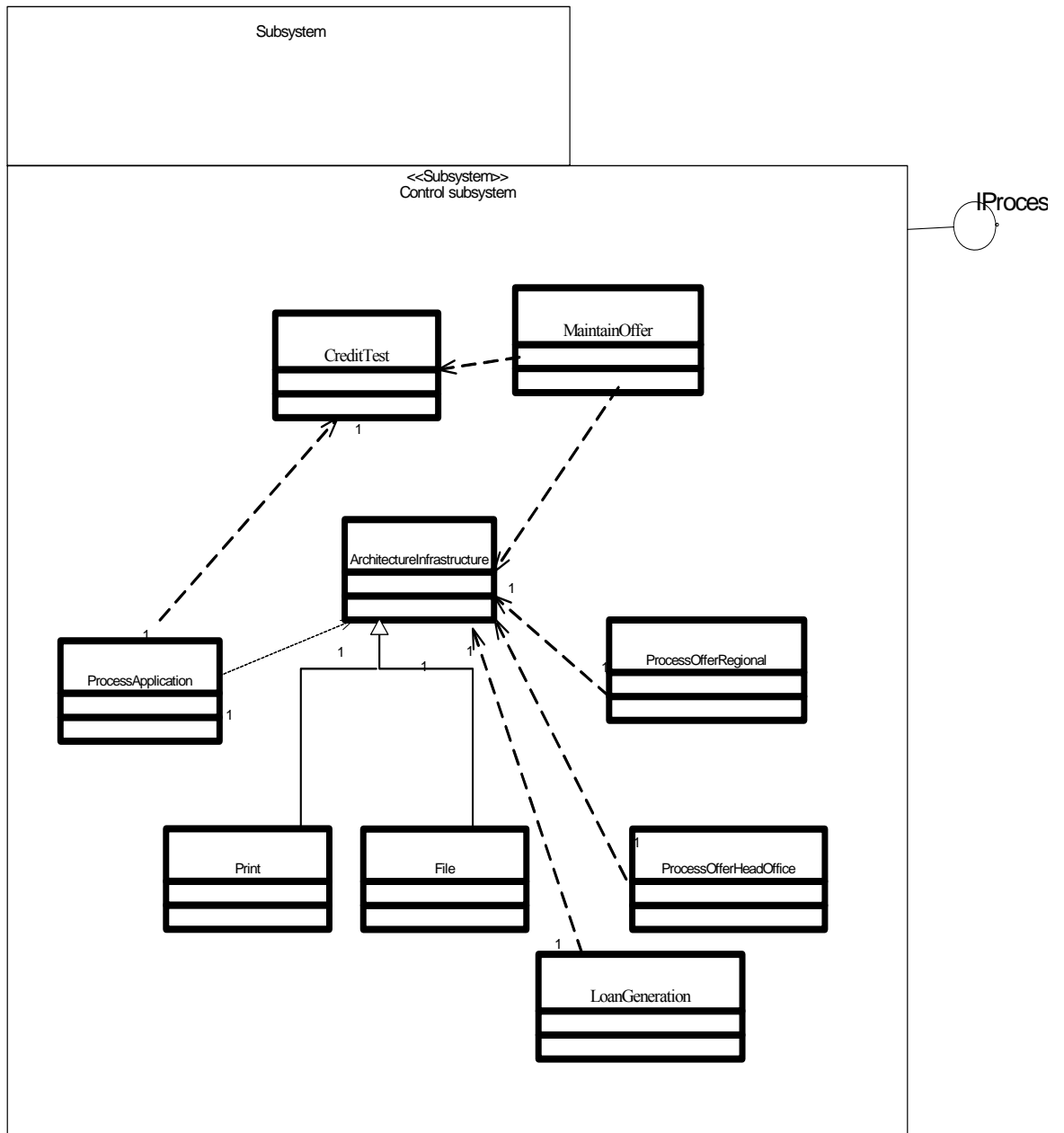


Figure 4.13 Control Package



**Figure 4.14 Process Offer Head Office Sequence Diagram**

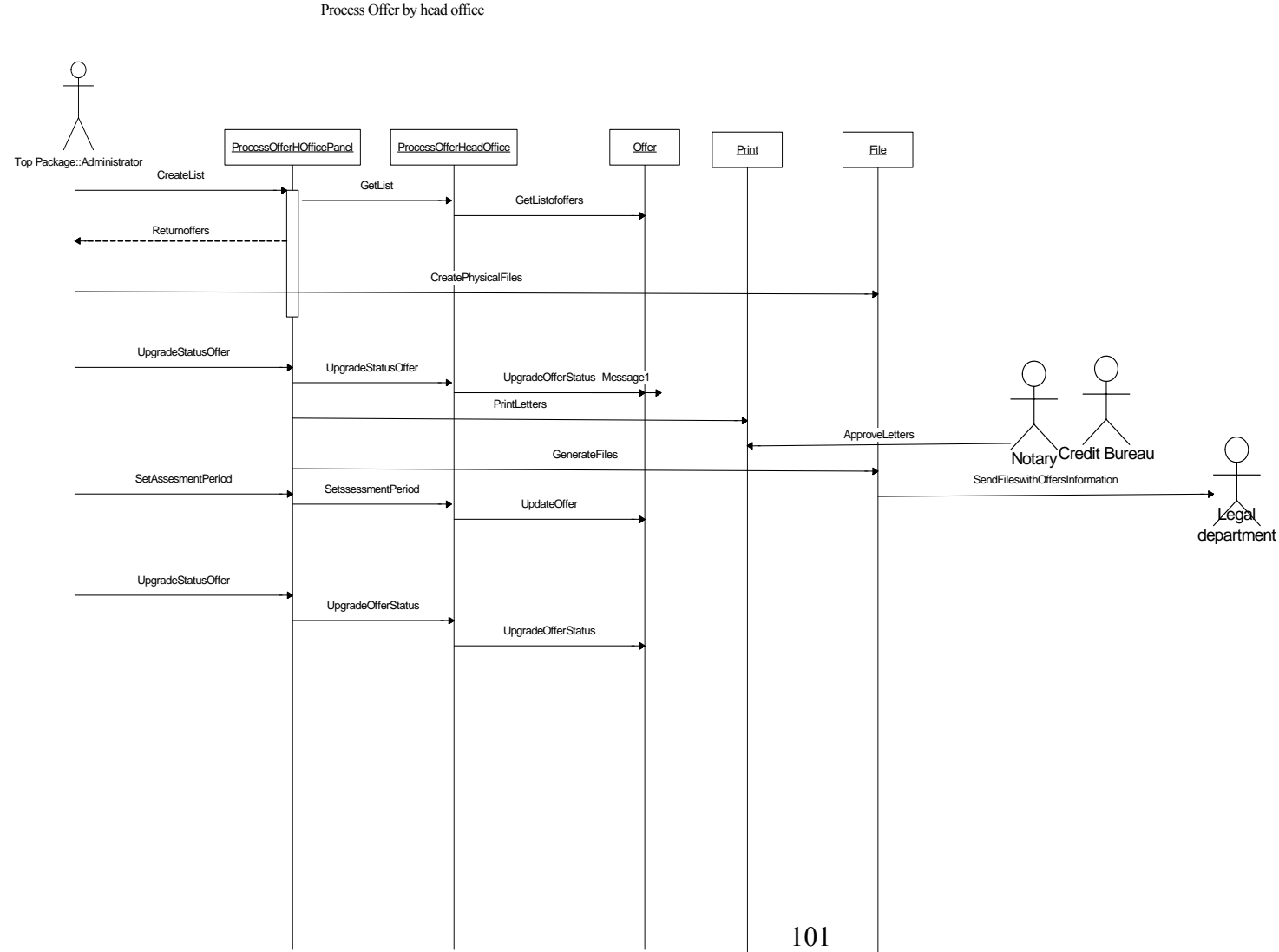


Figure 4.15 Loan generation activity diagram

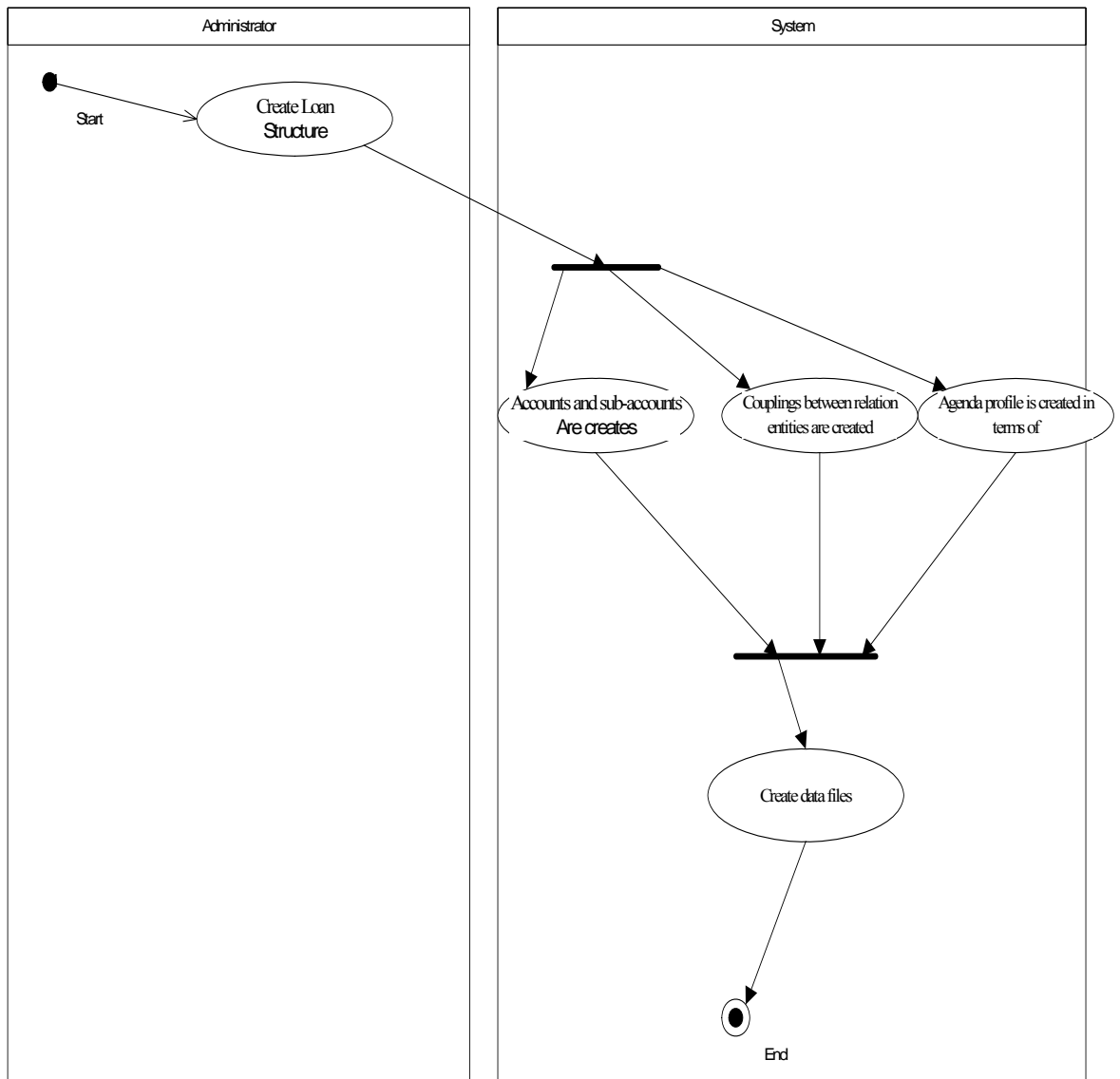
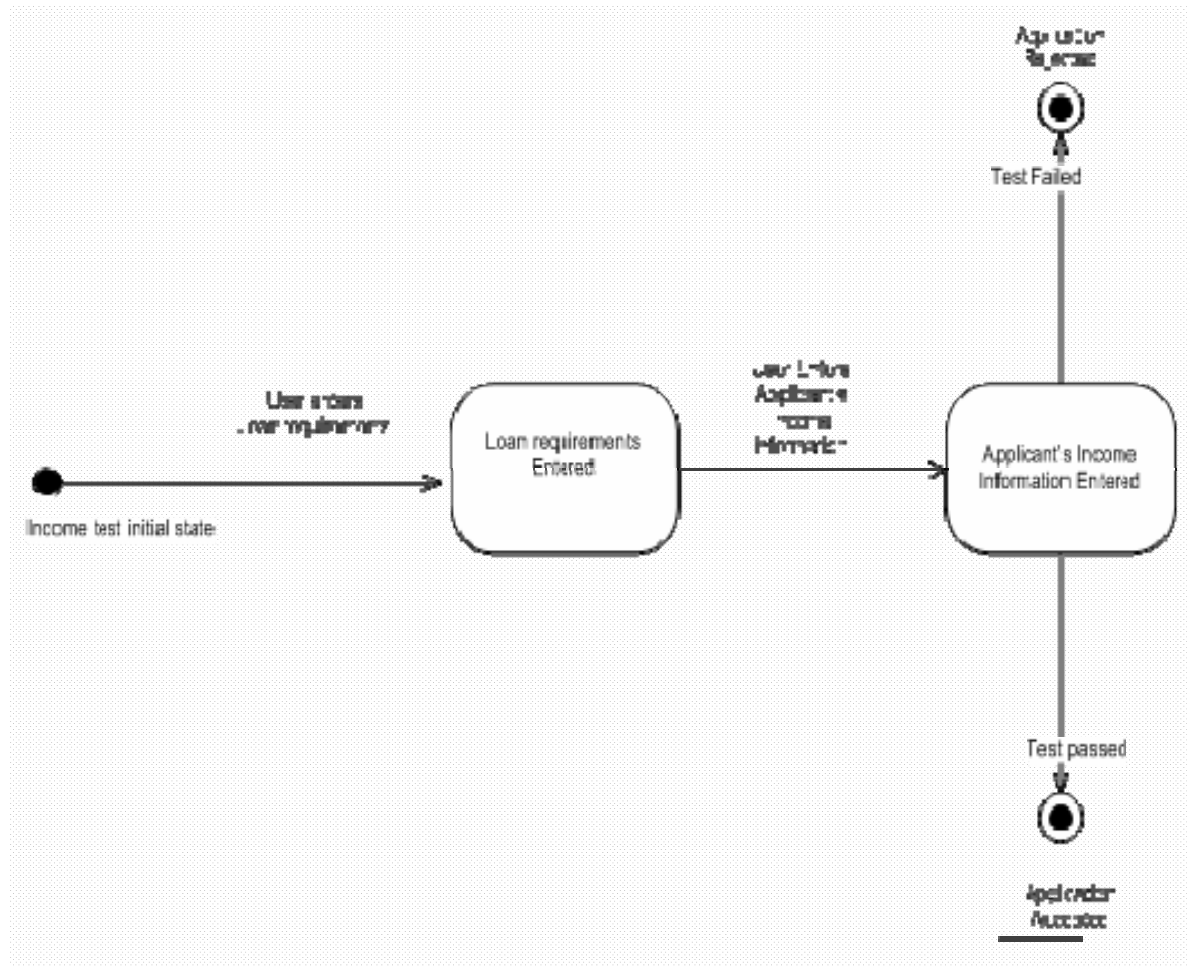




Figure 4.16 State diagram of the Income test object



After the UML diagrams that represent the component analysis model were completed, a mapping of each analysis object to the implementation of the old system was conducted as suggested by Jacobson and Lindstrom (1991).

The primitive description elements used for this research were architecture diagram, database schema, training manuals, interviews with developers, database scripts and system observation.

A mapping was performed in order to verify that all the objects for each of the diagrams that represent the component analysis model were motivated by at least one primitive description element. A summary of this mapping is included in table 4-5. The map shows that all analysis objects and dependencies for each component-based diagram were motivated by at least one primitive description element.

**Table 4.5 Mapping of Component Analysis models into Description Elements**

<b>Diagram</b>	<b>Description Elements</b>
Use case	Architecture diagram, Interviews, Training manuals, System observation
Class	Interviews, Training Manuals, Database scripts, System observation
Sequence	Interviews, Training Manuals, Database scripts, System observation
State	Interviews, Training Manuals, System observation
Activity	Interviews, Training Manuals, System observation

#### **4.4 Ontological Evaluation**

Evaluation is an important step in the design science research framework for this study. The research activity for this step is the ontological evaluation of requirements models by using the Fettke and Loos (2003a) methodology. This activity consists of four steps (Fettke & Loos 2003a):

- Developing a transformation mapping,
- Identifying ontological modeling deficiencies,
- Transforming the reference model, and
- Assessing the results.

The purpose of the first and second step is to evaluate the constructs used to build the legacy and re-engineered component models. This evaluation was performed in order to identify ontological deficiencies in these constructs when it comes to representation of functionality in re-engineered component requirements models that were represented by the traditional requirements model used to build the legacy information system.

The third and fourth step have the objective of evaluating the requirements models built as part of the reverse engineering and re-engineering research activities that are part of the research framework. In the third step, the requirements models will be transformed into an ontological model. The outcome of this step is two ontologically normalized reference models. In order to generate these normalized reference models in BWW terms, the Rosemann and Green (2002) BWW meta-models are used.

In the fourth step, the two models are compared based on their ontologically normalized representations. The result of a comparison will be that the models are ontological equivalent, complementary or in conflict. The summary of the findings is presented in the discussion Chapter 5.

#### 4.4.1 Developing a transformation mapping

In this section of the study, traditional and component diagrams are mapped onto BWW constructs. The mapping of traditional diagrams to BWW constructs is depicted in Table 4-6 and was the result of a research study conducted by Valverde and Toleman (2006). The traditional diagrams used are the three types proposed by the Yourdon (1989) structured analysis:

- Context diagram
- Data flow diagram
- Entity relationship diagram.

There are two types DFD models: 1) Physical DFDs – where the diagram describes the physical components of the information system and 2) Logical DFDs – that describe the meaning or the ‘what’ of the components of the information systems (Wand & Weber 1989).

As this research deals only with the requirements models, only logical DFDs will be analyzed. The logical DFD to BWW construct mapping is based on the work of Wand and Weber (1989). In logical DFDs, data flows represent external and internal events. Properties of real things may be represented by data elements described in data dictionaries but not in data flows and data stores.

There is no explicit representation of the states of the real system in a DFD. Rather, the possible and allowed states of the information systems are defined implicitly in terms of possible and allowed values of the data elements described in the data dictionary and therefore not represented in the DFD (Wand & Weber 1989).

Events of the information system are represented by data flows. External events are represented by data flows coming from a source while internal events are represented by internal data flows that are generated when the system responds to an external event (Wand & Weber 1989).

A DFD fully represents a system if and only if there is a path between every pair of processes. If this is not the case, then the DFD represents two or more disconnected information systems. External agents and data stores are represented by things and they form part of the environment in the BWW model (Wand & Weber 1989).

The things in the system are processes and data according to Wand and Weber (1989), given this interpretation processes can be represented by things. Data linked to a process, a process linked to another process and a process linked to an external agent may be interpreted as coupling (Wand & Weber 1989). External agents are interpreted as things, therefore a link between them and processes may be interpreted as coupling as they link things in a system (Wand & Weber 1989).

In DFDs, decomposition involves breaking a process into a number of sub-processes. DFDs conform to the BWW model notion of a good decomposition (Wand & Weber 1989). In their analysis, Wand and Weber (1989) did not include the transformation construct mapping as this was added in the BWW model after the publication of their analysis. In this dissertation, transformations were interpreted as

processes because they represent a procedure by which data inputs are transformed into data outputs (Satzinger et al. 2002 p. 196).

Entity relationship diagrams (ERD) contain entities and relationships. Although Wand and Weber's (1989) interpretation that both can be viewed as representing things of a real system, the interpretation of Green and Rosemann (2000) of the entity representing a class was used as entities can represent multiple instances of things. Properties are represented directly in the entity relationship diagram via the notion of attributes. Coupling between things can be represented by the lines between the entities and relationships (Wand & Weber 1989). The interpretation of the functional decomposition diagram mapping was taken from Green and Rosemann's (2000) work.

**Table 4.6 Mapping between traditional and BWW constructs**

BWW construct	Context Diagram	DFD	ERD
Thing	External agents External data stores System	External Agents External Data Store Data Stores Process	
Property: In particular IN PARTICULAR In general Intrinsic Mutual Emergent Hereditary Attributes			Attribute type
Class			Entity type
Kind			Specialization/ generalization (IS- A)
Conceivable state space			
State law			Specialization/ generalization descriptors; [Min., max.] cardinalities
Lawful state space			
Event		Data flow	
Process		DFD	
Conceivable event space			
Transformation		Process	
Lawful transformation			

BWW construct	Context Diagram	DFD	ERD
Lawful event space			
History			
Acts on			
Coupling: Binding mutual property	Ext. Agent->Data Flow-> System  System->Data Flow-> External Data store	Process->Data Flow->Ext. Agents  Ext. Agent->Data Flow-> Process  Process->Data Flow-> Data store  Data stores ->Data Flow-> Process	Relationship type (no symbol for relationship in grammar)
System	System	DFD	
System Composition		External agents and data stores in a DFD	
System Environment	External Agent External data stores	External Agent External Data Stores	
System structure		DFD	
Sub-system		DFD	
System decomposition		DFDs and sub diagrams	
Level structure		Series of processes decomposed at different levels	
External event		Data flow	
Stable state			
Unstable state			
Internal event		Data flow	
Well-defined event			
Poorly defined event			

(Source: Valverde and Toleman 2006 p. 65)

Component-based models were generated using UML diagrams and mapped onto BWW constructs as shown in Table 4-7.

Irwin and Turk (2005) propose that the UML-actor should be defined as a type of classifier in the UML meta-model, which corresponds to a BWW kind, and as a role or facet of a thing in the UML specification, which corresponds to a BWW property. The interpretation of Opdahl and Henderson-Sellers (2002b) of the UML-actor

being a BWW thing was used here as it matches the definition of Wand and Weber (1995) of things acting on the proposed system.

Irwin and Turk (2005) and Opdahl and Henderson-Sellers (2002b) agree that the use case UML-association corresponds to a binding mutual property of an external entity BWW construct. They also agree that the same applies to the UML-use case that is interpreted in both studies as a BWW process.

The UML-extend and UML-include were mapped as a BWW-binding mutual property (Opdahl and Henderson-Sellers 2002b). Irwin and Turk (2005) argue that UML actors also can be considered a BWW system environment as they are external entities that interact with the system. Although Opdahl and Henderson-Sellers (2002b) argue that the BWW system environment construct is not defined in the UML grammar, the interpretation by Irwin and Turk (2005) was used in Table 4-7 because the argument that actors that are outside the system boundary and can be considered ‘environment’ is valid as this satisfies the Wand and Weber (1995) definition of BWW-system environment. However, the UML grammar does not support the “internal actors” (Irwin & Turk 2005) therefore all the actors are external and considered part of the system environment.

Irwin and Turk (2005) propose that the use case construct represents a BWW thing and a process at the same time and is therefore ontologically overloaded. The argument that the use case is also a thing was accepted and used in Table 4-7 for the following reasons: 1) the use case can be defined as a classifier; and 2) this is an element that has behavioral and structural features and that may participate in relationships (Irwin & Turk 2005). It also satisfies the definition of Wand and Weber (1995) of a BWW mutual property construct as two use cases can be linked with a UML-extend or UML-include construct that was previously interpreted as a mutual property and requires the link between two or more things.

The UML-system is consistent with the BWW definition, and thus there is technically no ontological discrepancy with BWW system construct (Irwin & Turk 2005).

**Table 4.7 Mapping between UML diagrams and BWW constructs**

<b>BWW construct</b>	<b>Use Case</b>	<b>Sequence</b>	<b>Class</b>	<b>State</b>	<b>Activity</b>
Thing	Actor Use case	Object		Object	Object Swimlane
Property: In particular IN PARTICULAR In general Intrinsic Mutual Emergent Hereditary Attributes			UML attribute		Activity
Class			Class		

BWW construct	Use Case	Sequence	Class	State	Activity
Kind			Generalization UML aggregate class UML composite class		
State				State	State of object
Conceivable state space				State machine	
State law			UML- multiplicit y	State>Tra nsition>St ate	
Lawful state space				Sub states	
Event				Trigger	Activity
Process	Use Case				Activity diagram Activity
Conceivable event space				All triggers	
Transformation			UML operation		Activity
Lawful transformation					Guard conditions On transitions
Lawful event space					
History				Shallow history state construct	
Acts on					
Coupling: Binding mutual property	UML associatio n UML extend UML include	Messages	UML associatio n UML interface		.
System	System Boundary	Sequence Diagram	Package with <<system >>		
System					

BWW construct	Use Case	Sequence	Class	State	Activity
Composition					
System environment	Actor	<<Stereotype>>			
System structure		Messages			
Sub-system			Package with <<subsystem>>		
System decomposition					
Level structure					
External event				<<Stereotype>	
Stable state				Final State	
Unstable state				Initial State	
Internal event				<<Stereotype>>	
Well-defined event				Trigger	
Poorly defined event					

(Source: Dussart et al. 2004 p.85)

The interpretation for the mapping of BWW constructs for the activity, state, class and sequence diagrams comes for the most part from the work of Dussart et al. (2004) although some changes were made based on the interpretation of Opdahl and Henderson-Sellers (2002b) and the BWW construct definitions of Wand and Weber (1995). The BWW ontological construct “thing” can be associated with the object in the sequence, activity and state diagrams. The activity chart can show the transformations made on objects during activities and therefore interpreted as BWW transformation and property constructs (Dussart et al. 2004).

In the sequence diagram, the interpretation of Dussart et al. (2004) is consistent with the interpretation of Opdahl and Henderson-Sellers (2002b) for the UML-object and UML-message being mapped as BWW thing and binding mutual property constructs but it conflicts with the interpretation of the UML-message being interpreted as a BWW system structure construct because this is not mapped to any UML construct in the Opdahl and Henderson-Sellers (2002b) interpretation. However, the interpretation by Dussart et al. (2004) was used as this satisfies better the Wand and Weber (1995) definition of a BWW system structure construct of a set of couplings that exist among things in the system and things in the environment of the system as UML-messages link objects (things) and actors (Environment) together.



The interpretation of a UML-object being considered a BWW system composition of Dussart et al. (2004) conflicts also with the interpretation by Opdahl and Henderson-Sellers (2002b) of a BWW system composition construct being mapped as a UML-physical system. The interpretation by Dussart et al. (2004) was not used as the system composition BWW construct is a set of things in the system (Wand and Weber 1995) while an object is only considered one thing and not a set of things.

A system can be represented using the sequence diagrams, the system environment, that is to say external and internal things to the system cannot be differentiated without a stereotype (Dussart et al. 2004).

The state diagram interpretation by Dussart et al. (2004) did not conflict with the interpretation by Opdahl and Henderson-Sellers (2002b). States of the thing are represented by the state of the object in the activity diagram or by the state construct in the state diagram. A state machine in the state diagram represents the conceivable State Space, defined as all the states that a thing may ever assume. A Lawful State Space can be represented in a state diagram using substates. Stable States and Unstable States can respectively be represented by the final state or the initial state in a state diagram (Dussart et al. 2004).

The activity diagram interpretation by Dussart et al. (2004) was used for the most part except the interpretation of the UML-swimlane that came from Opdahl and Henderson-Sellers (2002b). Dussart et al. argue that the UML-swimlane can represent either a thing (such as an organization) or a hereditary property of the thing (a user of the organization). However, a user in an organization is in fact an actor, defined earlier as a thing and not a property.

Events are represented as the trigger for a transition in the state diagram. But events can also be represented as an activity in the activity diagram. There is no grammatical differentiation for external and internal events but the use of the use cases for human-machine interaction diagrams or the use of stereotypes could help make the differentiation possible. The Conceivable Event Space can be observed on the state machine of a thing by looking at all transitions' triggers. There exists no construct for a poorly defined event, and well-defined events use the same grammatical construct as a normal event (Dussart et al. 2004).

Lawful transformations are represented by guard conditions on transitions. There is no grammatical construct for Lawful event space. History can be modeled using the shallow history state construct in the state diagram. The BWW construct Acts on cannot be represented in the same way as it is defined in the definitions of the ontological constructs but could eventually be associated to the composition relationship in the class diagram, for example, in a composition relation between a thing "Activity" and a thing "Project" (Dussart et al. 2004).

Class diagrams can contain symbols for classes, associations, attributes, operations, and generalizations. Class and kind are respectively represented in the UML in the class diagram with the class and the generalization constructs (Dussart et al. 2004). UML operations can be depicted by BWW transformations and UML attributes as BWW characteristic intrinsic properties (Opdahl & Henderson-Sellers 2004).

Class diagrams can also show the subsystem architecture, where the primary elements are UML system and subsystems. Subsystems represent components during development (Opdahl and Henderson-Sellers 2004). Subsystems and systems can be represented using a stereotyped package (Dussart et al. 2004).

Relations between classes are depicted by UML associations, these can be represented by using the BWW mutual binding property construct and UML-multiplicity represented by state law (Opdahl & Henderson-Sellers 2004).

In the next section, ontological modeling deficiencies identified in the traditional and component-based diagram constructs will be examined.

#### **4.4.2 Identifying ontological modeling deficiencies**

Ontological limitations of process modeling with traditional models are acknowledged by Rosemann et al. (2005) and Green and Rosemann (2000). Functional decomposition diagrams are ontologically redundant when compared to the combination of DFDs, ERDs, and context diagrams.

No representations exist for conceivable state space, lawful state space, conceivable event space, lawful transformation or lawful event space. Accordingly, problems may be encountered in capturing all the potentially important business rules of the situation.

No representations exist for acts on, history, stable state, unstable state, well defined event and poorly-defined event. Again, the usefulness of traditional diagrams for defining the scope and boundaries of the system being analyzed is undermined.

As for an analysis of ontological completeness in component models, several constructs cannot find representation in any diagrams: lawful event space, acts on, poorly defined event. A construct overload is found for the activity construct in the activity diagram that can represent a transformation, a process, a property in general or an event. Construct overload was also observed for the swimlane of the activity diagram that can represent either a thing (such as an organization) or a hereditary property of the thing (a user of the organization). Finally, overload was also identified for the trigger construct (that can represent either an event or a well-defined event).

There is construct redundancy in the case of the process ontological construct that can be either represented by a complete activity diagram or by the activity construct in an activity diagram. In the case of the activity diagram, construct excess can also be identified because the branching construct could not find any matching ontological construct. Overlaps occur in the activity diagram and the state diagram (Dussart et al. 2004).

As for the transformation of legacy requirements models using traditional diagrams into UML models, the ontological analysis reveals that all the BWW constructs represented in the traditional models can be represented in the UML models. Context diagrams can be depicted by use case diagrams as these contain all the

BWW constructs required for ontological equivalent representation. ERDs are represented by property, class, kind, state law and coupling constructs. The class diagram is able to represent the same constructs therefore able to represent the same requirements. DFDs are able to represent thing, property, transformation, process, coupling, system composition, system environment, system decomposition and level structure constructs. These could be represented with the help of activity, class and use case diagrams. Finally, functional decomposition diagrams can be depicted with the use of the same diagrams.

The use of state and sequence diagrams is redundant in the representation of structured diagrams. The main reason is that structured traditional diagrams are not able to represent states and the overlap of sequence diagrams with use case diagrams.

Based on the ontological analysis presented in tables 4-6 and 4-7, the following rules can be used when mapping traditional and component-based requirements models into BWW constructs.

For traditional requirement models, the following rules can be used for DFD, ERD and Context diagrams. For the DFD diagram:

- For every external agent, data store and external data store in the DFD, create a BWW thing construct.
- For every process in the DFD, create a BWW thing construct and a transformation BWW construct.
- For every data flow in the DFD, create a BWW internal event construct if the data flow represents an internal event or an external event BWW construct if the data flow represents an external event.
- For every process connected to an external agent and a process connected to a data store via a data flow, create a BWW coupling construct and use it couple the BWW thing constructs used to represent the process, external agent or data store.
- Create a process, system structure, sub-system, system decomposition and level structure BWW constructs for the DFD.

For the ERD diagram:

- For every entity, create a class BWW construct.
- For every relationship, create a coupling BWW construct. Use this construct to couple the BWW class constructs that represent the entities in the relationship. Use the cardinality of the relationship to establish the state law for the BWW coupling construct used to represent the relationship.

For the Context diagram:

- Create a BWW thing construct that represents the system
- For every external agent and data store, create a BWW thing construct.

- For every external agent connected to the system via a data flow, create a BWW coupling construct and use it to couple the BWW thing constructs that represent the external agent or data store.

For traditional component-based models, the following rules can be used for Use case, Class, Sequence and State diagrams.

For the Use case:

- For every Actor or Use case found, create a thing BWW construct.
- For every Use case found, create a process BWW construct.
- For every UML association, extent or include construct, create a coupling BWW construct and use it to couple the thing BWW constructs created to represent Actors or Use cases.
- Create a BWW system construct to represent the boundary of the diagram.
- For every actor found, create a system environment BWW construct.

For the Sequence diagram:

- For every object, create a BWW thing construct.
- For every message, create a BWW coupling and system structure constructs. Use the BWW coupling construct to couple the BWW thing constructs that represent the objects used in the exchange of messages.
- Create system BWW construct to represent the sequence diagram.
- Create a BWW system environment construct for every stereo type construct found in the diagram.

For the UML Class diagram:

- For every UML class in the diagram, create a BWW class construct.
- For every UML association in the diagram used to connect two UML classes, create a BWW coupling construct and use it to couple the BWW class constructs used to represent the UML classes. Use the UML multiplicity cardinalities to establish the state law for the BWW coupling construct used to represent the UML association.
- For every package with sub-system create a BWW subsystem construct
- For every package with system create a BWW system construct
- For every UML interface, create a BWW coupling construct and use it to couple the sub-systems or systems BWW constructs used to represent UML sub-system or system packages that are coupled by the interface.

For the State diagram:

- For every UML object in the diagram, create a BWW thing construct.
- For the initial state in the diagram, create a BWW unstable state construct.
- For the final state in the diagram, create a BWW stable state construct.
- For every other state in the diagram, create a BWW state construct.

- For every trigger in the diagram, create a BWW internal construct if the trigger is generated by an internal object or a BWW external construct if the event is generated by an actor.
- For every transition in the diagram from a state to another state, create a state law BWW construct and use this construct to couple the BWW state constructs used to represent the states in the transition.
- Create one conceivable state BWW construct for the diagram

Although the ontological mappings provide evidence that component-based models derived from the traditional models of a legacy system are able to represent the same BWW constructs, a normalized reference model comparison can be used as a tool to verify that the same requirements captured in the legacy system traditional requirements models are represented in the component-based models. In the next section, normalized reference models for both legacy and re-engineered requirements models will be constructed for a comparison analysis.


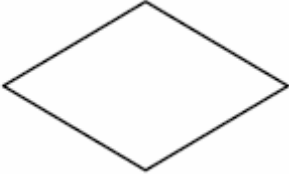
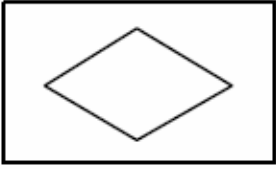
#### **4.4.3 Generation of normalized ontological meta-models**

In their book “Ontological analysis of business systems with ontologies”, Green and Rosemann (2005) proposed the use of a meta-model based on the BWW ontology for the representation of business systems. Green and Rosemann (2005) explained that these meta-models can be used to compare ontological models. In this section of the dissertation, normalized ontological models based on the Green and Rosemann (2005) BWW meta-model were constructed in order to compare legacy and re-engineered requirements models.

Normalized ontological models were developed by using the meta language defined by Rosemann and Green (2002). A summary of the language construct is defined in table 4.8. A set of processes were defined in order to generate the normalized ontological models. These processes are indicated below:

- For every diagram in the legacy and component-based models, a normalized ontological was created.
- Every construct in the legacy and component-based diagrams was mapped onto a BWW construct with the help of tables 4-5 and 4-6.
- For each construct that was not mapped onto a BWW coupling construct, an entity meta language construct was created.
- For each construct that was mapped onto a BWW coupling construct, a relationship meta language construct was created.
- Every entity in the normalized ontological model was labeled with the name of the construct that it represents and its corresponding BWW construct(s) in parentheses.
- Entities in the diagram representing BWW thing constructs were coupled with the help of relationship entities and cardinalities indicated for each relationship in the form of N1, N2 where N1 is the minimum and N2 the maximum number of instances that the state law allows for each couple.
- Any non coupling relationship among entities was represented with the help of relationship entities.

**Table 4.8 Elements of meta language for Normalized Ontological Models**

Language Construct	Explanation
	Entity type
	Relationship type combining at least two entities.
	If it is necessary to model that a relationship type is related to another element of the data model (entity or relationship type), the relationship type has to be reinterpreted.

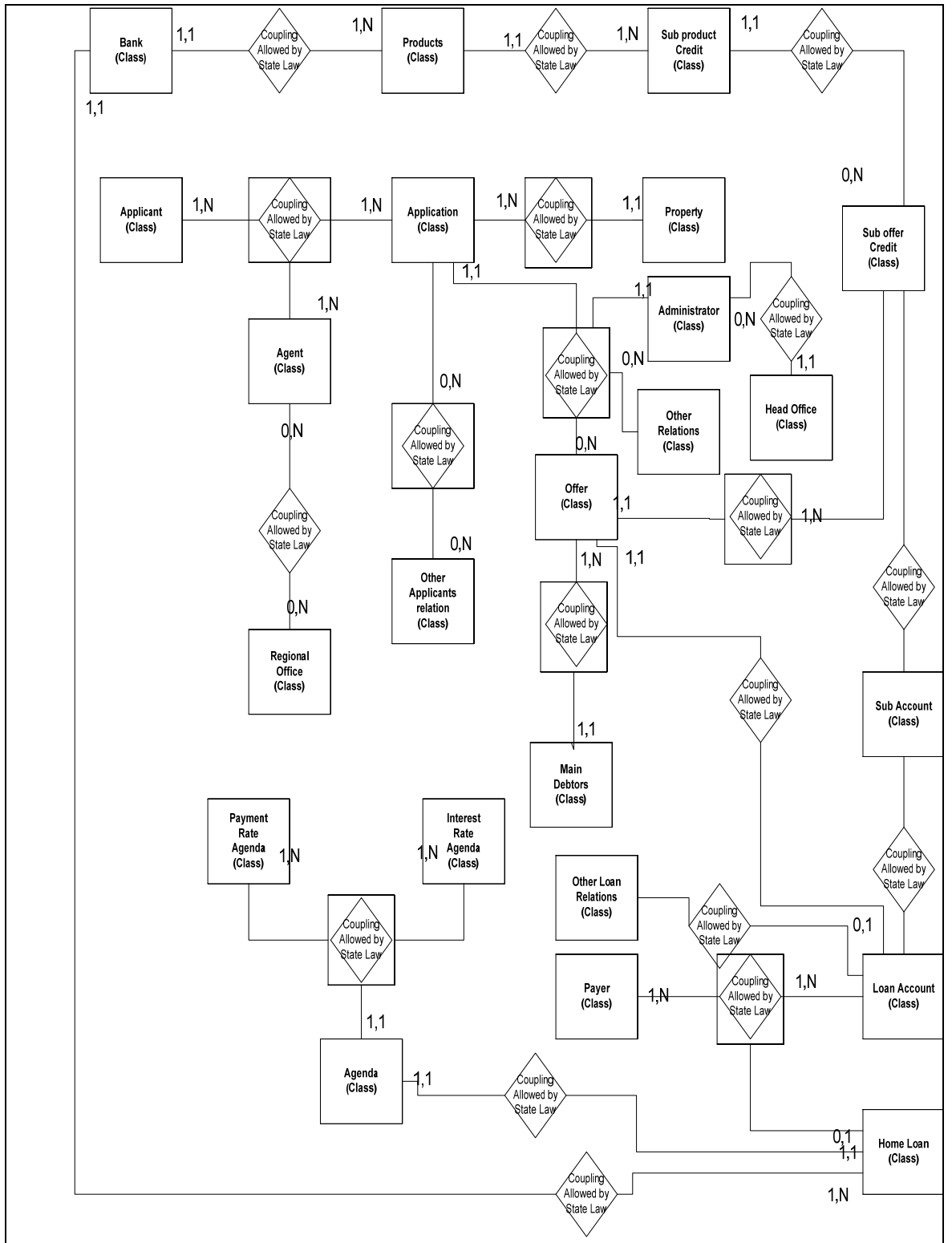
(Source: Rosemann and Green 2002 p. 80)

In Figure 4-17, the BWW meta-model for the ERD of the legacy system is depicted with the help of the BWW model proposed by Green and Rosemann (2005). Entities are represented by classes that are coupled and their relationship governed by state laws. Figure 4-18 depicts the BWW meta-model for the context diagram of the legacy system. This model represents external entities as things that are part of the system environment and system composition, these external entities are coupled with the system by using data flows in the context diagram.

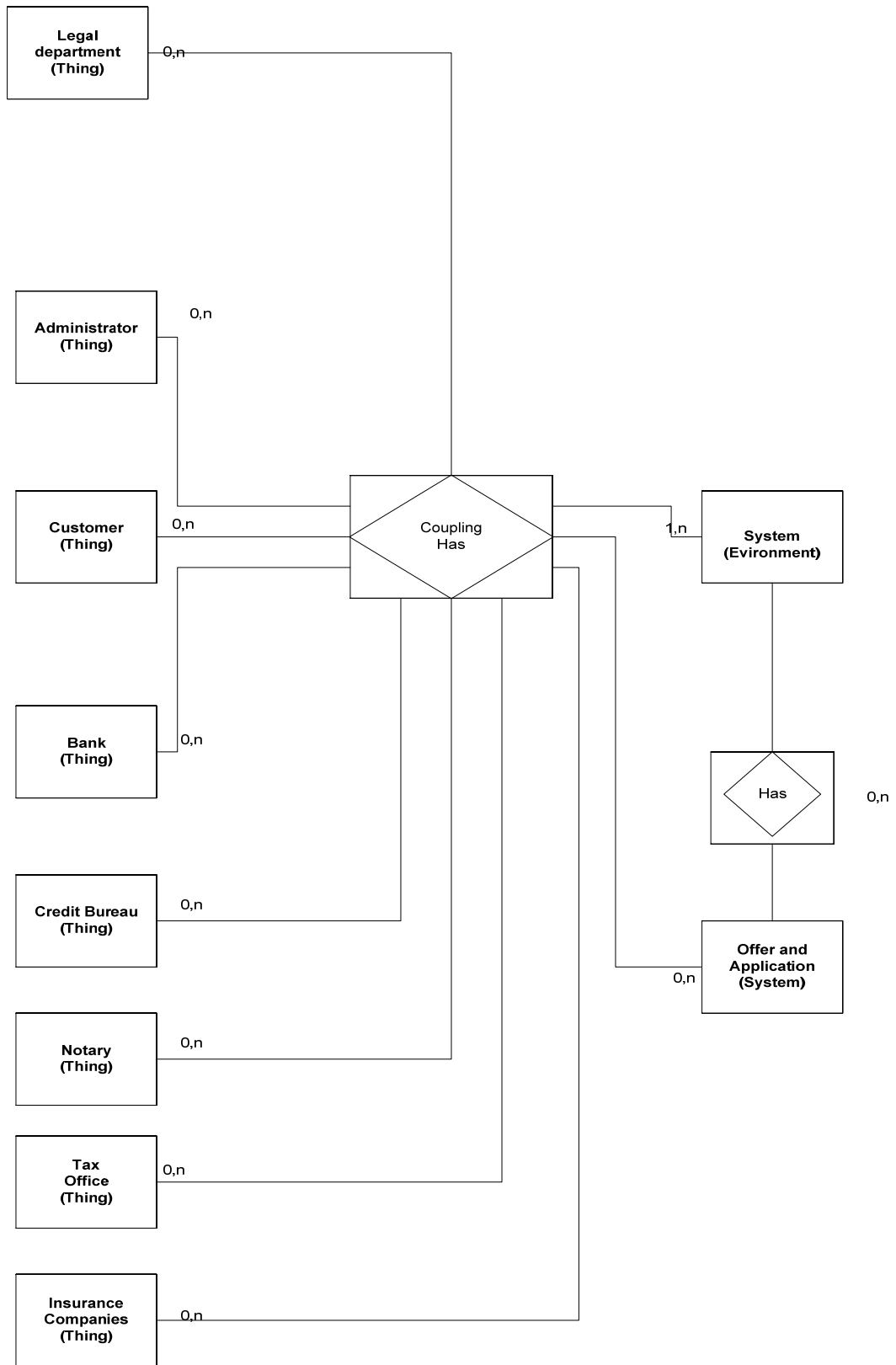
Figure 4-19 depicts the BWW meta-model for the DFD of the process offer by head office use case (Appendix D-3). External entities are mapped as things, data flows as external events or coupling when used to bind data stores with process or external entities with processes. Processes are represented by transformations. Systems, subsystem, processes, environment, structure and composition are also represented in the DFD.

Figure 4-20 shows the BWW meta-model for the use case diagram for the re-engineered system. Use cases represent things or processes that are coupled to other use cases and actors. Actors can be coupled with use cases or other actors. Actors are part of the system's environment that is part of the system.

**Figure 4.17 Normalized ontological model for the ERD of the legacy system**

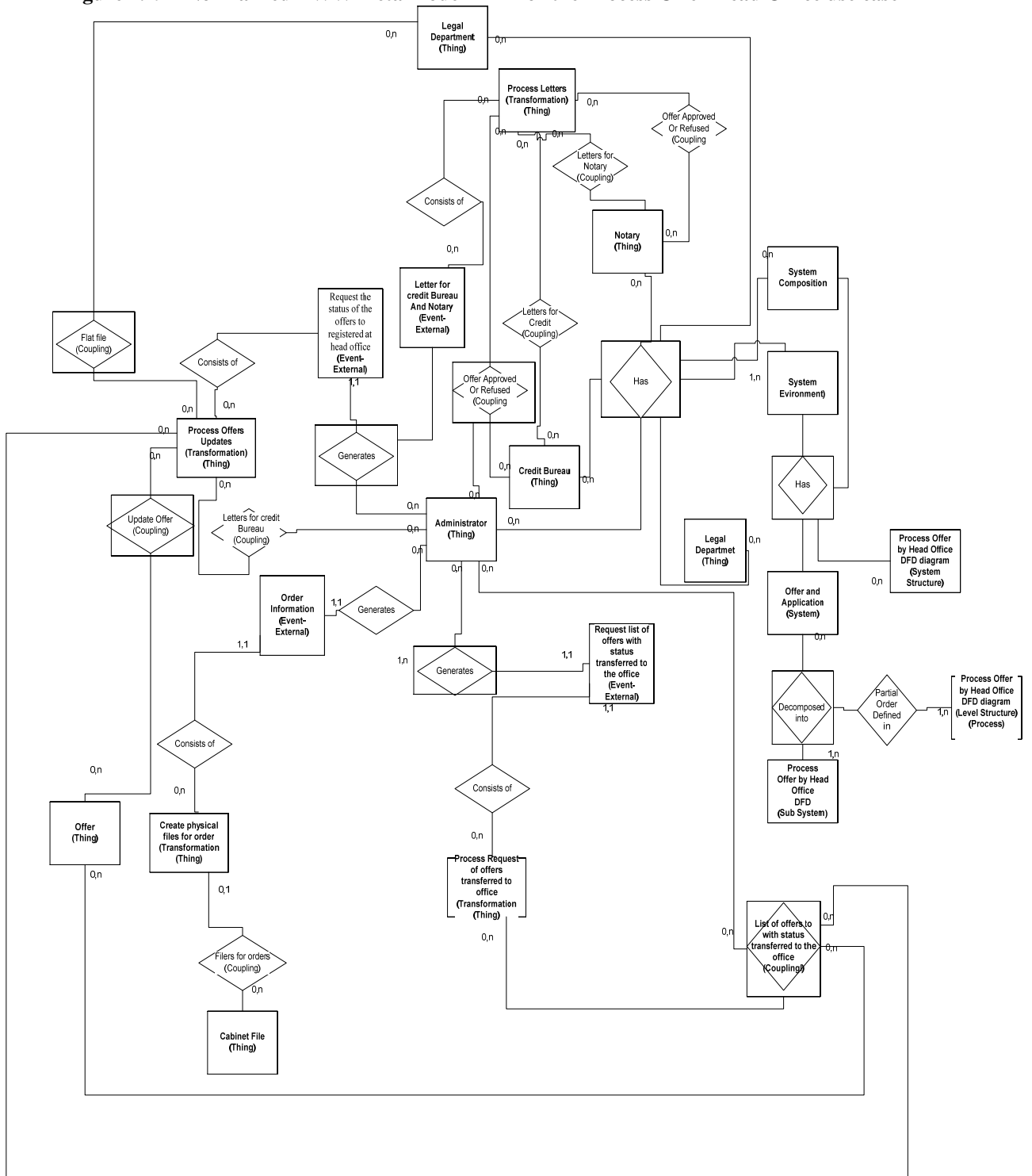


**Figure 4.18** Normalized BWW meta-model for the context diagram of the legacy system





**Figure 4.19 Normalized BWW meta-model DFD for the Process Offer Head Office use case**



**Figure 4.20 BWW meta-model for the use case diagram for the re-engineered system**

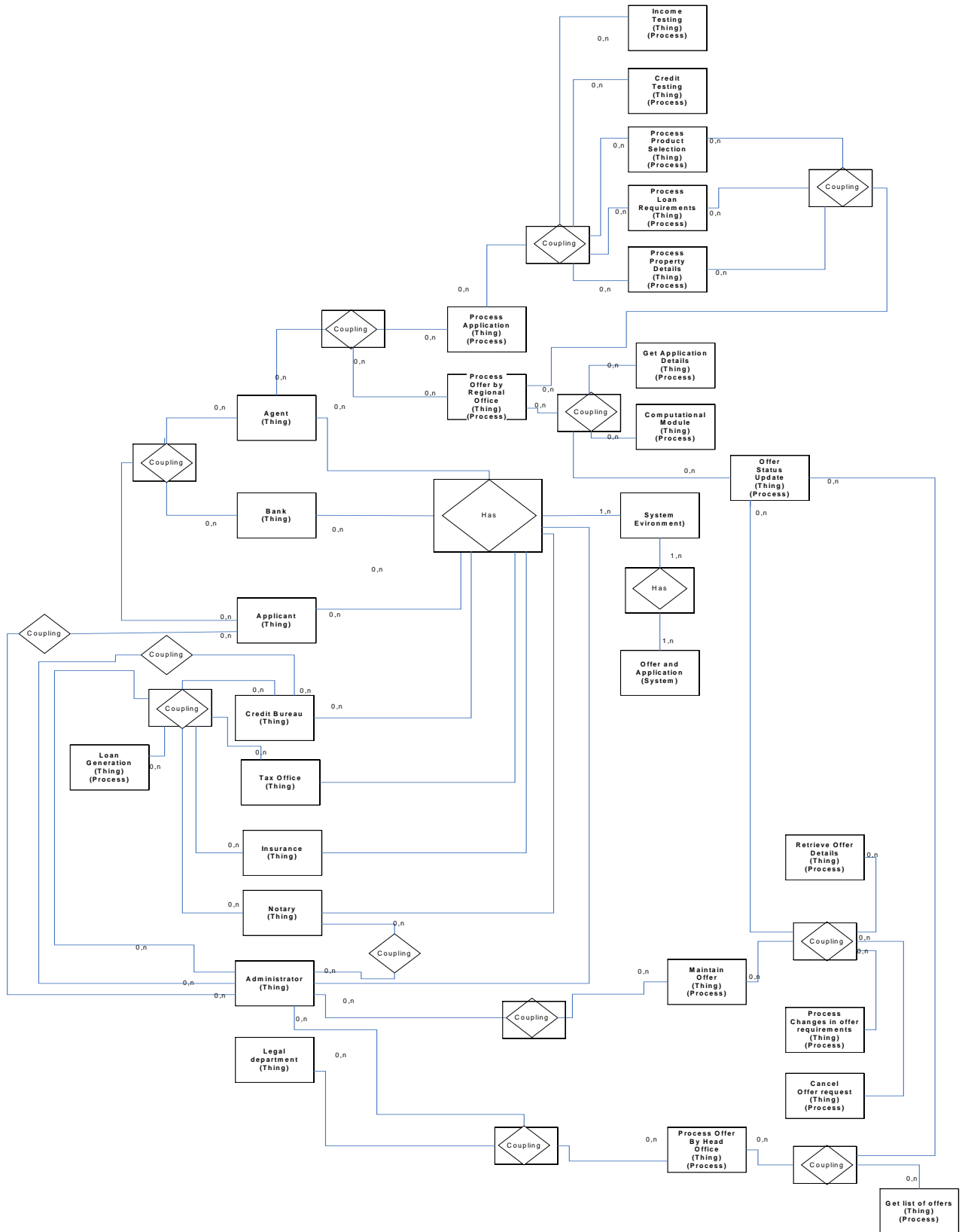
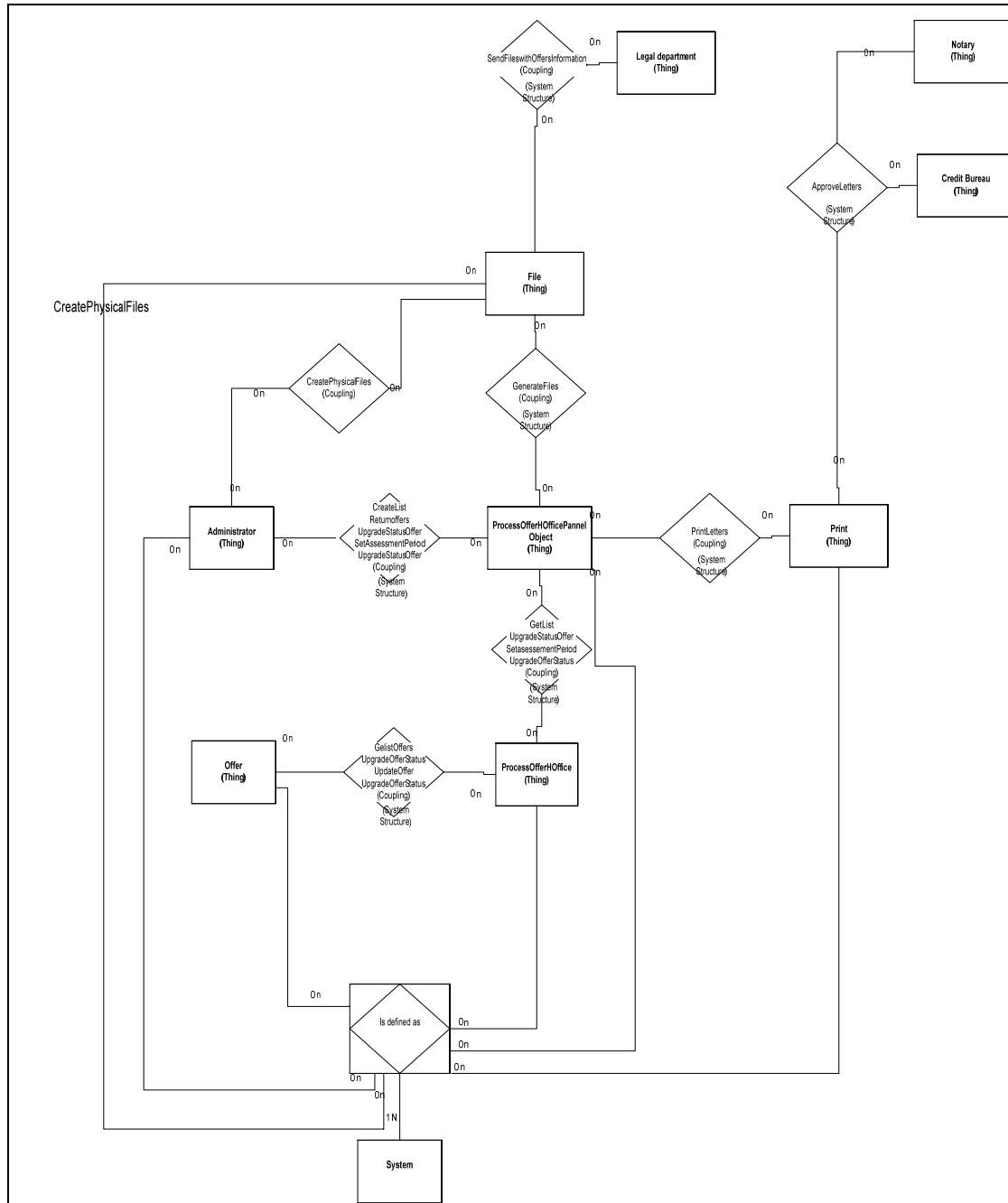


Figure 4-21 shows the BWW meta-model for the sequence diagram of the Process Offer Head Office use case for the re-engineered system. Objects represent things that are coupled by messages; messages are part of the system structure and objects part of the system.

**Figure 4.21 BWW model for the sequence diagram of the Process Offer Head Office use case**



Appendix F-1 shows the BWW meta-model for the class diagram of the database package for the Process Offer Head Office use case for the re-engineered system. The database package is represented by the BWW sub-system construct, the package is composed of classes represented by BWW classes that are coupled to other classes due to the UML association. Couplings are governed by state laws that represent the UML multiplicity between classes in the diagram.

Figure 4.22 BWW meta-model for the class diagram of the database package for the process offer by head office use case for the re-engineered system

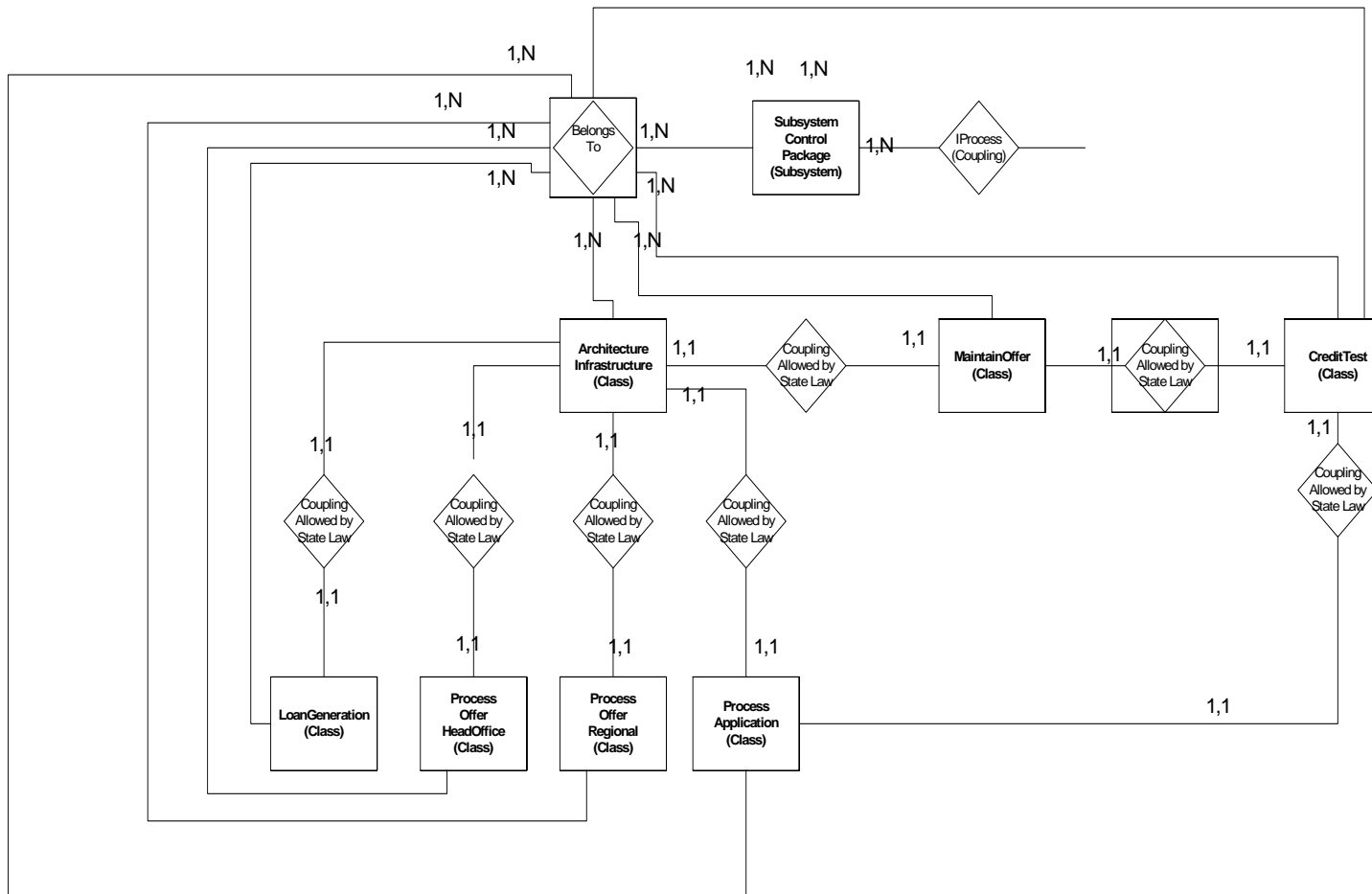


Figure 4-22 and Appendix F-2 show the BWW meta-models for the class diagram of the process control and interface packages for the Process Offer Head Office use case for the re-engineered system.

Figure 4-23 shows the BWW meta-model for the package diagram for the Process Offer Head Office use case. In this diagram, packages are represented by BWW subsystems that interface to each other by using the coupling construct.

**Figure 4.23 BWW meta-model for the package diagram for the Process Offer by Head Office use case for the re-engineered system**

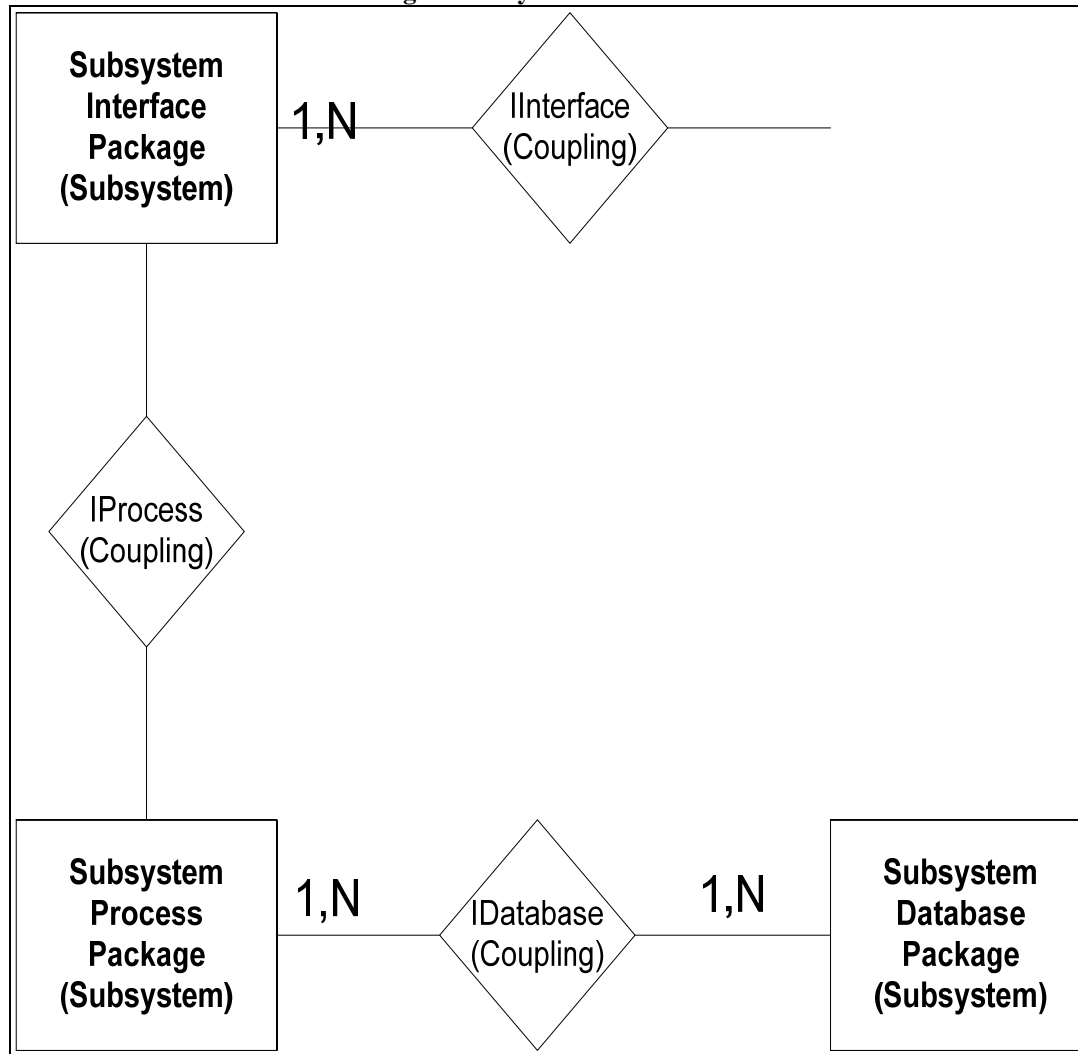


Figure 4-24 depicts the BWW meta-model for the state diagram (Appendix E-10) of the ProcessOfferHOfficePanel object identified in table 4-4. In the diagram, states are depicted as BWW state constructs. The set of possible states represent the conceivable state space. Transitions from one state to another state are governed by state laws. Triggers that allow transitions from one state to another are depicted by BWW event constructs. The final state is depicted by a stable state and the initial state as an unstable state. All triggers are part of the conceivable space event.

Figure 4.24 BWW meta-model for the state diagram of the ProcessOfferHOfficePanel object

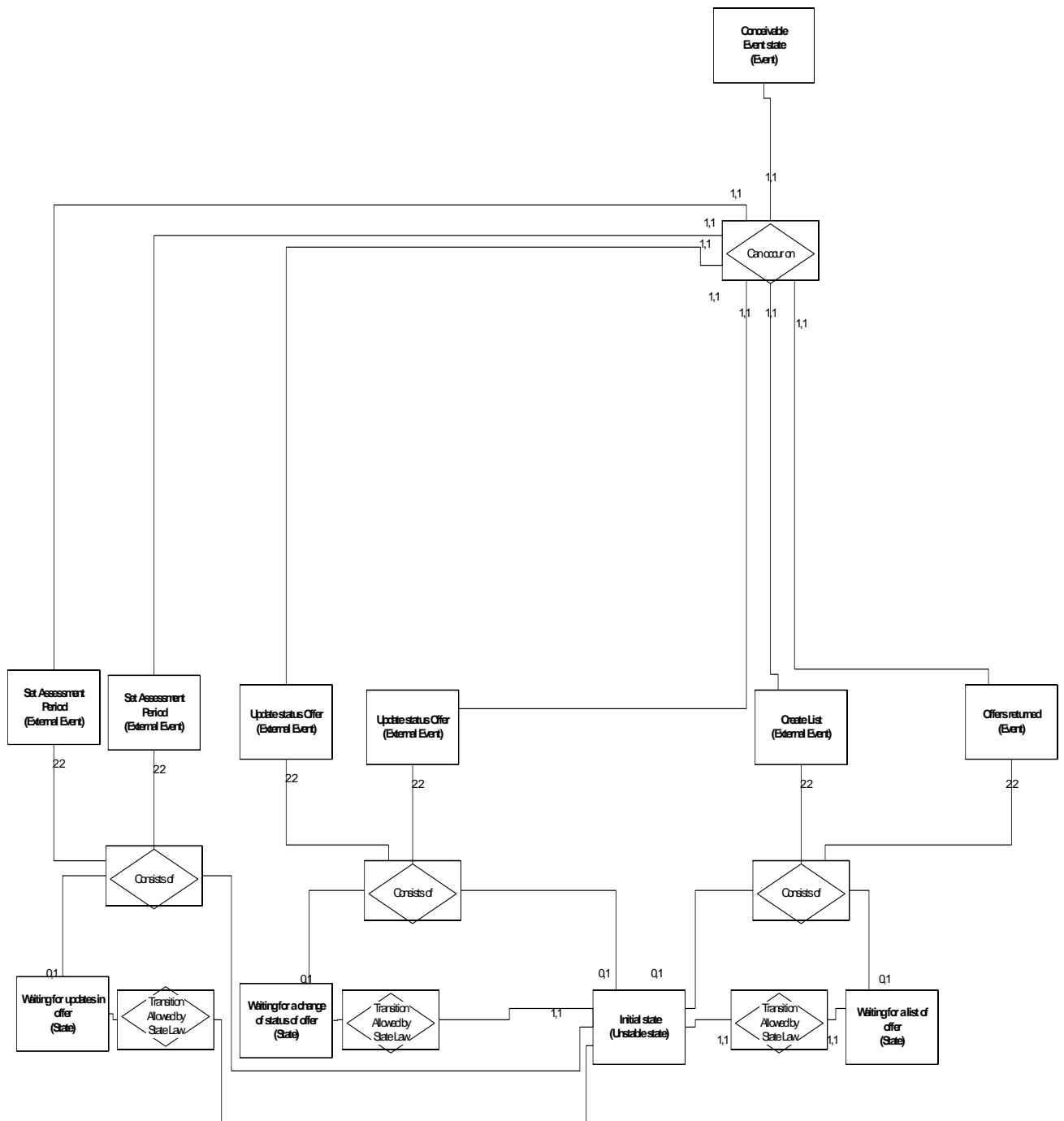
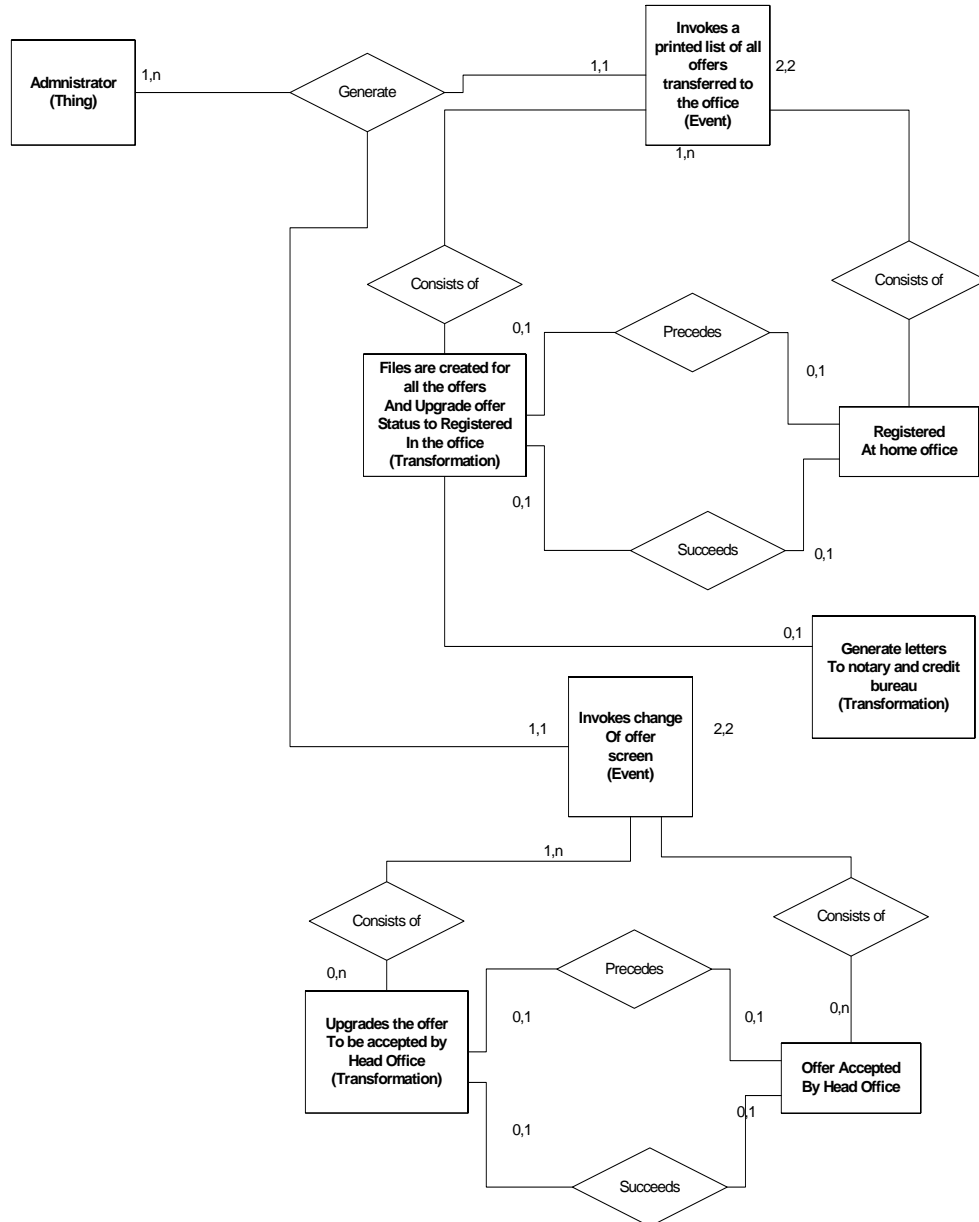


Figure 4-25 depicts the BWW meta-model for the activity diagram for the Process Offer Head Office use case (Appendix E-8). In this diagram, the swim lane UML object represents a BWW thing construct that generates an event through an activity that is being handled by another activity that represents a transformation. Activities might represent an event, a transformation or a process. The states of an object are also represented inside activities as these describe the object states.

**Figure 4.25 BWW meta-model for the activity diagram of the Process Offer Head Office**



## 4.5 Conclusions

A bank's Home Loan Information System was selected as a case study and the Offer and Application sub-system was selected as the focus for the case study.

Reverse engineering was applied to the case study in order to recover its requirements model. A concrete graph of the case study was identified as part of the reverse engineering process proposed by Jacobson and Lindstrom (1991). The concrete graph of the case study was presented by providing a description of the loan offer and application sub-system, sub-system architecture, interface architecture and business processes.

Use cases were created to document the business processes of the case study. The use cases were used to generate the legacy requirements model in terms of context, DFD and ERDs and a component-based requirements model in terms of UML was generated by using the re-engineering technique proposed by Jacobson and Lindstrom (1991).

Traditional and UML component-based diagrams were mapped onto BWW constructs as part of the ontological evaluation of requirements models. Ontological modeling deficiencies of the different diagrams analyzed were acknowledged and BWW meta-models for the legacy and re-engineered requirements models were generated.

In this chapter, models were built and evaluated in terms of ontological modeling deficiencies as part of the design science framework used for this dissertation. In the next chapter, re-engineered component-based models are evaluated in terms of their capacity to represent the same business requirements embedded in the legacy requirements model.



## Chapter 5 Analysis and Conclusions

### 5.1 Introduction

The previous chapter presented the collected data and analysis for the research while this final chapter discusses conclusions and implications of the research. The aim of this dissertation is to provide an answer to the research question, introduced in section 1.2:

*Is the resulting component-based requirements model ontological equivalent to the legacy requirements model when shifting paradigms in the re-engineering process?*

The answer to this question requires the answers to three research issues. The first deals with the possible conflict that might occur if one grammar construct in one diagram of the legacy requirements model can be mapped to more than one grammar construct in one diagram in the target re-engineered component-based requirements model. The second deals with the accommodation of all legacy requirements model grammar constructs into the re-engineered component-based requirements model and the third with the possibility of the component requirements model being complementary to the legacy requirements model, which means that the re-engineered requirements model is able to accommodate all the grammar constructs of the legacy requirements model and complement in addition more constructs that were not able to be represented in the original requirements models. Thus the three research issues, introduced in Section 1.2 and justified in Chapter 2, are:

RI1: Are the compared requirements models in conflict?

RI2: Can the business component model accommodate all the grammar constructs of the legacy requirements model?

RI3: Are the compare requirements models complementary?

Requirements models are regarded in conflict if one grammar construct in one diagram of the legacy requirements model can be mapped to more than one grammar construct in one diagram in the target re-engineered component-based requirements model. If this happens, the system modeler needs to interpret the grammar in conflict and map it to the grammar that is closer in meaning to the interpretation of the original grammar.

If the component-based requirements model is able to accommodate all the grammar constructs of the legacy business model, the system modeler only needs to follow a mapping table that can be used to map each original grammar construct into a target re-engineered component-based grammar construct.

When the component-based requirements model complements the original legacy model, the system modeler might opt to supplement the original legacy requirements model with grammar constructs that can help to represent better the system's requirements from the ontological view.

The previous chapter presented and analyzed the data by using an ontological evaluation of requirements models in order to address the three research issues with the use of a case study. This chapter builds on the literature review of Chapter 2, the methodology in Chapter 3 and the data analysis of Chapter 4 to draw conclusions and discuss the implications of this study along the lines of the three identified research issues. The research question is addressed and implications for theory and practice, as well as limitations are discussed. Finally, future research needs are identified and directions for further study are recommended.

## **5.2 Conclusions on the research issues**

This section discusses the conclusions reached about the three research issues. The conclusions of each research issue are discussed in more detail in the following subsections, 5.2.1-5.2.3. Contributions are summarized in Section 5.4. Next, the conclusions on research issue 1, ‘Conflicts’ are discussed.

### **5.2.1 Conclusion to the research issue 1: “Conflict”**

This section discusses the findings of this study with respect to research issue 1. The research issue investigates:

RI 1: Are the compared requirements models in conflict?

Requirements models are regarded as in conflict if one grammar construct in one diagram of the legacy requirements model can be mapped to more than one grammar construct in one diagram in the target re-engineered component-based requirements model.

The legacy requirements model is normally represented with three diagrams: ERD, DFD and Context. ERDs are composed of three main constructs; entities, cardinalities and relationships.

By looking at the normalized ontological model for the ERD of the legacy system of the case study for this dissertation (Figure 4-18), it was found that entities were represented by BWW classes, cardinalities represented by BWW conceivable space state laws and relationships represented by BWW couplings constructs.

The BWW constructs represented in Figure 4-18 were also found in the normalized BWW meta-model for the database package of the class diagram for the case study (Appendix F-1). Each entity in the ERD was mapped onto a class in the class diagram for the database package. In addition, each of the cardinalities was mapped as a UML multiplicity and each relationship as a UML association.

There were no conflicts encountered for the case of the ERDs re-engineered into UML class diagrams for the component-based requirements model as each construct in the legacy model was mapped onto only one construct in the UML component requirements model (Table 5-1).

**Table 5.1 Mapping of ERD into UML Class diagrams**

<b>Entities</b>	<b>UML class</b>
Cardinalities	UML multiplicity
Relationships	UML association

DFDs are mainly composed of external agents, data stores, processes and data flows. External agents were represented by both things and system environment, in BWW terms, in the normalized meta-model DFD for the Process Offer Head Office use case of the case study (Figure 4-19).

External agents for the DFD were mapped in the re-engineered requirements model as things and system environment BWW constructs in the meta-model of the use case diagram for the re-engineered system. In the case of external agents, there were no conflicts detected in the re-engineering process (Figure 4-24).

In Figure 4-20, the normalized BWW meta-model for the DFD of the Process Offer by Head Office use case identifies the data stores offer and cabinet as BWW thing constructs. The same constructs are identified as BWW things constructs in figure 4-25 for the BWW model for the sequence diagram of the Process Offer Head Office use case. This suggests that for the case of data stores, there were no conflicts as these data stores originally represented in the DFD of the legacy system are also represented by objects in the sequence diagram of the target re-engineered system.

Data flows have different interpretations depending on their use. They could represent a BWW external event or internal event constructs, external if initiated by an external agent and internal if initiated by a process (Table 4-4). In addition, they could represent BWW coupling constructs if they are used to bind processes with external agents or data stores (Table 4-4).

In the normalized BWW meta-model for the DFD for the Process Offer Head Office use case in Figure 4-23, it is possible to see that the data flow that represents the “Request list of offers with status transferred to the office” initiated by the administrator, is represented as an external event in the meta-model of Figure 4-23. Internal and external events represented originally in the legacy requirements models as data flows, can be represented in the re-engineered component-based requirements model as triggers in state diagrams or activities in activity diagrams (Table 4-6).

The “Request list of offers with status transferred to the office” event was mapped in the BWW meta-model for the activity diagram of the Process Offer Head Office use case of Figure 4-26. However, this diagram is unable to differentiate between internal and external events as all activities represent BWW events as indicated in Table 4-6, therefore this type of diagram cannot be used to represent data flows.

In the case of the state diagram, the use of stereotypes is used to differentiate between external and internal events. For the case of the “Request list of offers with status transferred to the office” event represented in the DFD by a data flow, this is represented in the BWW meta-model for the state diagram of the ProcessOfferHOfficePanel object of Figure 4-25 as an external event.

Data flows could also represent a BWW coupling construct if a data flow is used to bind a process with an external agent, another process or data store.

In the normalized BWW meta-model of the Process Offer Head Office use case (Figure 4-20), the administrator (external agent) was mapped into a BWW thing construct. This external agent receives a data flow “List of offers with status of transferred to the office” from the process “Process request list of offers transferred to the office” that is represented as a BWW coupling construct in the meta-model of Figure 4-20. In this case, the data flow diagram is used to bind a process with an external agent.

A similar representation is found in the component requirements model. Figure 4-22 shows the BWW meta-model for the sequence diagram of the Process Offer by the Head Office use case, in this diagram it is possible to see that the Administrator represented by a BWW thing construct is coupled via “return offer” message from the ProcessOfferHOfficePanel Object represented as a BWW thing construct, this object is also coupled to the ProcessOfferHOffice object that is derived from a class of the same name that is part of the subsystem process package. In Appendix F-4, it is possible to see that the ProcessOfferHOffice class is represented by a BWW class construct that is part of the subsystem process package. This class has a “GetList” operation that is represented as a BWW transformation construct in the same diagram. The GetList transformation represents the same “Process request list of offers transferred to the office” transformation in the legacy requirements model as this transformation is in charge of retrieving the list of offers transferred to the head office. In this case, there were no conflicts encountered.

In the case of data flows binding processes with data stores, it is possible to see in the normalized BWW meta-model of the Process Offer Head Office use case (Figure 4-20) that the process “Process request list of offers transferred to the office” that is represented as a BWW transform construct in the meta-model is bound to the data store offer represented as a BWW object construct by using the data flow “List of offers with status of transferred to the office”. The component-based requirements model shows a similar representation in Figure 4-22 which shows the BWW meta-model for the sequence diagram of the Process Offer Head Office use case. This diagram shows that the object ProcessOfferHOffice represented by a BWW object construct is coupled to the offer object represented as a BWW object construct. Appendix F-4 depicts the ProcessOfferHOffice class that is represented by a BWW class construct that has the “GetList” operation that is represented as a transformation BWW construct in the same diagram. As this “GetList” transformation represents the same “Process request list of offers transferred to the office” transformation in the legacy requirements model, this suggests that there is no conflict for the representation of the data flow “List of offers with status of transferred to the office” of the legacy requirements model.

Data flows can also be used to represent couplings between DFD processes. Processes in DFDs can be interpreted as BWW things and transformations according to table 4-6. This is consistent with the interpretation of Want and Weber (1989) which states that the things in the system are processes and data. In the normalized BWW meta-model of the process Offer by Head Office use case (figure

4-20), the process “Process Offers updates” is coupled via the data flow “List of offers with status of transferred to the office” with the process “Process Request List of Offers Transferred to the Office”. In the component-based requirements model, this is represented in the BWW model for the sequence diagram of the process offer by the head office use case (Figure 4-22) by the BWW object ProcessHOffice coupled by a message to the BWW object Offer. This representation is ontological equivalent as the ProcessHOffice object is derived from the class of the same name that appears in the BWW meta-model subsystem diagram for the Process Offer Head Office use case (Appendix F-4) and has the transformation “GetList” that performs the activities required to get the list of offers transferred to the office and it is coupled to the Offer object via the IDatabase interface which has the transformation “UpdateOfferStatus” that performs the process that updates the offer status. In this case, both legacy and component-based have the same representation in BWW terms.

As shown in table 5-2, the data flow construct can be mapped onto two UML constructs depending on the data flow interpretation. This shows a potential conflict when re-engineering DFDs into component-based UML diagrams.

**Table 5.2 Potential conflict in the representation of data flows in the re-engineered component model**

Data flow interpretation	UML representation
Internal Event	UML trigger
External Event	UML trigger
Coupling between process and external agent	UML message
Coupling between process and data store	UML message
Coupling between process and process	UML message

Context diagrams are mainly composed of external agents, data flows and the main system. Figure 4-19 depicts the normalized meta-model for the context diagram of the legacy system. In this figure, external agents are represented by BWW things and system environment. These were mapped one to one in Figure 4-24 where each external agent was mapped onto one actor in the use case diagram that represents a BWW thing and is part of the system environment. The system in the context diagram was mapped as the system boundary in the use case diagram in Figure 4-24 as both represent BWW system constructs. The data flows in the context diagram were mapped as UML associations in the use case diagram as both represent BWW coupling BWW constructs that bind external agents with the system. Based on the previous analysis, it was found there were no conflicts for the case of the context diagrams re-engineered into UML case diagrams (Table 5-3).

**Table 5.3 Mapping of Context Diagrams onto UML Use Case diagrams**

External agents	Actor
System	System boundary
Data flows	UML associations

Class diagrams are not the only ones that can represent BWW transformation constructs; activity diagrams are also able to depict this type of construct. The process “Process Letters” depicted as a transformation BWW construct in the normalized BWW meta-model of the DFD for the Process Offer Head Office use case (Figure 4-20) was mapped as “Generate letters” process depicted in the normalized BWW model for the activity diagram of the Process Offer Head Office use case (Figure 4-26). However, the activity diagram is not capable of representing the administrator that is coupled with this process by using a data flow as depicted by Figure 4-20 as activity diagrams are not capable of representing BWW thing and coupling constructs. On the other hand, Figure 4-22, representing the BWW model for the sequence diagram of the Process Offer Head Office use case is capable of representing the administrator as a BWW thing construct that is coupled to the object ProcessOfferHOfficePanel represented as a BWW thing construct. This object is coupled with the Print object depicted by a BWW thing construct that is derived from the class of the same name that has the operation print letters that is in charge of generating the letters as shown in Appendix F-4 and represented as a BWW transformation construct. This shows the redundancy of the use of activity diagrams in the re-engineering of legacy requirements models as class diagrams are able to represent operations that encapsulate the business processes depicted by legacy systems.

Although a process in the DFD can be mapped onto an activity in the activity diagram or an operation in the class diagram, this does not represent a conflict as the activity and operations UML constructs belong to two different UML diagrams.

In conclusion, the data flow is the only construct of the legacy requirements model that presented conflict in the re-engineering process. Suggestions on how to deal with this conflict will be included in section 5.2.4 as part of the conclusions of this dissertation.

### **5.2.2 Conclusion to the research issue 2: “Grammar accommodation”**

This section discusses the findings of this study with respect to research issue 2. The research issue investigates:

RI 2: Can the business component model accommodate all the grammar constructs of the legacy requirements model?

In the previous section, it was demonstrated that component-based requirements models were able to accommodate all the grammar constructs of the legacy requirements model. The results of this mapping are summarized in Table 5-4.

**Table 5.4 Traditional diagrams representation in UML component diagrams**

Type of diagram	Diagram element	UML representation
Context Diagram	External Agents	Actor (Use case diagram)
	Data Flow	UML association (Use case diagram)
	System	System Boundary(Use case Diagram)
DFD	External agents	Actor (Use case diagram)
	Data stores	Object (Sequence diagram)
	Data flows (internal and external events)	Triggers (State diagram)
	Data flows (external agent and process coupling)	UML message
	Data flows (process and data store coupling)	UML message
	Data flows (process and process coupling)	UML message
	Process	Activities (Activity diagram) UML operations (Class diagram)
ERD	Entities	UML class (Class diagram)
	Cardinalities	UML multiplicity (Class diagram)
	Relationships	UML association (Class diagram)

As indicated in Table 5-4, the component-based requirements model is able to accommodate all grammar constructs of the business legacy model provided that the interpretation (event or coupling) of the data flow is known in order to map it to the appropriate UML construct in the component-based requirements model. Table 5-4 shows that data flows can have two interpretations and it is necessary to know the interpretation used in the diagram in order to map it to the appropriate UML representation. Each of the other diagram elements of the business legacy model can be mapped to one UML representation as indicated in Table 5-4.

### 5.2.3 Conclusion to the research issue 3: “Complementary”

This section discusses the findings of this study with respect to research issue 3. The research issue investigates:

RI 3: Are the compared requirements models complementary?

If the re-engineered requirements model is able to accommodate all the grammar constructs of the legacy requirements model and supplement more constructs that were not able to be represented in the original requirements models, this can be regarded as complementary.

As identified in Section 4.4.2, the traditional models offer ontological limitations when representing business requirements. During the ontological analysis it was found that no representations exist for conceivable space, lawful state space, conceivable event space, lawful transformation or lawful event space constructs. In addition, no representations exist for acts on, history, stable state, unstable state, well defined event and poorly-defined event constructs. Accordingly, problems may be encountered in capturing all the potentially important business rules of the situation in traditional requirements models.

In Section 4.4.2, it was found that the ontological completeness in component models is lacking as several constructs cannot find representation in any diagrams: lawful event space, acts on, poorly defined event.

However, the ontological mappings listed in tables 4-4 and 4-5 provide evidence that component-based models derived from the traditional models of a legacy system are able to complement the legacy requirements model as they are able to represent many of the BWW constructs that the legacy requirements model was not able to represent.

Table 5-5 shows the ontological constructs that are not represented in the traditional requirements models but that are represented in the component-based requirements models.

**Table 5.5 UML constructs that complement traditional requirements models**

BWW constructs not represented in the traditional requirements models	UML constructs in component models used to represent the BWW constructs not represented in the traditional requirements models
Conceivable state space	State machine of the UML state diagram
Lawful state space	Sub states of the UML state diagram
Conceivable event space	All triggers of the UML state diagram
Lawful transformation	Guard conditions on transitions of the Activity diagrams
History	Shallow history in the UML state diagram
Stable state	Final state in the UML state diagram
Unstable state	Initial state in the UML state diagram
Well defined event	Trigger in the UML state diagram

Based on the table above, it is possible to conclude that component-based requirements models are able to complement traditional requirements models by including constructs that are not able to be represented in the original requirements model. This supports the idea that the component-based models are more ontologically complete than the traditional models and thus makes them better options to document business requirements. However, there are still three BWW constructs that cannot be represented with the present UML grammar and might be possibly included in future versions of UML.

The other issue is that although the representation of BWW subsystems are included in the original legacy requirements model with the use of DFDs, these subsystems



were not designed with the intention of reusability unlike components in the case of component-based systems. The notion of component is not supported as such in legacy systems and constitutes a complement to the original legacy systems.

In conclusion, the component-based requirements model is able to complement the legacy model by adding more constructs that are able to better cover the BWW ontology as a proof of being able to represent requirements better in ontological terms.

#### **5.2.4 Conclusions to research question**

Based on the discussion of the three research issues in sections 5.2.1-5.2.3, this section proposes an answer to the research question:

Is the resulting component-based requirements model ontological equivalent to the legacy requirements model when shifting paradigms in the re-engineering process?

In Section 5.2.1, it was found that there was a conflict with the use of data flows as these can represent events (internal or external) and also couplings between processes to data stores, processes with processes and processes with external agents (Wand & Weber 1989).

Although this might be seen as a potential conflict in the re-engineering process, the problem of mapping the data flow with UML triggers or UML messages (Table 5-3) can be eliminated if the interpretation is known before the legacy requirements model is re-engineered. The interpretation can be easily found by reading the use cases or business process descriptions of the legacy requirements model and a rule can be used to solve this conflict. The rule can require mapping the data flow as a UML trigger if it is interpreted as an event, and mapping it as a UML message if the data flow is interpreted as coupling.

In the Section 5.2.2, it was found that the re-engineered component requirements model was capable of representing all the legacy requirements model constructs. Table 5-4 shows the mapping of all the legacy requirements model constructs onto the component-based requirements model as a proof of this.

In addition, in Section 5.2.3 it was discussed that the component-based requirements model is able to complement the legacy requirements model and therefore able to better represent requirements in ontological terms.

Based on the results of the three research issues discussed in this section, the answer to the research question is that the resulting component-based requirements model is ontological equivalent to the legacy requirements model when shifting paradigms in the re-engineering process and is able to represent the same requirements. The answer to this question verifies that a re-engineered component-based requirements model generated using UML grammar is able to represent the requirements encapsulated in a legacy system requirements model represented by the traditional DFD, ERD and Context diagrams models.

### 5.3 Implications for theory

The main contribution to theory of this study is a systematic approach for the re-engineering of legacy systems into component-based systems in order to generate ontological equivalent requirements models. Based on the ontological analysis in this dissertation, the following rules can be used when re-engineering legacy systems in order to ensure the same representation of requirements in the re-engineered requirements models.

For the case of the context diagram in the legacy requirements model, this can be represented with the help of the use case diagram in the component-based model by following the rules below:

- For every external agent, create an actor that interacts with the system in the use case diagram.
- For every data flow, create a UML association that will bind actors with the system.
- For the case of ERD in the legacy requirements model, these can be represented with the use of UML class diagrams in the component-based model by following the rules below.
- For every entity in the ERD of the legacy requirements model, a class should be created in the class diagram of the component-based model.
- Relationships in the ERD should be respected in the class diagrams and implemented with UML associations.
- Cardinalities in the ERD should be respected in the class diagrams and implemented with UML multiplicity constructs.

For the case of DFD in the legacy requirements model, these can be represented with the use of sequence diagrams, state diagrams and class diagrams by following the rules below:

- For every external agent, create an actor in the sequence diagrams.
- For every process, create an operation in an appropriate class of the class diagram that implements the process in the DFD.
- For every data flow interpreted as an internal or external event, create a trigger in the state diagram of the appropriate object in charge of generating the event. If the event is external use a stereotype to indicate this in the trigger.
- For every data flow interpreted as coupling, create a message in the sequence diagrams. Data flows used to couple external agents with processes should be represented in the sequence diagram as a message between the actor representing the external agent and the object that is in charge of implementing the process by using the operation created for this in rule 2. Data flows used to couple processes with data stores should be represented in the sequence diagram as a message between the object implementing the process and an object representing the data store. Data flows used to couple a process with another process should be implemented by a message between an object implementing the first process and another object implementing the

second one. If both processes are implemented by the same object this could be represented by a message being sent from the object to itself.

Table 5-6 shows the UML component-based diagrams required to represent the legacy requirements model.

Table 5-6 UML component diagrams

Type of diagram	UML representation
Context Diagram	Use case diagram
ERD	Class diagram
DFD	Sequence diagrams Class diagrams State diagrams

These sets of rules represent the main contribution to theory of this dissertation. The use of these rules would ensure an ontological equivalence when re-engineering legacy requirements models into component-based requirements models.

#### 5.4 Contributions of this study

Five levels of contribution are distinguished. The five levels of contribution are as follows:

First, the application of the BWB (Wand and Weber 1995) ontology for the re-engineering of information systems was demonstrated during the study. An ontological approach to re-engineering is an important contribution of this dissertation as the literature review revealed that little research has been carried out applying this ontology in the re-engineering of legacy systems.

Second, this study demonstrated the application of the work of Wand and Weber (1989) with regards to interpretation of DFDs and ERDs in BWB terms, and the BWB representation for dynamic aspects of UML of Dussart et al. (2004) and Opdahl and Henderson-Sellers (2002b) as the basis for an ontological approach to re-engineering of information systems.

Third, rules were generated based on the ontological analysis in this dissertation as provided in Section 5.3. These rules are a clear contribution as they can be used for every re-engineering project in order to ensure that the requirements of the legacy requirements model will be represented in the target re-engineered requirements model.

Fourth, based on the results of this study, a recommendation to create an improved version of UML that can include UML grammar for the representation of three BWB constructs: lawful event space, acts on, and poorly defined event, is made as these constructs are not currently represented by UML.

Fifth, the results of the study supports the idea that the component-based models are more ontologically complete than the traditional models and thus makes them better option to document business requirements.

## **5.5 Implications to Information Systems Practice**

The nature and intention of this study, in addition to contributing to theory, is to provide the tools for information systems re-engineering of legacy systems into component-based systems. The dissertation demonstrated the use of the BWL ontology as the main framework for information systems re-engineering. The mapping (table 5-4) can be used to transform constructs in the traditional requirements model into UML constructs for the target re-engineered component-based requirements model. In addition, rules identified in section 5.3 can be used as the basis of information systems re-engineering for legacy systems.

BWL meta-model diagrams were introduced as useful tools in the re-engineering process as these are footprints of the models in terms of ontological representations. These footprints can be used for comparison of models and verification of representation of business requirements of the original legacy requirements model in the target re-engineered component based requirements model.

The dissertation verified that a re-engineered component-based requirements model based on UML grammar is able to represent business requirements embedded in legacy requirements models originally created to be implemented with structured languages.

The study also revealed that the required UML diagrams in the re-engineering of legacy systems are use case, sequence, state and class diagrams. The study demonstrated the redundancy of the use of activity diagrams in the re-engineering of legacy requirements models as class diagrams are able to represent operations that encapsulate the business processes depicted by legacy systems.

This research study also confirmed the study of Dussart et al. (2004) that UML grammar is ontologically incomplete. Therefore, the complete set of BWL constructs would need to be included in future versions of UML in order to allow for models that can better represent the reality captured in business requirements.

## **5.6 Limitations**

One of the most important limitations of this study was that the legacy system selected was poorly documented in terms of functional processes and technical design. This problem was overcome by recovering the requirements model of the case study by applying a reverse engineering approach as specified in the Jacobson and Lindstrom (1991) methodology. This methodology used data collection methods of interviews, direct observation and secondary data.

Another important limitation was that the study only concentrated on a re-engineering project that does not include new requirements. This was required as the study needed that both legacy and re-engineered requirements models to be ontological equivalent for direct comparison. However, the results could be

extended to projects that include new requirements as it is always possible to use the proposed methodology to transfer legacy requirements and verify that the requirements are represented in the re-engineered requirements models, then new business requirements could be included once the verification is complete.

Another issue is the generalisability of the study due to the use of only one case in the research. Many critics of the case study approach question the academic value of this method as they argue that generalisation from results is not applicable (Bell 1992). However, Bassey (1981) insists that "the relatability of a case study is more important than its generalisability". He continues, stating that "if case studies are: carried out systematically and critically, if they are aimed at the improvement of education, if they are relatable, and if by publication of the findings they extend the boundaries of existing knowledge, then they are valid forms of educational research" (Bassey 1981 p.86). For this study, the researcher followed the suggestions of Bassey in order to minimize the generalisability limitation of the study.

Another limitation was that the intense exposure to study of the case could have biased the findings because the researcher who conducted the re-engineering was the same person who conducted the evaluation analysis (Soy 1996). However, this was minimized by following a rigorous methodology.

## **5.7 Implications for Methodology**

This study has methodological implications; first, design science was the main methodology for this dissertation and it proved to be useful for the research of information systems re-engineering. The research activities that March and Smith (1995) identify for this methodology are build and evaluate and these were fundamental for this study as the first was used for construction of business and BWW normalized models and the second was used in the evaluation of these models for equivalency of business requirements.

The dissertation showed that the methodology by Fettke and Loos (2003) can be used as a framework for the evaluation of requirements models for equivalency of representation of business requirements. Another important implication of this dissertation is the use of the Rosemann and Green (2002) meta-models as the primary tool to represent the normalized requirements models in BWW construct terms. The Fettke and Loos (2003) methodology was based on the use of the concept of normalized requirements models and the Rosemann and Green (2002) meta-models proved to be an effective way to generate this type of model.

In summary, this dissertation was based on the design science methodology that created BWW models based on the Rosemann and Green (2002) meta-models, evaluated these models by using the Fettke and Loos (2003) ontological methodology and showed these to be effective tools for information systems re-engineering.

## **5.8 Directions for future research**

This study has set the theoretical background for future research in the development of automated tools for the re-engineering of information systems. A software tool could be constructed to compare legacy and re-engineered requirements models based on the methodology proposed and the rules generated in this dissertation. This software tool could translate the legacy and component-based requirements models into ontological normalized referenced models that could be used for comparison.

Although meta-models were used as the main tool for the creation of ontological normalized referenced models, these have limitations as they cannot be easily programmed and implemented in automated tools.

Future research could explore the use of meta-languages such as XML that could be used to represent requirements models in BWW terms and could be easily interpreted and visualized by software tools.

Future research could also be concentrated in the development of an ontological requirements modeling language that could be used to model business requirements. This modeling language could be independent of the programming paradigm in order to avoid possible conflicts and omissions when re-engineering information systems.

As this dissertation only concentrated on a case of re-engineering with no new requirements, further research can be done for the inclusion of re-engineering projects that include new requirements.

In addition, the ontological deficiencies detected in the UML grammar could be incorporated in future versions of UML in order to make it more ontologically complete.

## **5.9 Summary**

The study evaluated the requirements models generated by the component-based and traditional approaches when shifting paradigms in the re-engineering process in order to verify that the re-engineered requirements model is capable of representing the same business requirements of the legacy system. A legacy system was selected as part of the case study and re-engineered using the component-based paradigm with the help of UML notations. The requirements model of the legacy system was recovered using reverse engineering and compared to the component-based requirements model using normalized reference models generated with the help of BWW transformation maps. These maps revealed that the re-engineered requirements models in UML are capable of representing the same business requirements of the legacy system. The identified UML diagrams required to represent the legacy models were class, use case, state and sequence diagrams.

A set of rules were suggested for re-engineering legacy systems into component-based information systems in order to ensure the same representation of legacy system's requirements in the re-engineered requirements model.

Finally, this research included directions of future research that put emphasis on the development of automated software tools for systems re-engineering that could implement the rules suggested in this study and the ontological methodology approach used for this dissertation.

## REFERENCES

- Ackermann, J., Brink, F., Conrad, S., Fettke, P., Frick, A., Glistau, E., Jaekel, H., Kotlar, O., Loos, P., Mrech, H., Raape, U., Ortner, E., Overhage, S., Sahm, S., Schmietendorf, A., Teschke, T. & Turowski, K. 2003, Standardized Specification of Business Components, University of Augsburg, Augsburg.
- Anderson, J.G., Aydin, C. E., & Jay, S. J. 1994, Evaluating Health Care Information Systems: Methods and Applications. Sage publications, USA.
- Avison, D. E. & Fitzgerald, G. 1995, Information Systems development: methodologies, techniques and tools, McGraw-Hill, New York.
- Babiker, E., Simmons, D., Shannon, R. & Ellis, N. 1997, 'A model for re-engineering legacy expert systems to object-oriented architecture', Expert Systems with Applications, vol.12, no.3, pp.363- 371.
- Bassey, M. 1981, 'Pedagogic research: on the relative merits of search for generalisation and study of single events', Oxford Review of Education, vol.7, no.1, pp. 73-79.
- Bell, J. 1992, Doing your research project, Milton Keynes: Open University Press, UK.
- Benbasat, I., Goldstein, D. & Mead, M. 1987, 'The Case Research Strategy in Studies of Information Systems', MIS Quarterly, vol.11, no.3, pp. 368-386.
- Bennett, K. 1995, 'Legacy Systems: Coping with Success', IEEE Software, vol.12, no.1, pp.19-23.
- Bisbal, J., Lawless, D., Wu, B. & Grimson, J. 1999, 'Legacy information systems: issues and directions', IEEE Software, vol.16, no.5, pp. 103-111.
- Bloor Research 2001, Understanding business goals, [Online] Available <http://www.it-director.com/business/content.php?cid=2254>, [Accessed 19 November 2007].
- Brown, A.W. 2000, Large-Scale, Component-Based Development, Prentice Hall, New Jersey.
- Brookshear, J.G. 2000, Computer Science: an overview, Addison-Wesley, Boston.
- Bunge, M. 1977, Treatise on Basic Philosophy: Volume 3: Ontology 1: The furniture of the world, Reidel, Boston.
- Bunge, M. 1979, Treatise On Basic Philosophy: Volume 4: Ontology II: A World of Systems, Reidel, Dordrecht.
- Bunge, M. 1999, Dictionary of Philosophy. Prometheus Books, Amherst.



- Burstein, F. & Gregor, S. 1999, The Systems Development or Engineering Approach to Research in Information Systems: An Action Research Perspective, Proceedings of the 10th Australasian Conference on Information Systems, Wellington, 1-3 December.
- Carey, J. & Carlson, B. 2001, 'Business Components', in Component Based Software Engineering, Addison-Wesley, Reading, MA.
- Chen, P. P.-S. 1976, 'The Entity-Relationship Model: Toward a Unified View of Data', ACM Transactions on Database Systems, vol.1, no.1, pp. 9-36.
- Cheesman, J. & Daniels, J. 2001, UML Components: simple process for specifying component-based software, Addison Wesley, New Jersey.
- Chikofsky, E.J. & Cross, J.H. 1990, 'Reverse Engineering and Design Recovery - a Taxonomy', IEEE Software. vol.1, no.7. pp. 13-17.
- Chisholm, R.M. 1996, A Realistic Theory of Categories: An Essay on Ontology, Cambridge University Press, UK.
- Cormella-Dorda, S., Wallnau, K., Seacord R.C. & Robert J., 2000, A survey of legacy system modernization approaches [Online], Available: <http://www.sei.cmu.edu/publications/documents/00.reports/00tn003.html>, [Accessed 03 Dec 2007].
- De Lucia, A., Di Lucca, G. A., Fasolino, A. R., Guerra, P., & Petruzzelli, S. 1997, Migrating legacy systems towards object-oriented platforms, Proceedings of the International Conference on Software Maintenance (ICSM'97), Bari, 1-3 October.
- Davies, L., Green, P. & Rosemann M. 2002, Facilitating an ontological foundation of information systems with meta models, Proceedings of ACIS International conference of computer science, software engineering, information technology, e-business, and applications (ACIS 2002), St-Louis, 24-26 October.
- Desmet L., Jaco L., Mertens K. & Verhanneman T. 2003, COTS, the safety nightmare of component-oriented frameworks [Online], Available <http://www.cs.kuleuven.be/publicaties/rapporten/cw/CW367.pdf> , [Accessed 09 Sept 2008].
- Deursen, A.V. , Elsinga B., Klint P. & Tolido R. 2000, From legacy to component software renovation in three steps [Online], Available <http://home.hetnet.nl/~daan.rijzenbrij/cwicap/index.html>, [Accessed 08 Sept 2007]
- Deursen, A.V. & Kuipers, T. 1999, Identifying objects using cluster and concept analysis, Proceedings of the 21st International Conference of Software Engineering, ICSE-99 pp. 246-255.

Deursen, A.V. & Kuipers, T. 1998, Rapid system understanding: Two COBOL case studies. Proceedings of the sixth International Workshop on Program Comprehension; IWPC'98, IEEE Computer Society, pp. 90–98.

Deursen, A.V & Moonen, L. 1999, Understanding COBOL systems using types, Proceedings of the 7th Int. Workshop on Program Comprehension, IWPC'99, IEEE Computer Society, pp.74–83.

Degree, T. 1988, 'Business Object Component Architecture', IEEE Software, vol.15, no.5, pp.60-69.

D'Souza, D.F. & Wills, A. 1999, Objects, Components and Frameworks with UML: The Catalysis approach, Addison-Wesley, Reading, MA.

Dussart A., Conseil R., Aubert B. & Patry M. 2004, 'An Evaluation of Inter-Organizational Workflow Modeling Formalisms', Journal of Database Management, vol.15, no.2, pp. 74-104.

Erlikh L. 2002, 'Leveraging legacy system dollars for e-business', IT Professional, vol.2, no.6, pp. 54 – 56.

Evermann, J., Wand, Y. 2001, 'An Ontological Examination of Object Interaction in Conceptual Modeling'. Proceedings of the 11th Workshop on Information Technologies and Systems, (WITS 2001). New Orleans, Louisiana.

Favre, J.M , Duclos, F., Estublier, J., Sanlaville, R. & Auffret, J.J. 2001, Reverse Engineering a Large Component-based Software Product, Proceedings of the Fifth European Conference on Software Maintenance and Re-engineering (CSMR), Lisbon, Portugal, 14-16 March.

Fettke, P., Loos, P. 2003, 'Ontological evaluation of reference models using the Bunge Wand-Weber-model'. Proceedings of the Ninth Americas Conference on Information Systems. Tampa, FL, USA, pp. 2944-2955.

Fettke, P., Loos, P.2004 'Ontological evaluation of the specification framework proposed by The "Standardized Specification of Business Components" memorandum –some preliminary results. Proceedings of the 1st International Workshop on Component Engineering Methodology, Erfurt, Germany, 11-12 September.

Flynn, D. 1998, Information Systems Requirements: determination and analysis, McGraw-Hill, New York.

Fontannete, V., Garcia, V.C., Perez, A., Prado, A.F. 2002, 'Component-Oriented Software Re-engineering using Transformations'. Proceedings of the International Conference on Computer Science, Software Engineering, Information Technology, E-business, and applications, Foz do Iguacu - Paraná. Proceedings of the ACIS International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications. A Publication of the Internacional Association for Computer and Information Science, p. 206-211.

- Gilbert, A. L. 1993, A contingent theory of information technology planning: Rationality, learning, or negotiations?, PhD thesis, Northeastern University.
- Green P. & Rosemann M. 1999, An Ontological Analysis of Integrated Process Modelling, Proceedings of the 11th International Conference on Advanced Information Systems Engineering (CAiSE 99), Springer: Berlin, pp. 225-240.
- Green, P. & Rosemann, M. 2000, 'Integrated Process Modelling: an ontological evaluation', Information Systems, vol.25, no.2, pp. 73-87.
- Green, P. & Rosemann, M. 2005, Business Analysis with Ontologies, Idea Group Publishing, New York. US.
- Haddox, J.M., Kapfhammer, G.M., 2002, 'An approach for understanding and testing third party software components'. Proceedings of 2002 Annual Reliability and Maintainability Symposium, Seattle, WA, January 28-31.
- Heineman G. T. & Councill W.T, 2001, 'Definition of a Software Component and its elements' in Component Based Software Engineering , Addison-Wesley, Boston, Mass.
- Henderson-Sellers B. & Barbier F 1999, 'Black and White Diamonds', Proceedings of the UML 99-The Unified Modeling Language (Beyond the Standard), UML 99 R.France and B. Rumpe, eds., Lecture Notes in Computer Science (LNCS), No. 1723, Springer Verlag, Berlin, Germany, 1999, pp. 550-565.
- Henderson-Sellers B. 2001, 'An Open Process for Component-Based Development', in Component Based Software Engineering , Addison-Wesley, pp. 243-263.
- Hevner, A.R., March, S.T., and Park, J. 2004, 'Design Research in Information Systems Research', MIS Quarterly, vol.28, no.1, pp 75-105.
- Hoffman, C. 1997, 'Requirements model Tutorial' [Online], available: <http://www.balsamfir.com/documents/overview.html>, [Accessed 15 January 2003].
- Houston, K. & Norris, D. 2001, 'Software Component and the UML' in Component Based Software Engineering , Addison-Wesley, pp. 243-263.
- Hyperion Solutions Corporation 2001 Requirements Modelling: Ready for Prime Time Report [Online], Available <http://whitepapers.silicon.com/0,39024759,60043623p,00.htm>, [Accessed 1 April 2001].
- Irwin, G. & Turk, D. 2005, 'An Ontological Analysis of Use Case Modeling Grammar', Journal of the Association for Information Systems, vol.6., no.1., pp.1-36

- Kaplan, B., & Dorsey, P. 1991, Requirements Analysis interviewing: alternative perspectives. Technical Report 91-021. Washington, D.C.: The American University, Department of Computer Science and Information Systems.
- Jacobson, I. 1987, 'Object Oriented Development in an Industrial Environment.'. Proceedings of the OOPSLA. Orlando, Florida.: ACM Press, pp. 183-191.
- Jacobson, I. & Lindstrom F. 1991, 'Re-engineering of Old Systems to an Object-Oriented Approach,' Proceedings of the In Conference on Object-Oriented Programming Systems, Languages and Applications OOPSLA 1991, pp. 340-350.
- Jacobson, I., Christerson, M., Jonsson, P. & Overgaard, G. 1993, Object-oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, Wokingham, England.
- Jorgensen, D. L. 1989, Participant observation: Methodology for human studies, Sage publications, Newbury, CA.
- Jones, C. 1991, Applied Software Measurement: Assuring Productivity and Quality. McGraw-Hill. US.
- Kerlinger, F.N. 1986, Foundations of behavioral research, Holt Rinehart and Winston Publisher, New York.
- Kinnear, T. & Taylor, J. 1996, Marketing Research, McGraw-Hill, New York.
- Koschke, R., 2000, Atomic Architectural Component Recovery for Understanding and Evolution, Phd. Thesis, University of Stuttgart.
- Hawthorne, M. & Perry D.E. 2005, 'Software Engineering education in the era of outsourcing, distributed development, and open source software: challenges and opportunities', Proceedings of the 27th international conference on Software engineering, St. Louis, pp.643-644.
- Hammersley, M., & Atkinson, P. 1989, Ethnography: Principles in practice, Tavistock publisher, London.
- Lakhotia, A. 1997, 'A unified framework for expressing software subsystem classification techniques', Journal of Systems and Software, vol.36, no.3. pp. 211–231.
- Longworth, R. 2003, Modeling Events for Today's System Requirements, [Online] available from Internet <http://www.cips.ca/it/resources/eventviews.pdf> [Accessed 20 Oct 2003].
- Lunn, K. 1997, OO Analysis and Design - Course Notes , [Online], Available from URL: <http://www.csc.liv.ac.uk/~1cs88/oonotes-relevant.htm> [Accessed 20 Oct 2003].

- Mylopoulos, J. 1998, 'Information modelling in the time of the revolution', *Information Systems*, vol. 23, pp. 127-155.
- Myles, M.B. & Huberman, A.M. 1994, *Qualitative data analysis: An expanded sourcebook*, Sage publications, Thousand Oaks, CA.
- Neuman, W.L. 1994, 'Sampling', in *Social Research Methods*, Allyn and Bacon, Boston.
- Nunamaker, J.F. & Chen, M. 1990, 'Systems development in information systems research', *Proceedings of the Twenty-Third Annual Hawaii International Conference on Systems Science*, IEEE, pp 631-639.
- OMG, 2003, UML 2.0 Superstructure Final Adopted Specification [Online], Available: <http://www.omg.org/cgi-bin/doc?ptc/2003-08-02> [Accessed 08 Aug 2006]
- Opdahl, A.L. & Henderson-Sellers, B. 2002a, 'Understanding and improving the UML metamodel through ontological analysis', *Journal of Software and Systems Modelling*, vol.1, no.1, pp. 43-67.
- Opdahl, A.L., & Henderson-Sellers, B. 2002b, 'Ontological Evaluation of the UML Using the Bunge-Wand-Weber Model', *Software Systems Model*, vol.1, no.1, pp. 43-67.
- Opdahl, A.L. & Henderson-Sellers, B. 2004, 'A template for defining enterprise modeling Constructs', *Journal of Database Management*, vol.15, no.2, pp. 39-73.
- Orlikowski, W. & Baroudi, J. 1991, 'Studying Information Technology in Organizations: Research Approaches and Assumptions', *Information Systems Research*, vol.2, no.1, pp. 1-28.
- March, S. & Smith, G. 1995, 'Design and Natural Science Research on Information Technology', *Decision Support Systems*, vol.15, no.4, pp. 251 - 266.
- Mišić, V.B. & Zhao, J. L. 2000, 'Evaluating the Quality of Reference Models', *Proceedings of ER 2000 – 19th Conference on Conceptual Modeling*, Berlin, Springer-Verlag, pp. 484-498.
- Peräkylä, A. 1997, 'Reliability and validity in research based on tapes and transcripts', in *Qualitative research: theory, method and practice*, eds D. Silverman (Ed.), Sage publishing, London.
- Perry, C. 1998, 'Processes of a case study methodology for postgraduate research in marketing', *European Journal of Marketing*, vol.32, no.9, pp. 785-802.
- Prado, F. 1992, *Estratégia de Engenharia de Software Orientada a Domínios*, Rio de Janeiro/RJ, 333, PhD thesis, Pontificia Universidade Católica.

- Reed, Jr, P.R. 2002, *Developing Applications with JAVA and UML*, Addison-Wesley, Boston.
- Rosemann, M. & Green, P. 2002, 'Developing a meta model for the Bunge-Wand-Weber Ontological Constructs', *Information Systems*, vol.27, no.2, pp. 75-91.
- Rosemann, M., Recker J., Indulska M. & Green P. 2005, *Process Modeling – A Maturing Discipline? BPMI.org: Business Process Modeling Notation (BPMN)*, available at: <http://www.bpmi.org/> accessed June 5 2005.
- Rosemann, M. & Green, P., 1999 'Enhancing the Process of Ontological Analysis "The "Who cares" Dimension', *Proceedings of the Information Systems Foundations Workshop Ontology, Semiotics and Practice*.
- Rosemann, M. & Zur-Muehlen, M. 1998, 'Evaluation of Workflow Management Systems – a Meta Model Approach', *The Australian Journal of Information Systems*, vol. 6, no.1, pp. 103-116.
- Rumbaugh, J. 1994, 'Getting started: Using use cases to capture requirements', *Journal of Object-Oriented Programming*, vol.7, no.5, pp. 8-12.
- Satzinger, J.W., Jackson, R.B., & Burd, S.D. 2002, *Systems Analysis and Design in a Changing World*, Course Technology, Boston, Mass.
- Seacord, R.C., Plakosh, D. & Lewis, G.A 2003, *Modernizing legacy systems*, Addison Wesley Boston.
- Serrano, M., Carver, D. & De-Oca, C. 2001, 'Re-engineering legacy systems for distributed environments', *Journal of Systems and Software*, vol.64, no.1, pp. 37-55.
- Sommerville, I. 2001, *Software Engineering*, Addison-Wesley, Boston.
- Scheer, A.W. 1998, *ARIS–Business Process Frameworks*, Springer-Verlag, Berlin.
- Soy, S. 1996, *The Case Study as a Research Method* [Online] available from Internet: <http://www.gslis.utexas.edu/~ssoy/usesusers/l391d1b.htm> [Accessed in 15 March 2003].
- Sparx Systems 2001, *An introduction to Business Process Modeling in Enterprise Architect, with BPMN and Eriksson-Penker* [Online], Available <http://www.sparxsystems.com.au/downloads/whitepapers/businessProcessModelTutorial.pdf>, [Accessed 2 Feb 2002].
- Spradley, J.P. 1979, *The ethnographic interview*, Holt, Rinehart and Winston, New York.
- Stake, R.E. 1995, *The art of case study research*, Sage publishing, California, CA.
- Stroulia, E., El-Ramly, M., Kong L., Sorenson, P., & Matichuck, B. 1999, 'Reverse engineering legacy interfaces: An interaction-driven approach', *Proceedings of the*

6th Working Conference on Reverse Engineering, WCRE'99, pages 292–301. IEEE Computer Society, Atlanta.

Strauss, A., & Corbin, J. 1990, *Basics of qualitative research: Grounded theory procedures and techniques*, Sage publishing, Newbury Park, CA.

Szyperski, C. 1998, *Component Software: Beyond Object-Oriented Programming*, Addison Wesley, Reading MA.

Tagg, R. 1991, 'A Review of the Suitability of Various Applications for an OO Approach', *Proceedings of the Tenth Annual Conference TAC 1991*, British Computer Society, Walsall.

Tull, D. and Hawkins, D. 1993, *Marketing Research: Measurement and Method*, Macmillan Publishing Company, New York, NY.

Uschold, M., King, M., Moralee, S. & Zorgios, Y. 1998, 'The enterprise ontology'. *The Knowledge Engineering Review*, vol.13, no.1, pp. 31-89.

Valverde, R. & Toleman, M. 2007, 'Ontological evaluation of requirements models: comparing traditional and component-based paradigms in information systems re-engineering' in *Ontologies: a handbook of principles, concepts and applications in information systems*, eds R. Sharman & R. Kinshore, Springer, New York.

Verrijn-Stuart, A.A. 2001, 'A Framework of Information System Concepts', *Proceedings of the IFIP TC8/WG8.1 Working Conference on Information System Concepts*, Brussels, November 15-16.

Vessey, I., Ramesh V. & Glass, R. 2004, 'An Analysis for Research in the Computing Disciplines', *Communications of the ACM*, vol.47, no.6, pp. 89-94.

Warrel, D. & Stevens K 2003, 'Towards and Understanding of Risk Management in Legacy Information Systems Modernisations', *Proceedings of the 14th Australasian Conference on Information Systems ACIS 2003* November, Perth Western Australia.

Wand, Y. & Weber, R. 1988, 'An ontological analysis of some fundamental information systems concepts', *Proceedings of the Ninth International Conference on Information Systems*, Minneapolis USA, November 30–December 3.

Wand, Y. & Weber, R. 1989, 'An ontological evaluation of systems analysis and design Methods', *Proceedings of the IFIP WG8.1 Working Conference on Information Systems Concepts: An In-Depth Analysis*, Namur, Belgium, pp.79–107, North-Holland, Amsterdam.

Wand, Y. & Weber, R. , 1990, 'An ontological model of an information system', *IEEE Transactions on Software Engineering (TSE)*, vol.16, no.11, pp. 1282–1292.

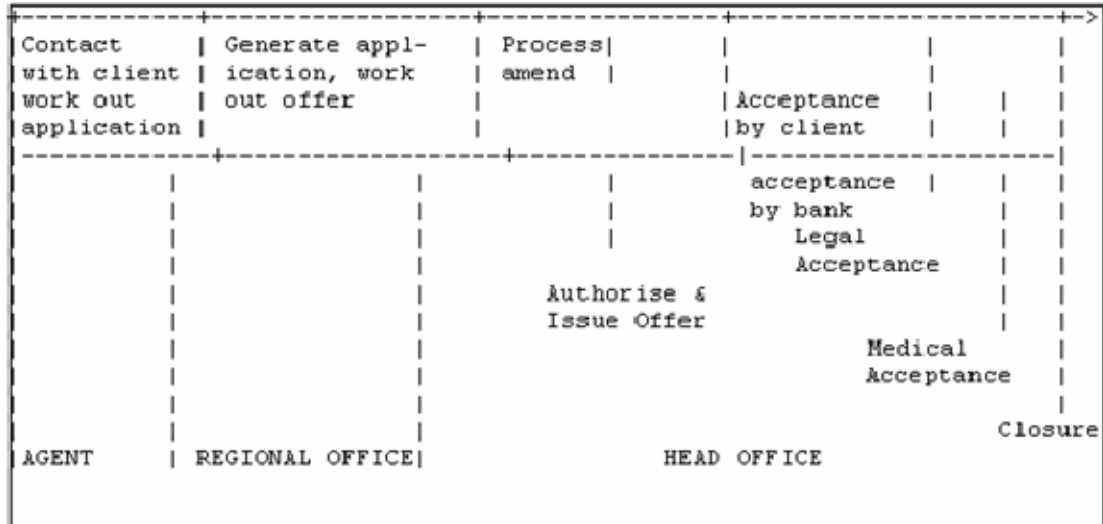
- Wand, Y. & Weber, R. 1993, 'On the ontological expressiveness of information systems analysis and design grammars', *Journal of Information Systems*, vol.3, no.4, pp. 217–237.
- Wand, Y. & Weber, R. 1995, 'On the deep structure of information systems', *Information Systems Journal*, vol.5, no.3, pp. 203–223.
- Wand, Y. & Weber, R. 2002, 'Research Commentary: Information Systems and Conceptual Modelling - A Research Agenda', *Information Systems Research*, vol.13, no.4, pp. 363-377.
- Weber, R. 1997, *Ontological Foundations of Information Systems*, Buscombe Vicprint, Blackburn, Victoria.
- Weber, R. & Zhang, Y. 1996, 'An analytical evaluation of NIAM's grammar for conceptual schema diagrams', *Information Systems Journal*, vol.6, no.2, pp. 147-170.
- Wolfgang, P. 1997, 'Component-Based Software Development - A New Paradigm in Software Engineering?', *Software-Concepts and Tools Software - Concepts and Tools*, vol.18, no.4, pp.169-174.
- Wolcott, H. 1990, *Writing up qualitative research*, Sage publishing, Newbury Park, CA.
- Whitten, J. L., Bentley D. L. & Dittman K.V. 2000, *Systems Analysis and Design Methods*, McGraw-Hill, New York
- Wreden, N. 1998, *Model Business Processes* [Online], Available: <http://www.informationweek.com/702/02iuprc.htm>, [Accessed 20 February 2007].
- Wu, B., Lawless, D., Bisbal, J., Grimson, J., Wade, V., O'Sullivan, D. & Richardson, R. 1997, 'Legacy Systems Migration - A method and its Tool-kit Framework', *Proceedings of the APSEC '97/ICSC '97*, Ed. IEEE Computer Society.
- Yin, R.K. 1994, *Case Study Research-Design and Methods*, Applied Social Research Methods Series, Sage publishing, Newbury Park.
- Yun-Tung, L. 2001, *The art of objects: Object Oriented design and Architecture*, Addison Wesley, Reading, MA.
- Zikmund, W.G. 2000, *Business Research Methods*, The Dryden Press, Forth Worth.
- Zou R. & Kontogiannis K. 2002, *Migration to Object Oriented Platforms*, Revised edition, Sage Publications, Newbury Park.



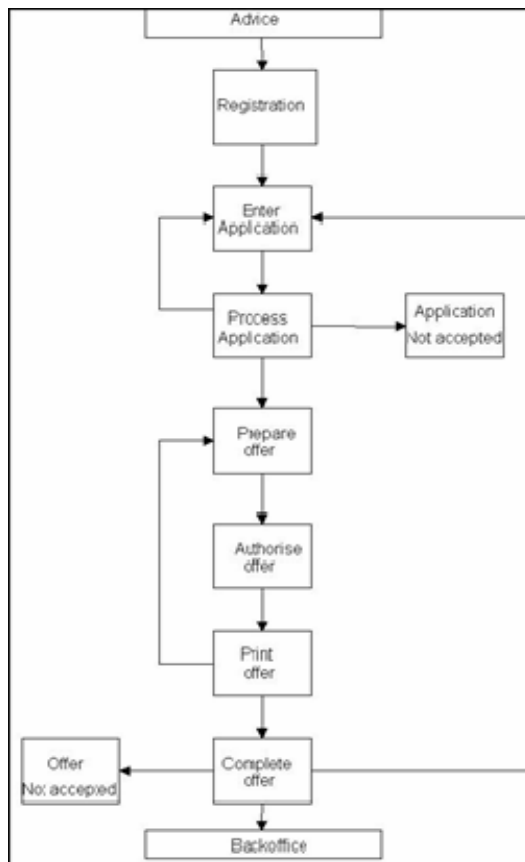
## APPENDICES

### Appendix A System Architecture documentation collected

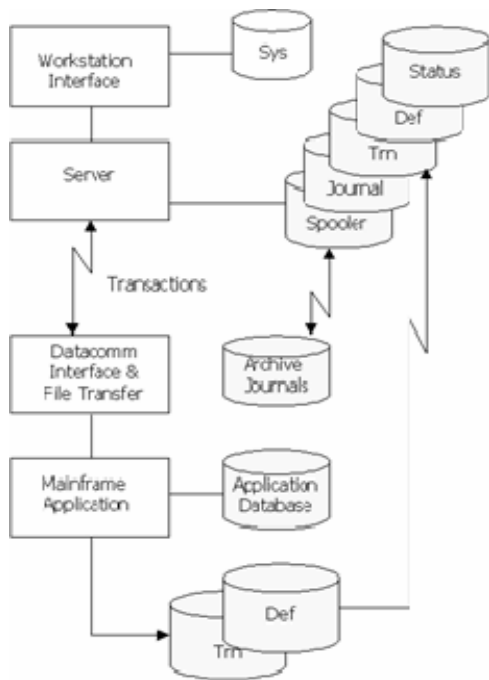
#### Appendix A-1 Sub-system flow (geographic)



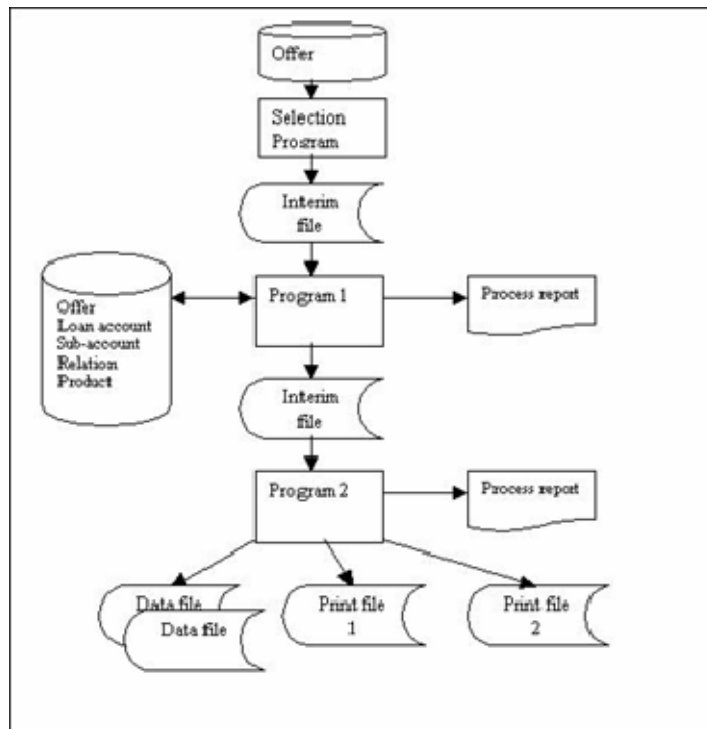
#### Appendix A-2 Sub-system process flow



### Appendix A-3 System architecture for the procedural model



### Appendix A-4 Typical batch process program flow



### Appendix A-5 Offer Screen

New Change Inquire Relation Change-status Calculate Print

### Appendix A-6 New Offer entry screen 1

Application number : _____
Okay Cancel

### Appendix A-7 New Offer Entry Screen Product 1

Product selected by entering product code or using F1 to select product from a table listbox.
Okay Cancel

### Appendix A-8 New Offer Entry Screen Product 2

Product details extended to include custom details such as repayment options & fixed interest rate duration.
Okay Cancel

### Appendix A-9 Calculate Offer Entry Screen

Offer number : _____
Okay Cancel

### Appendix A-10 Change status screen

Offer number : _____
New Status number : ____
Confirmation or denial (with reason) message displayed
Okay Cancel

## Appendix B Use Cases

### Appendix B-1 Use case for process application

Use Case ID:	1
Use Case Name:	Process Application
Actors:	Sales agent, Applicant.
Description:	This use-case satisfies all of the goals of setting up and processing a new application. This applies for both existing as well as new applicants. All aspects of the application process are covered, from initial entry to application acceptance.
Trigger(s):	All events dealing with new and existing applicants applying for a home loan through the usual sales channels.
Preconditions:	The sales agent must be logged onto the system.
Postconditions:	Application attains a final status of accepted by all parties. An application printout is made for record purposes.
Normal Flow:	<p>An applicant completes an application on-line, supplies all required information and the application is accepted by all parties.</p> <ol style="list-style-type: none"> <li>1. All relevant Applicant and sales details are entered for new relations including: <ul style="list-style-type: none"> <li>Debtor</li> <li>Agent (if applicable)</li> <li>Account manager</li> <li>Applicant name, address and bank account.</li> </ul> </li> <li>2. Entry of property details (to be used as security against the loan). <ul style="list-style-type: none"> <li>Purchase price</li> <li>taxation value (if known at this stage)</li> <li>location</li> <li>property type</li> </ul> </li> </ol>

	<ol style="list-style-type: none"> <li>3. Entry of Loan requirement details (credit application) including:  details of existing loans  details of assets and liabilities  monthly income and expenditure  Loan amount required  Deposit amount (if applicable)  Desired payment frequency</li> <li>4. Agent selects income test before proceeding to next step:  Income test, test which controls whether or not income is sufficient for required loan</li> <li>5. Product(s) is (are) selected:  Applicant selects desired product or product combination  Selected Product(s) entered and verified by the system</li> <li>6. Sales Agent invokes credit testing and income testing modules.</li> <li>7. Application document is requested and generated.</li> <li>8. Upon signature of the Application document by the Applicant, the Sales agent requests and receives Application Acceptance status from the system.</li> </ol>
Alternative Flows:	<p>An applicant completes an application via a physical application form, supplies all required information and the application is accepted by all parties.</p> <ol style="list-style-type: none"> <li>1. The sales agent prints out an application form, which the applicant completes and the details on the form are then entered online.</li> </ol>
Exceptions:	<p>The bank due to a failed credit check rejects the application.  The application is rejected by the bank due to failing to meet income/expenditure balance requirements.</p>
Includes:	
Priority:	1
Frequency of Use:	100/h
Business Rules:	1. Agent must be registered with bank.

	<ul style="list-style-type: none"> <li>2. Property must be within national borders.</li> <li>3. Loan amount may not be greater than 125% of taxation value of property.</li> <li>4. Standard requirement of 10% deposit against capital sum.</li> <li>5. Applicant must pass the income test for loan sum required.</li> </ul>
Special Requirements:	
Assumptions:	
Notes and Issues:	Location of source is in Regional Office

## Appendix B-2 Use Case for Loan Generation

Use Case ID:	2
Use Case Name:	Loan generation.
Actors:	System and Administrator
Description:	This use-case satisfies the goals of creating a loan structure based on the data provided in the accepted offer as well as the creation of all relevant database couplings.
Trigger(s):	The offer acquiring the status of accepted by head office and the applicant.
Preconditions:	The administrator must be logged onto the system.
Postconditions:	Offer obtains a. status of closed and the loan structure and all relevant couplings are made in the database.
Normal Flow:	<p>Once the offer is accepted by all parties, an administrator upgrades the offer status to accepted whereby the loan structure of the new customer is created, based on the accepted offer.</p> <ol style="list-style-type: none"> <li>1. A loan structure is created in the system. The following is created in the system (based on the product model and the data provided in the offer): All relevant accounts and sub-accounts are created</li> <li>2. Couplings between relation entities are created such as: Debtor to debit account Contract to loan to notary bond Agenda profile is created in terms of: Payment schedule obligations, Interest rate review periods, Archiving of records</li> <li>3. The generation of data files occurs, for external interfaces to agents, notaries, insurance companies, the credit</li> </ol>



	bureau and the tax department
Alternative Flows:	
Exceptions:	The system rejects the creation of the loan structure due to a business rule.
Includes:	
Priority:	1
Frequency of Use:	100/day
Business Rules:	<ol style="list-style-type: none"> <li>1. The generation of the contract data can only occur once offer is at status accepted by head office.</li> <li>2. Debit account must be generated within the loan structure in order to handle internal payments as well as external payments.</li> </ol>
Special Requirements:	
Assumptions:	
Notes and Issues:	Regional Offices and Agencies

**Appendix B-3 Use case for Process Offer by regional office (online process)**

Use Case ID:	3
Use Case Name:	Process Offer by regional office (online process)
Actors:	Sales agent, applicant, Applicant, Administrator, Debtor, and Insurer
Description:	This use-case satisfies all of the goals of setting up and processing a new offer to the status of production by regional office. This applies for both existing as well as new applicants. All aspects of the offer process are covered, from initial registration to the production of the offer at the regional office.
Trigger(s):	All events dealing with new and existing applicants applying for a home loan through the usual sales channels. Once the application is completed, the applicant requests an offer from the Sales Agent. An on-line offer form is loaded.
Preconditions:	The sales agent must be logged onto the system.
Postconditions:	Offer attains a status of produced at regional office.
Normal Flow:	Sales agent processes an offer and generates a yet to be authorized offer document all in a single session: <ol style="list-style-type: none"> <li>1. Relevant application data is made available in workstation offer entry screen once sales agent enters application number in the offer entry screen.</li> <li>2. Changes (if applicable) to relation and/or loan and/or property details are made.</li> <li>3. A product or product combination is captured. All entry requirements within the product or product combination structure are filled these include: the completion of all sub-product requirements and options such as linking savings, accounts and linking depot accounts, Fixed interest rate duration and Distribution of capital loan amount over product structure.</li> <li>4. Loan repayment options are captured including: external bank account number, method and frequency of payments and amount of deposit to be paid.</li> <li>5. Sales agent attempts to upgrade offer status to Registered. Once all local and mainframe based controls have</li> </ol>

	<p>been processed, the offer is upgraded by the system to status registered.</p> <p>6. The computational module is called from the workstation by the sales agent in order to calculate: Loan - interest payments, percentages and amount based on duration of fixed interest, reduction on capital amount over time penalties for early loan settlement or non-scheduled payments against capital loan.</p> <p>7. On completion of the computational module, the offer is placed by the system as status ready for printing.</p> <p>8. The sales agent selects print from the offer entry screen. The offer document is printed.</p> <p>9. Once the applicant accepts the offer, the offer attains status of offer complete for transfer to head office.</p>
Alternative Flows:	<p>Sales agent processes an offer in a number of sessions with the applicant.. A range of offers is made under a single application.</p> <p>1 – 8Completed over time and from last registered offer status a range of offers is made under a single application.</p> <p>1 – 8Repeated for every required offer variation. Each new offer receives an incremented series number.</p>
Exceptions:	The system rejects the creation of the loan structure due to a business rule.
Includes:	
Priority:	1
Frequency of Use:	10/h
Business Rules:	<ol style="list-style-type: none"> <li>1. A maximum of four sub-accounts are allowed within a contract.</li> <li>2. External account details (the payer account) must be linked to a bank registered with the home loan bank.</li> <li>3. An internal saving account must be linked to the sub-account representing the product.</li> </ol>
Special Requirements:	
Assumptions:	
Notes and Issues:	Regional Offices and Agencies

**Appendix B-4 Use case for Process Offer by Head office (online process)**

Use Case ID:	4
Use Case Name:	Process Offer by head office
Actors:	Administrator, Applicant, Bank, Notary, Credit Bureau and Tax department
Description:	This use-case satisfies all of the goals of processing a new offer from the status of registered at head office to the status of accepted or declined by head office.
Trigger(s):	1. All events dealing with the gathering and processing of required information in order for the offer to be able to be assessed. 2. The offer obtains the status transferred to head office. The administrator requests (via an on-line menu option) a daily list of all offers that have attained the status of transferred to head office.
Preconditions:	The administrator must be logged onto the system.
Postconditions:	Offer attains a status of either accepted or declined by head office.
Normal Flow:	<p>The administrator processes an offer to the status of accepted by head office.</p> <p>The administrator requests (via an on-line menu option) a printed list of all offers that have reached the status of transferred to head office.</p> <p>Physical files for the holding of all required documents are created for all offers on the list.</p> <p>The administrator upgrades the status of the offer to registered at head office.</p> <p>Upon the offer obtaining the status of registered at head office, the system generates letters to the following external entities:</p> <p>Notary, requesting authorization for the creation of the home loan bond.</p> <p>Credit Bureau for credit approval of the debtor.</p> <p>The system also generates a flat file to be read in by the banks legal department sub-system.</p> <p>An assessment period is entered into wherein the administrator gathers all incoming documentation.</p>

	Once all required documentation is received and all external entities approve of the offer, the administrator upgrades the offer to accepted by head office.
Alternative Flows:	None
Exceptions:	The offer is declined by head office.
Includes:	
Priority:	1
Frequency of Use:	100/day
Business Rules:	1. The offer can only be accepted by head office upon: Receipt of Medical clearance for the debtor Receipt of Credit Bureau clearance Receipt of approval from relevant insurance companies Receipt of approval from the banks legal department Receipt of notary authorization
Special Requirements:	
Assumptions:	
Notes and Issues:	Place of process at Head Office

**Appendix B-5 Use case for Maintain offer (online process)**

Use Case ID:	5
Use Case Name:	Maintain Offer

Actors:	Administrator, Applicant
Description:	This use-case satisfies all of the goals of maintaining a new offer from the status of accepted by applicant to accepted or declined by head office.
Trigger(s):	1. All events dealing with the gathering and processing of required information in order for the offer to be able to be assessed. 2. The applicant calls the service centre to inquire about the status of an offer.
Preconditions:	The administrator must be logged onto the system.
Postconditions:	Offer obtains a status relevant to the change administered
Normal Flow:	An applicant calls to inquire about the status of an offer.  Upon the telephonic request from the applicant, the administrator invokes the offer inquiry screen. The administrator enters the applicable offer number and reads off status and/or other relevant offer details. The administrator terminates the screen upon termination of the applicants call.
Alternative Flows:	An applicant calls to change the details of an offer. Deposit amount or fixed interest changes, for example.  1 – 2 Administrator determines from screen information and/or offer status provided whether or not the desired change is allowed. 3. If deemed allowable, the administrator will invoke the change offer details screen and make the change. 4. Any change to the offer automatically results (by the system) in the status of the offer being reset to registered, effectively resulting in a new offer.

	<p>5. The administrator will change the status of the offer to registration authorised by regional office.</p> <p>6. The administrator will reprint the offer and send the document out for posting to the applicant.</p> <p>An applicant calls to cancel the offer/s.</p> <p>1 – 2 same as normal flow.</p> <p>3. The administrator will invoke the change offer details screen and make the cancellation.</p> <p>4. The system automatically changes the status of the offer to cancelled by applicant.</p> <p>5. Upon cancellation, the system will trigger the printing of a cancellation confirmation letter.</p> <p>6. The administrator will send the cancellation letter out for posting.</p>
Exceptions:	
Includes:	
Priority:	2
Frequency of Use:	100/hour
Business Rules:	<p>1. Any change to the capital amount to be borrowed must result in a credit check.</p> <p>2. No changes are allowed if the offer has reached the status of closed or definite.</p>
Special Requirements:	
Assumptions:	
Notes and Issues:	Place of process at Head Office Service Centre

## Appendix C Event-Response Tables

### Appendix C-1 Possible events for the Process Offer by Regional Office

Actor	Event	Trigger	Response
Sales Agent	Sales agent enters application number in the offer entry screen	The applicant requests an offer from the Sales Agent	Relevant application data is made available in workstation offer entry screen.
Sales Agent	Changes to relation and/or loan and/or property details are entered on the screen	Application data is on the screen and applicant requests changes on the application	Changes to relation and/or loan and/or property details are made in the application table
Sales Agent	A product or product combination is captured and all entry requirements within the product or product combination structure are Filled These include: 1.the completion of all sub-product requirements and options such as linking savings accounts and linking depot accounts. 2.Fixed interest rate	Application data is on the screen	The offer is created and saved in the system



	<p>duration.</p> <p>3. Distribution of capital loan amount over product structure.</p> <p>Loan repayment options are captured including:</p> <p>1. external bank account number, method and frequency of payments</p> <p>2. Amount of deposit to be paid.</p>		
Sales Agent	Sales agent attempts to upgrade offer status to Registered	Offer is created	Update the offer in the offer table with the status of “registered”
Sales Agent	The computational module is called from the workstation by the sales agent	Request of Computational module by Sales Agent + Offer has been created	The system calculates: Loan - interest payments (percentages and amount based on duration of fixed interest. Reduction on capital amount over time. Penalties for early loan settlement or non-scheduled payments against capital loan. The system displays a message “Ready” for printing”
Sales Agent	The sales agent selects print from the offer entry screen	The system displays a message “Ready”	The offer document is printed. Offer is sent to applicant

Applicant	Accepts or refuses offer	Offer is received by applicant	Informs sales agent about his decision
Sales Agent	Retrieves Offer from the system and updates status to “complete for transfer to head office”	Offer is accepted by Applicant	The system updates offer in the offer table with the status of “complete for transfer to head office”

**Appendix C-2 Possible events for the Process Offer by Head Office use case**

Actor	Event	Trigger	Response
Administrator	System enquiries to the database for offers with status of transferred to the office	Request for list of offers with status of transferred to office.	A screen output with all the offers that have reached status of transferred to head office is displayed  Administrator creates physical files for the holding of all required documents are created for all offers on the list.
Administrator	The administrator upgrades the status of the offers to registered at head office on the screen	Screen output with offers with “transferred to head office” status is displayed	System changes in the offers table the status of all the offers to “registered at the head office”  The system generates letters to the following external entities: Notary, requesting authorization for the creation of the home loan bond. Credit Bureau for credit approval of the debtor.  Letters are sent to the external entities by administrator.  The system also generates a flat file with the offers information to be read in by the banks legal department sub-system.
Administrator	Enter an assessment period for each offer wherein the administrator gathers all incoming documentation	Offers changed to status of “registered at the head office”	Update the offers table with the assessment period from the system
Administrator	upgrades the offer to accepted by head office	All required documentation is received and all	Update the offers table with the status of accepted by the office

		external entities approve of the offer	
--	--	---	--

**Appendix C-3 Possible events for the Maintain Offer use case**

Actor	Event	Trigger	Response
Administrator	the administrator invokes the offer inquiry screen and enters offer number and or offer relevant details	The applicant calls the service centre to inquire about the status of an offer	The system displays on the screen offer details
Administrator	The administrator will invoke the change offer details screen and make the cancellation	An applicant calls to cancel the offer/s. and the system has already displayed the offer details on the screen	The system automatically changes the status of the offer to cancelled by applicant in the offer table Upon cancellation, the system will trigger the printing of a cancellation confirmation letter. The administrator will send the cancellation letter out for posting
Administrator	The administrator will invoke the change offer details screen and make the change.	An applicant calls to change the requirements or details of an offer and the system has already displayed the offer details on the screen	The system will change the status of the offer being reset to be registered and a new offer will be created .
Administrator	The administrator will change the status of the offer to registration authorized by regional office.	Offer reset to status “to be registered” and customer called to change details in offer office	The system changed the registration status to registered for the offer

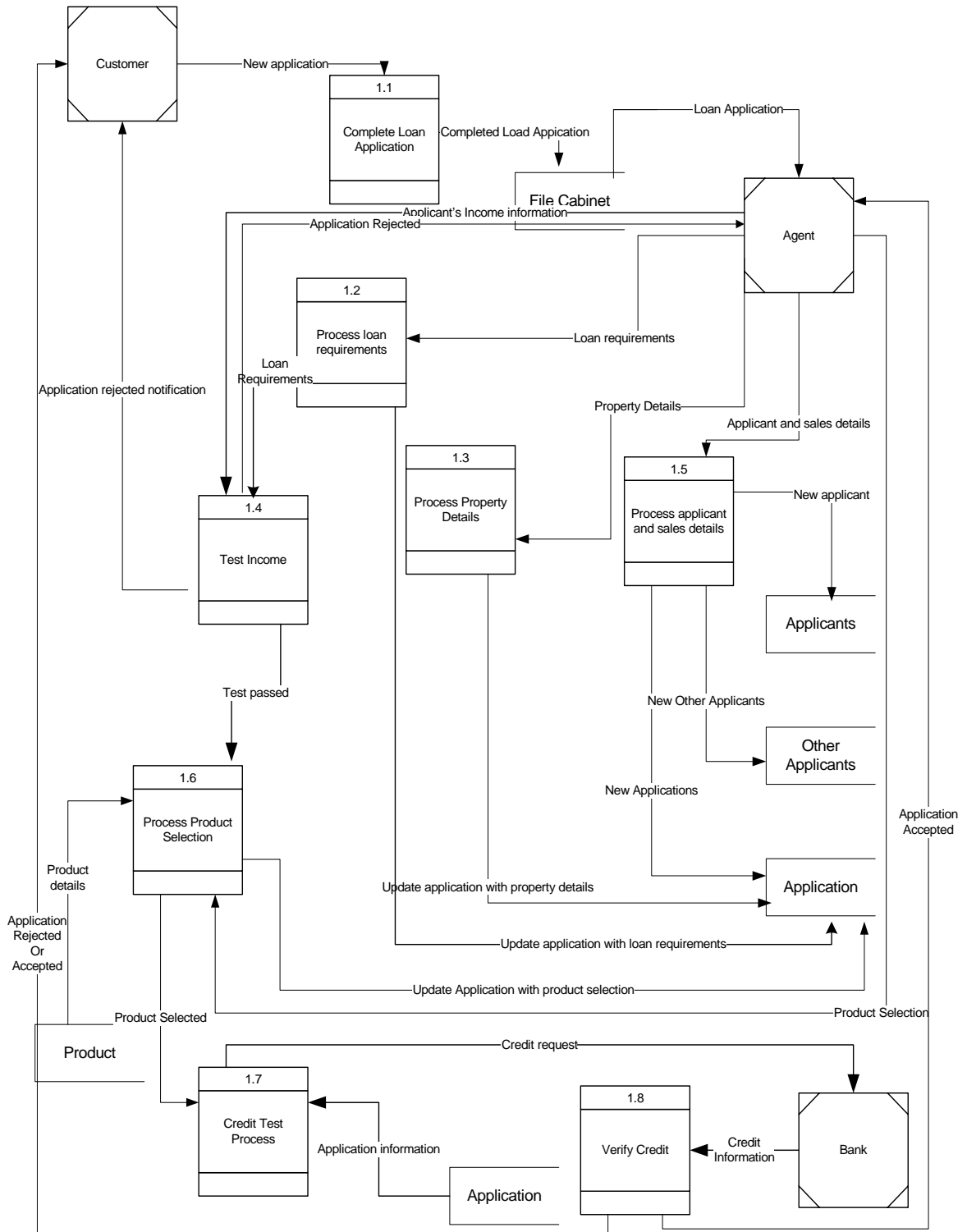
Administrator	Reprint the offer	System changed to registration status and Customer called for changes in offer	The system will print the offer in paper format. Administrator will send the offer
---------------	-------------------	--	---

**Appendix C-4 Possible events for the Loan Generation use case**

Actor	Event	Trigger	Response
Administrator	Create loan structure is in the system.	Offer acquiring the status of accepted by the head office	All relevant accounts and sub-accounts are created. Couplings between relation entities are created such as: Debtor to debit account Contract to loan to notary bond  Agenda profile is created in terms of: Payment schedule obligations, Interest rate review periods, Archiving of records
System	The generation of data files occurs	Loan structure created in the database	data files occurs are created for external interfaces to agents, notaries, insurance companies, the credit bureau and the tax department

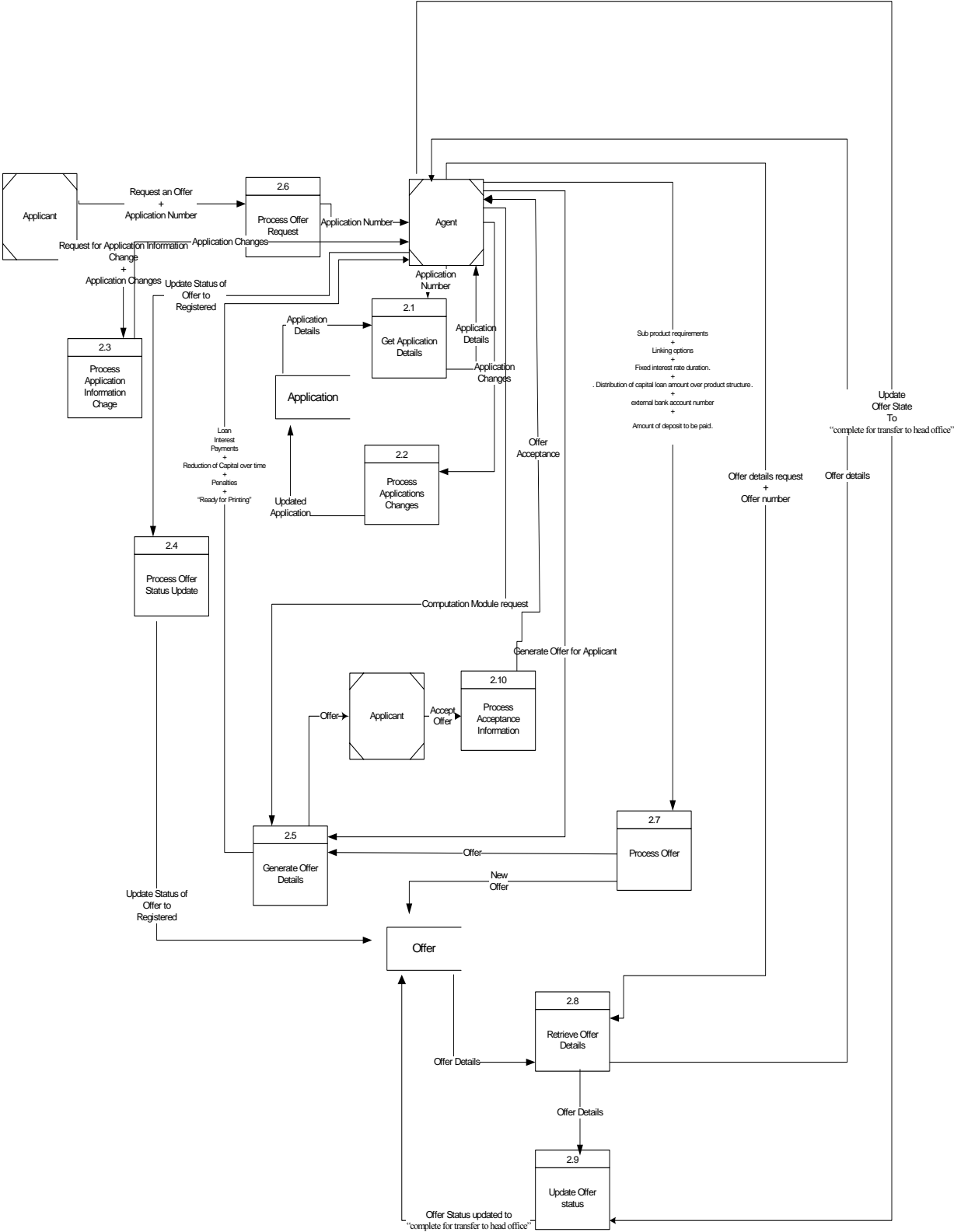
## Appendix D DFDs for legacy system of case study

### Appendix D-1 DFD for Process Application use case

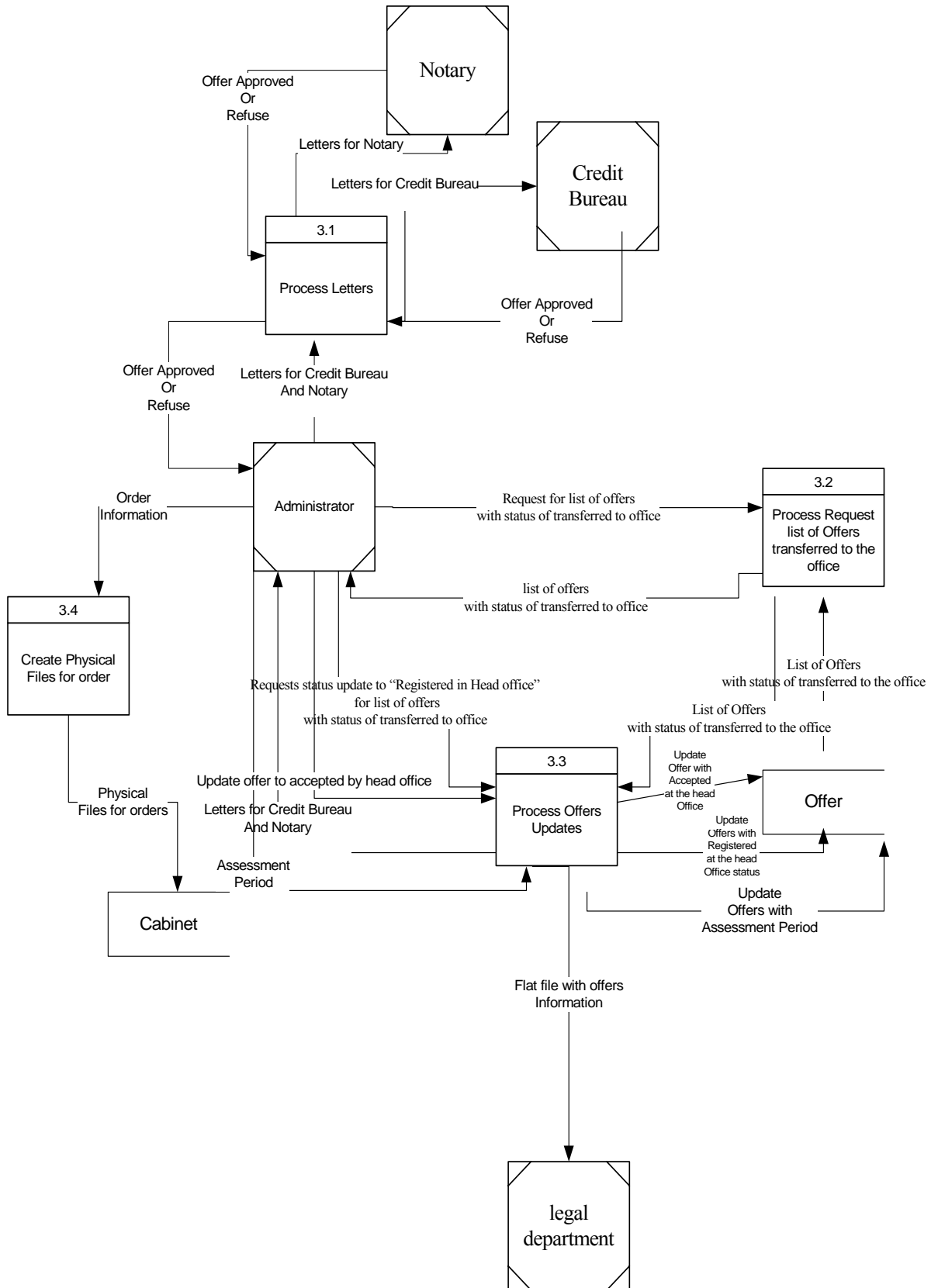




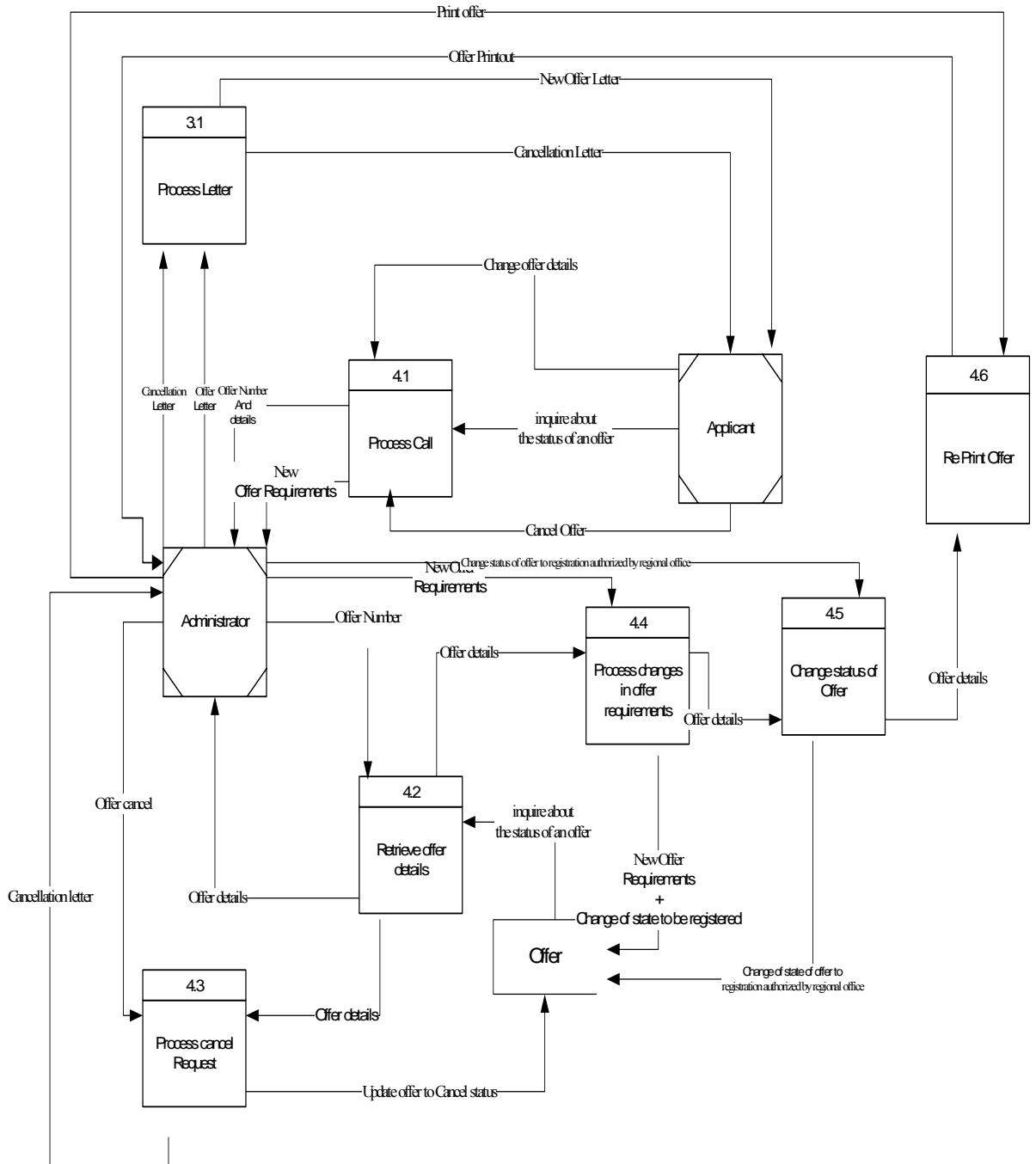
**Appendix D-2 DFD for Process Offer by Regional Office use case**



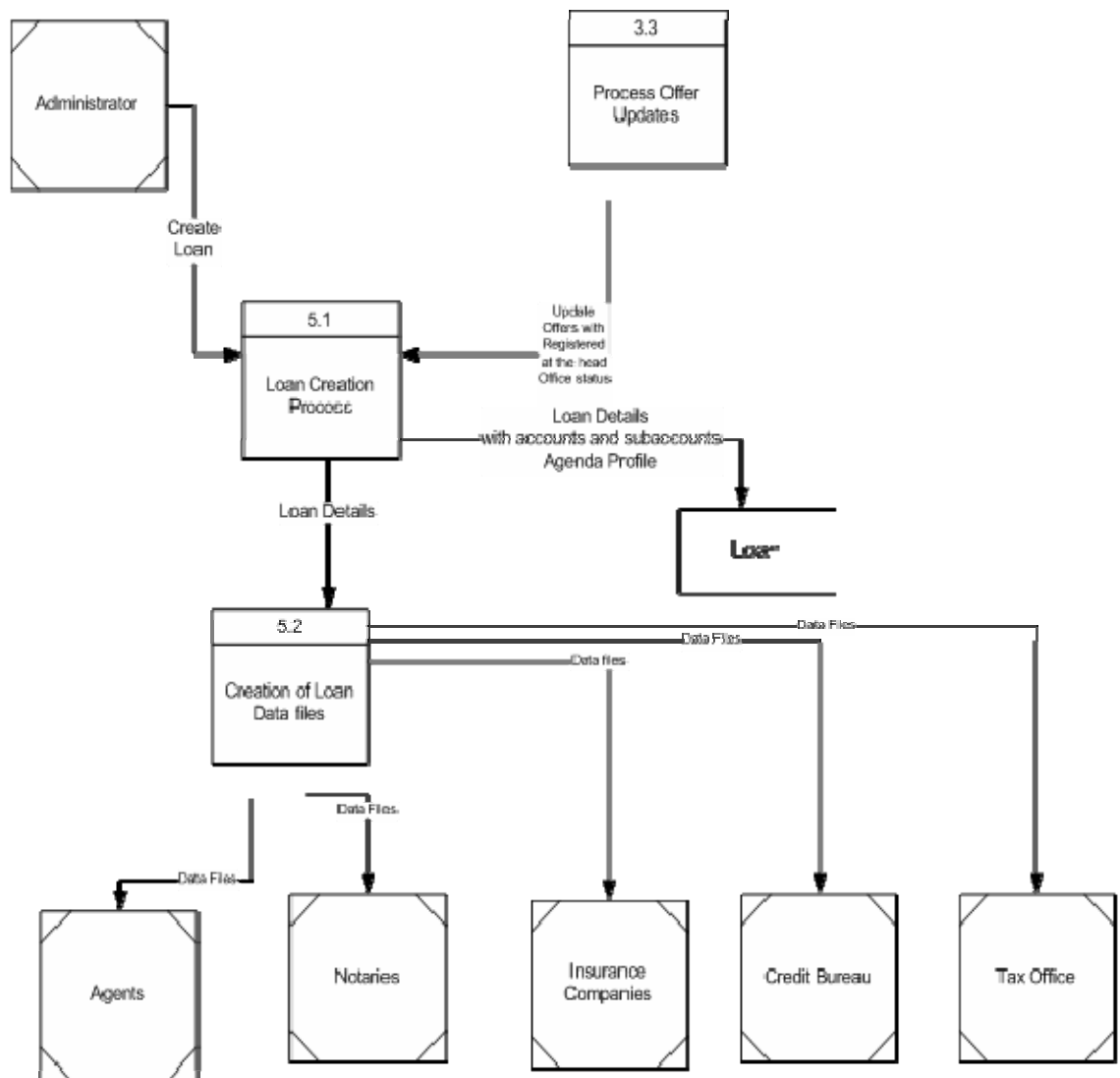
### Appendix D-3 DFD for the Process Offer Head Office use case



Appendix D-4 DFD for Maintain Offer use case

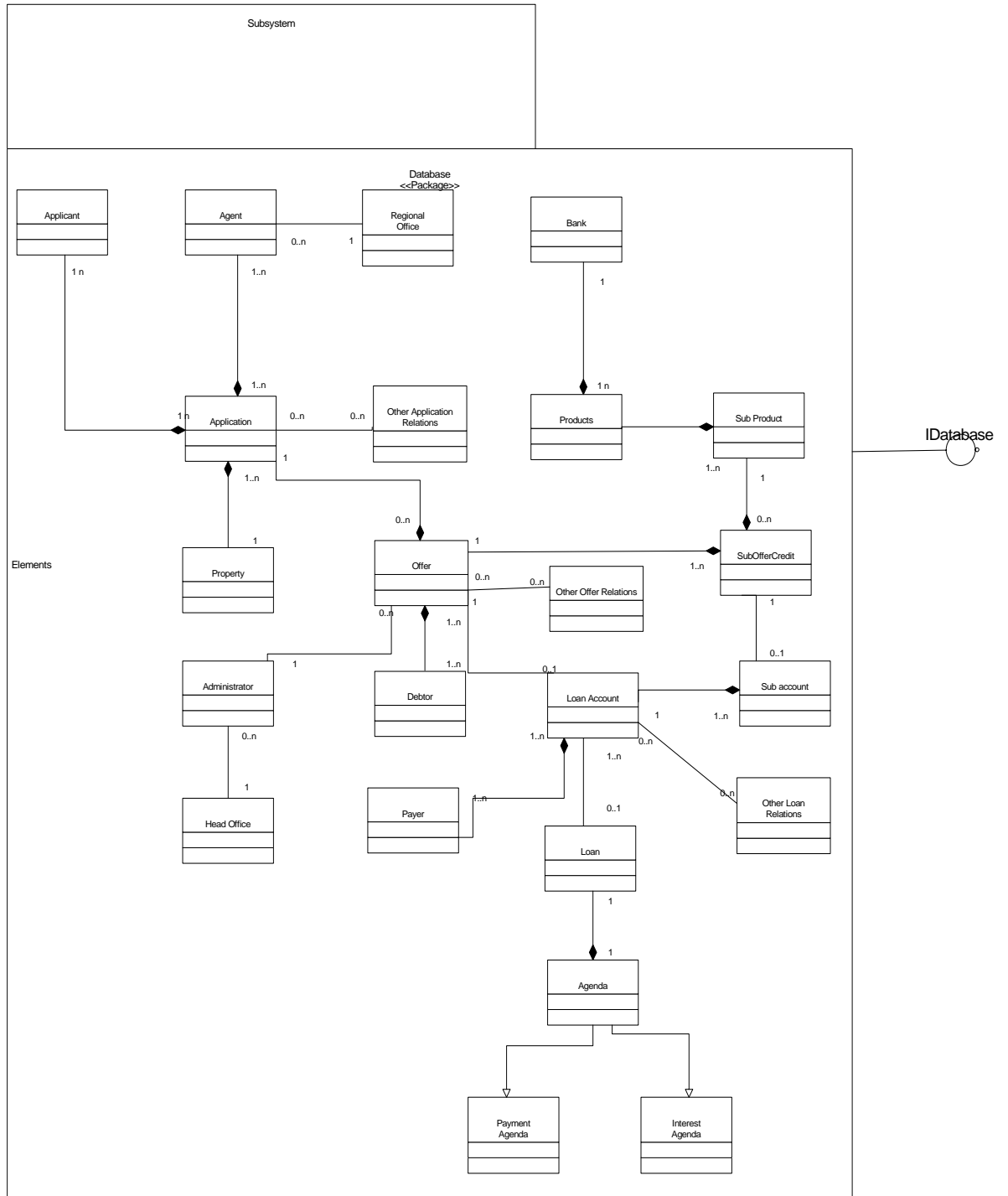


## Appendix D-5 DFD for the Loan Generation use case

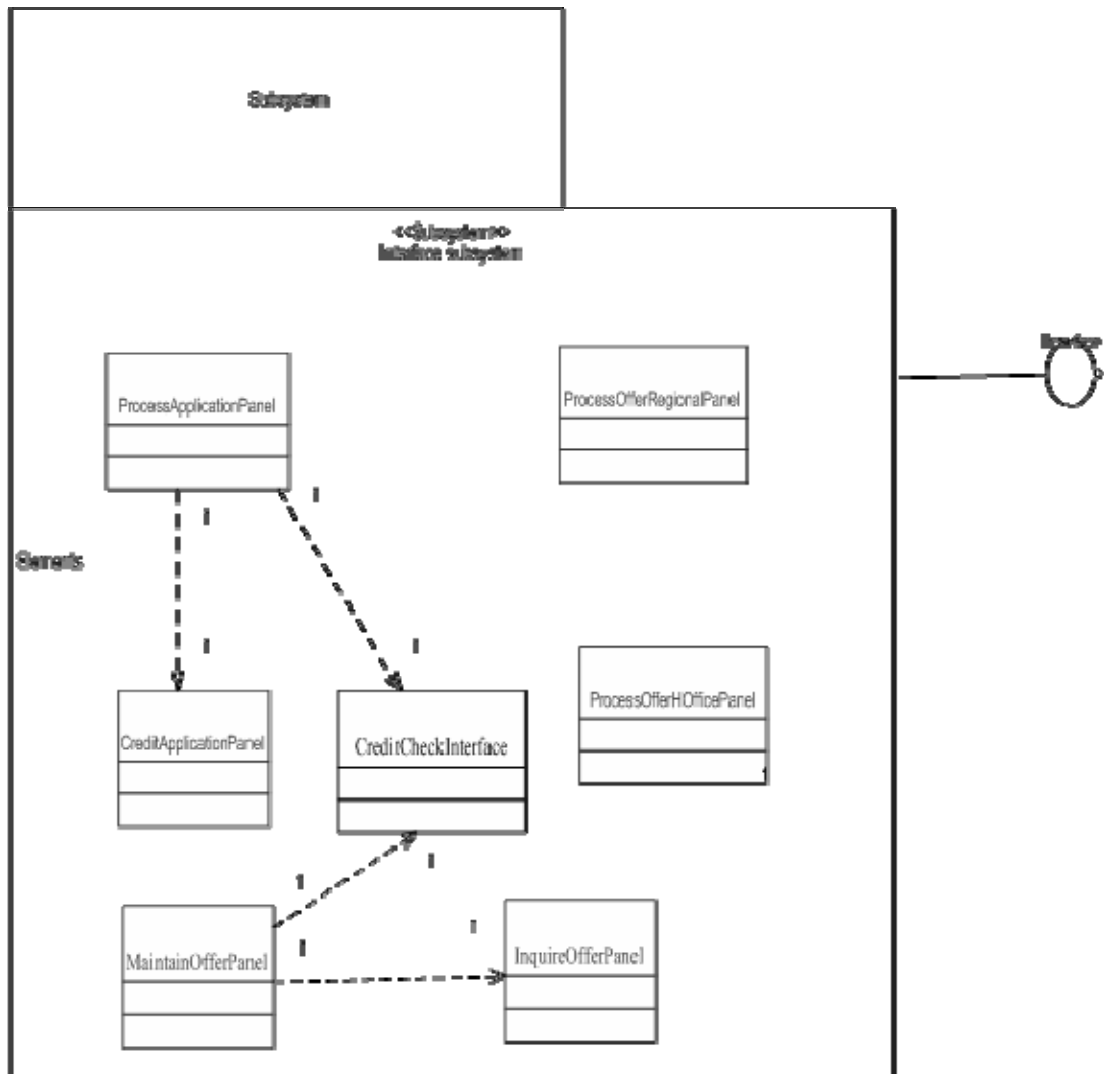


# Appendix E UML Diagrams for re-engineered system of case study

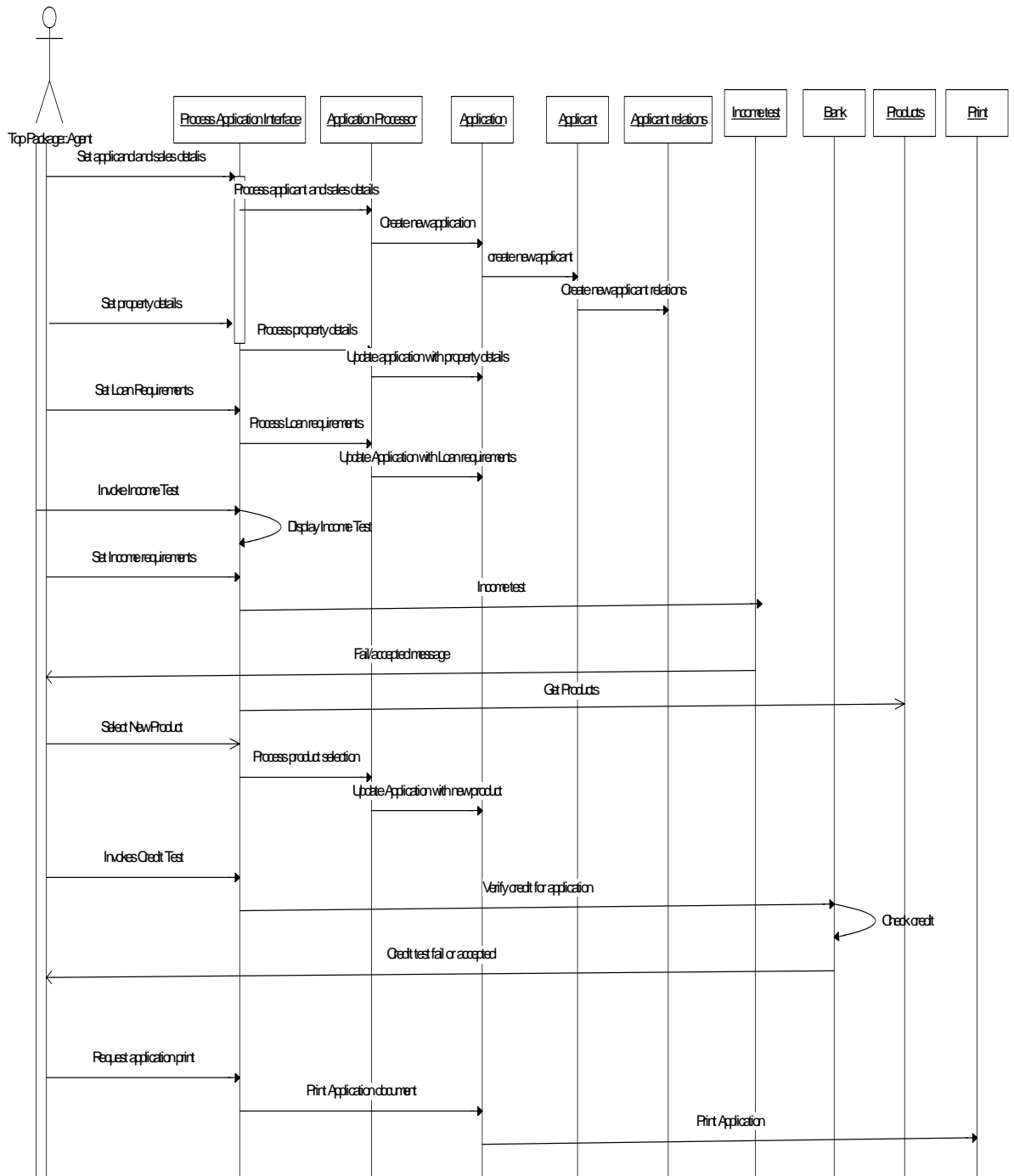
## Appendix E-1 Class diagram for the database subsystem package



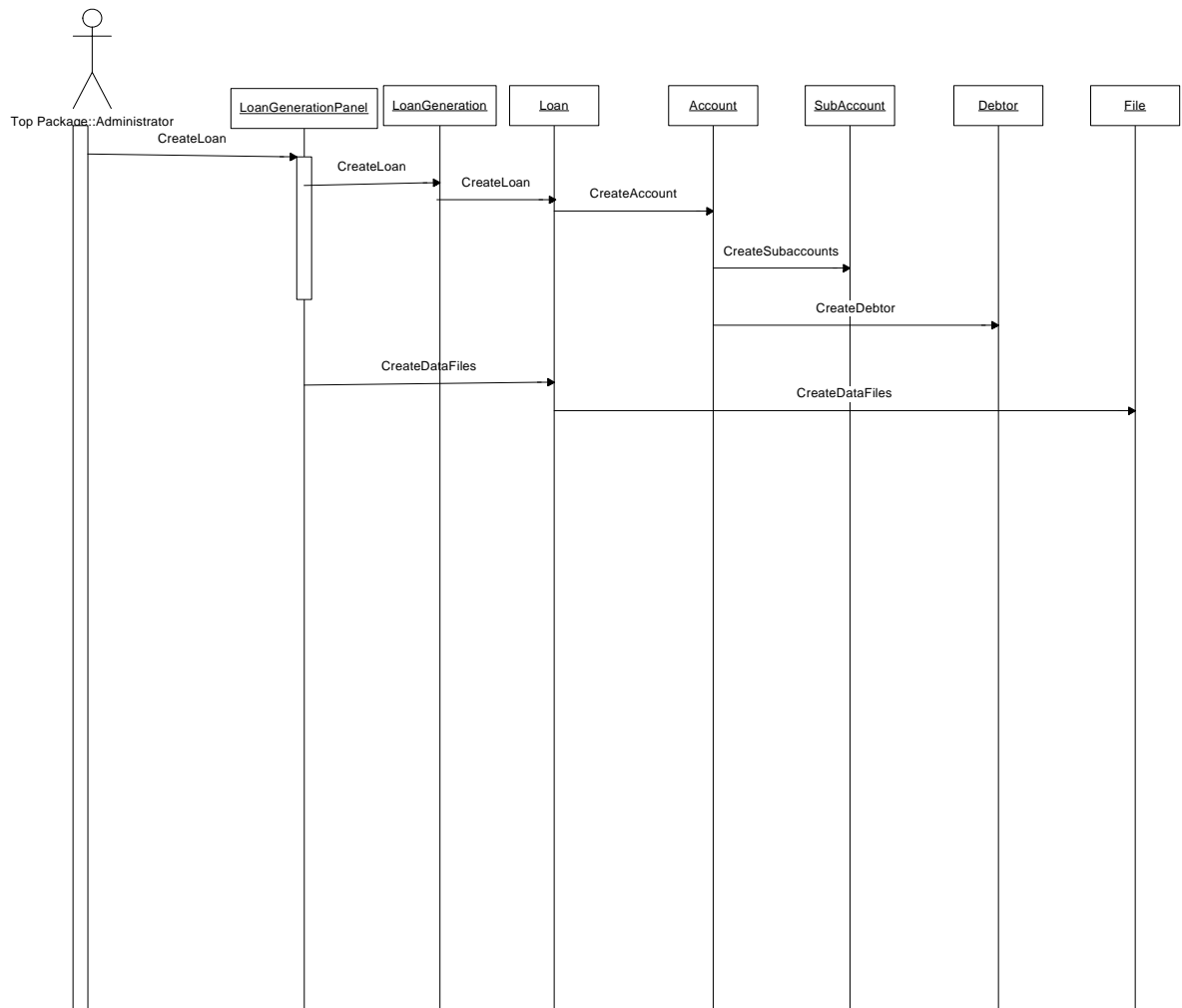
**Appendix E-2 Class diagram for the Interface subsystem package**



### Appendix E-3 Sequence diagram for the Process Application use case

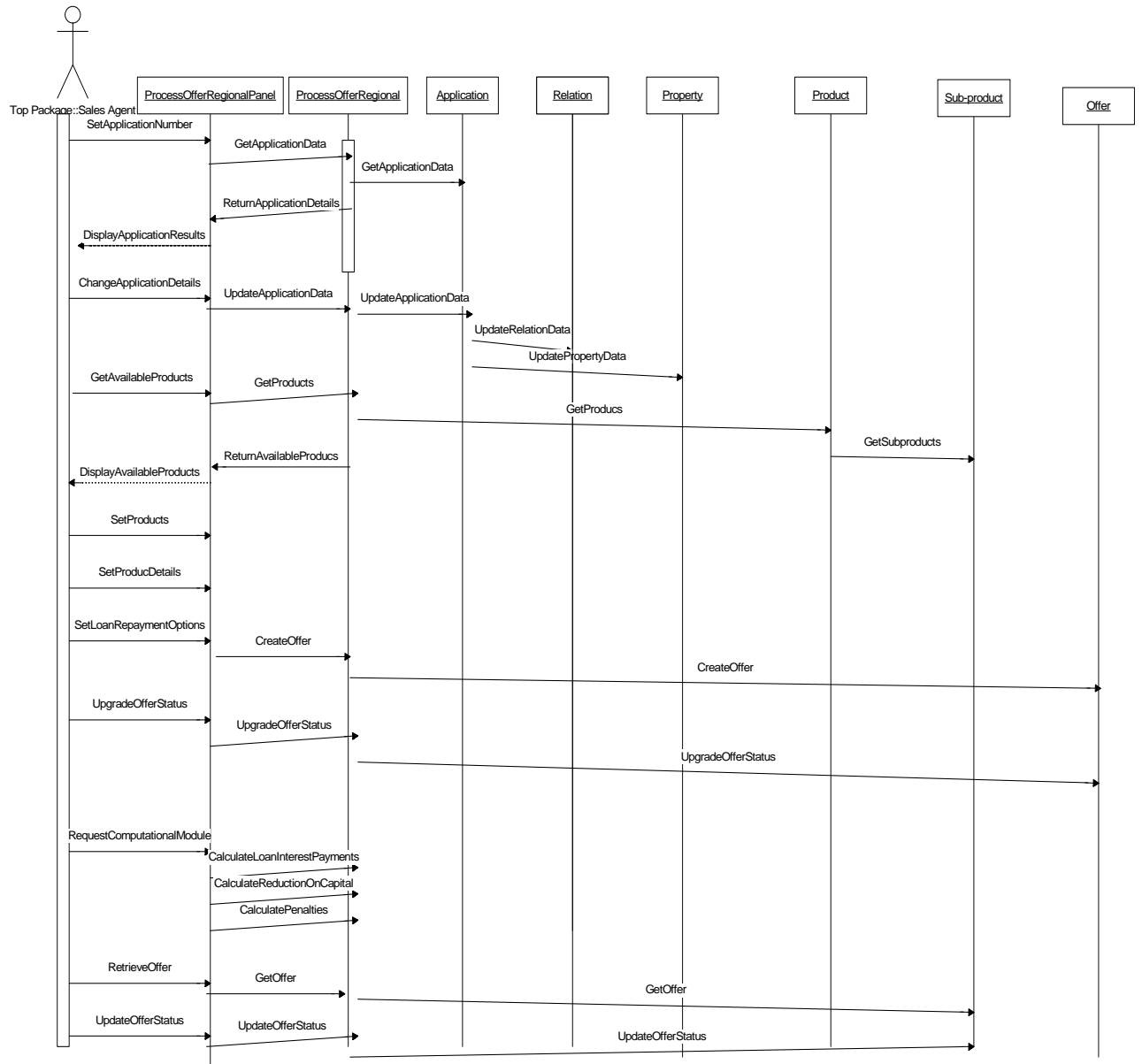


## Appendix E-4 Sequence diagram for the Loan Generation use case

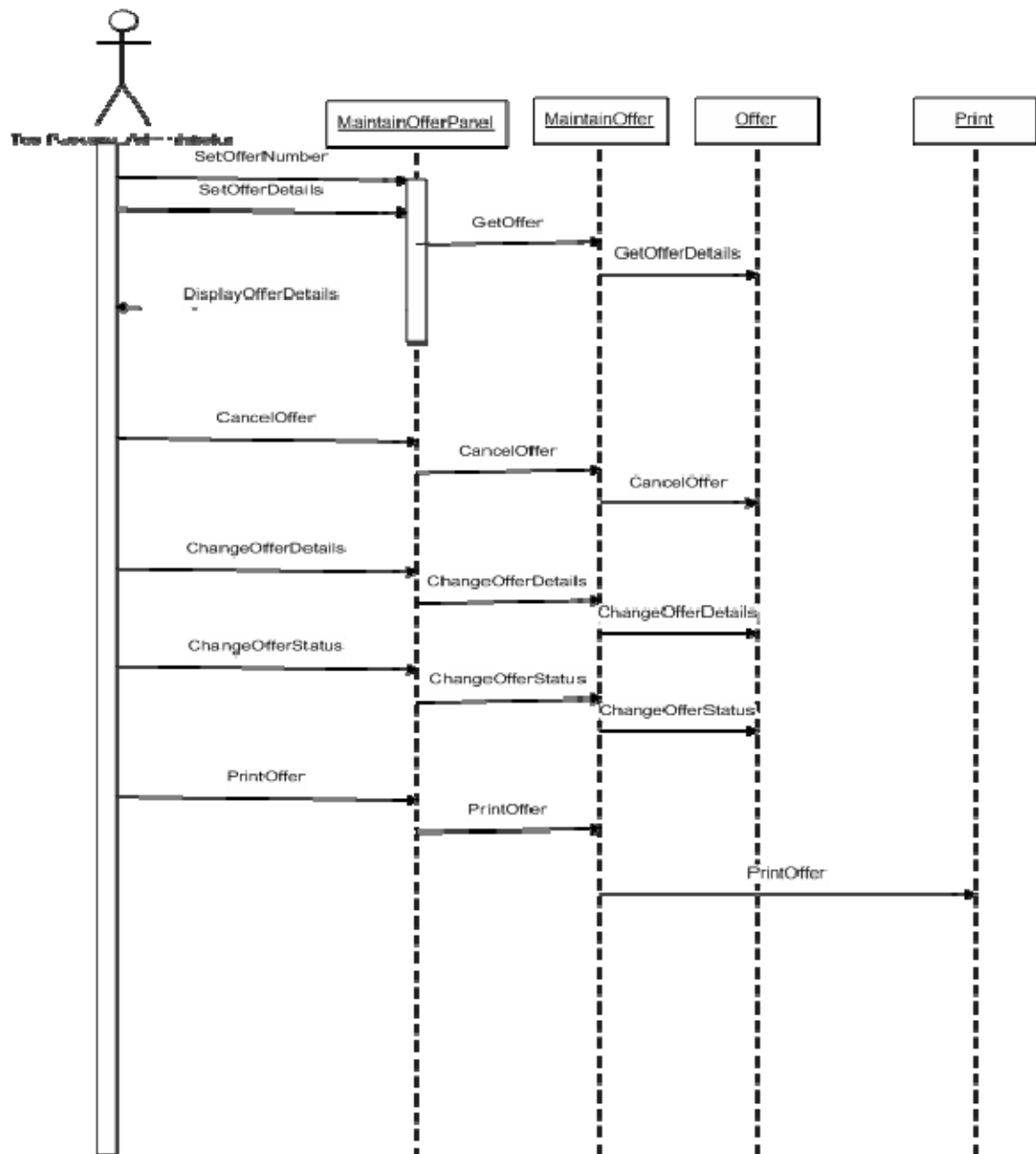




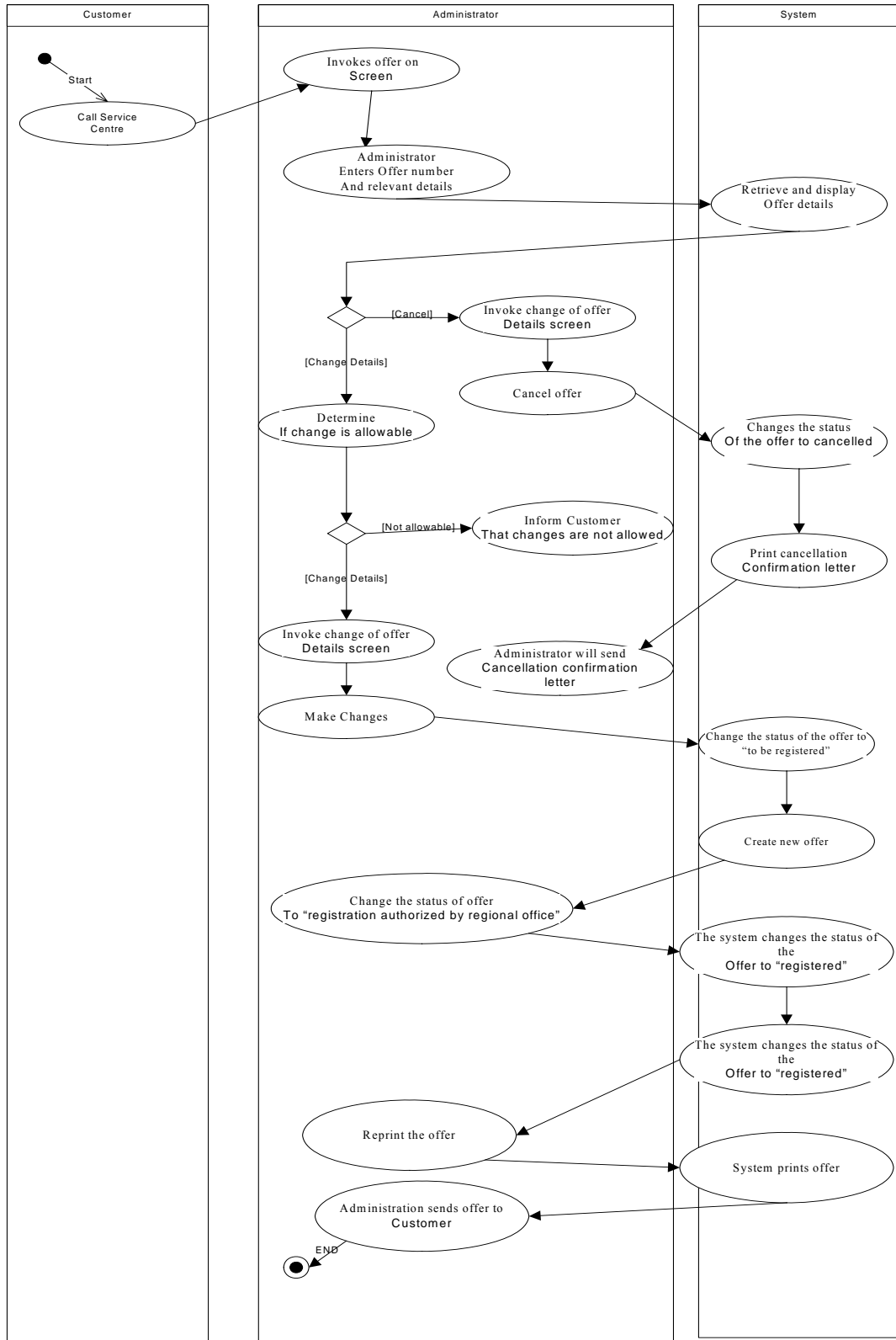
## Appendix E-5 Sequence diagram for the Process Offer by Regional Office use case



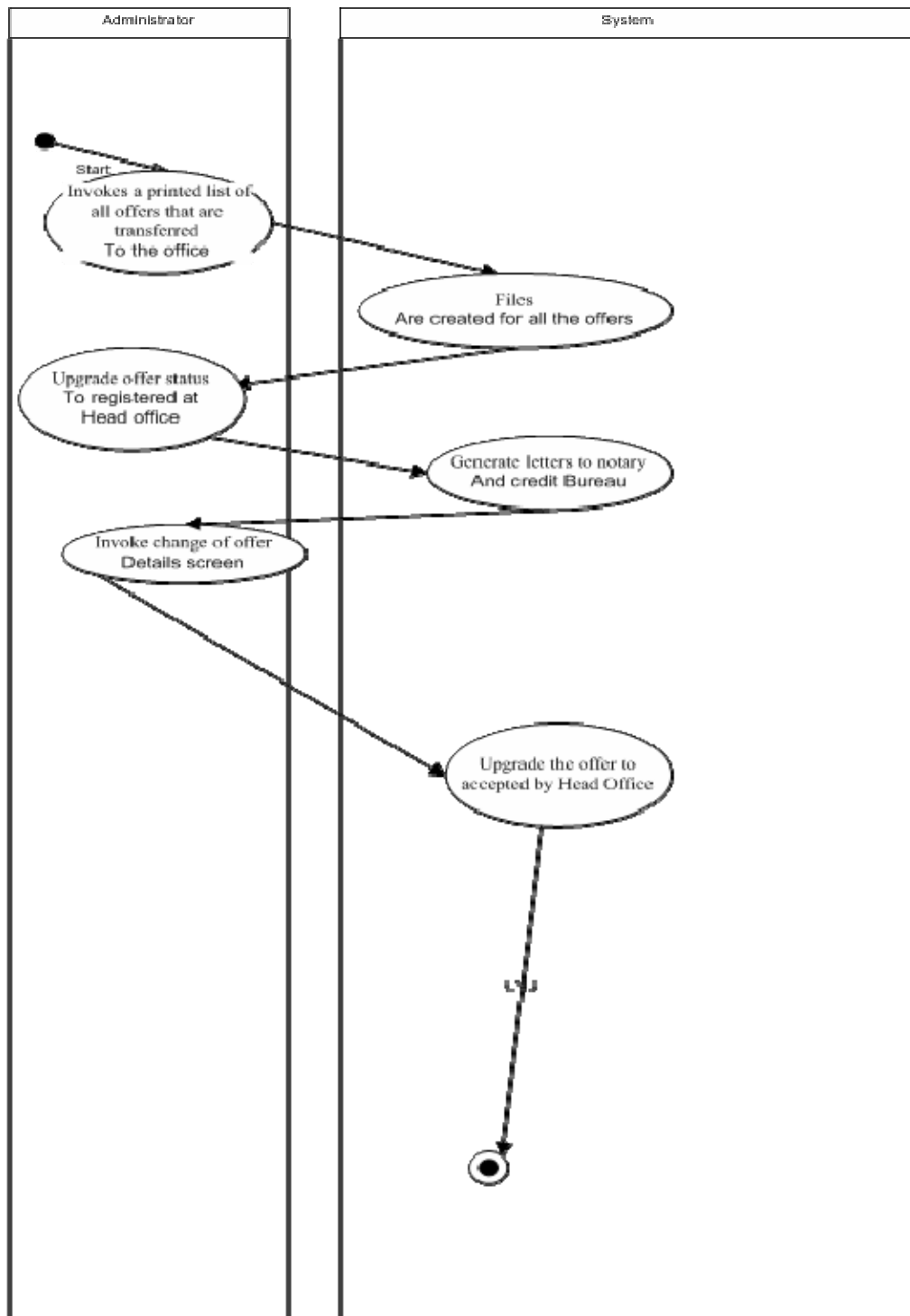
## Appendix E-6 Sequence diagram for the Maintain Offer use case



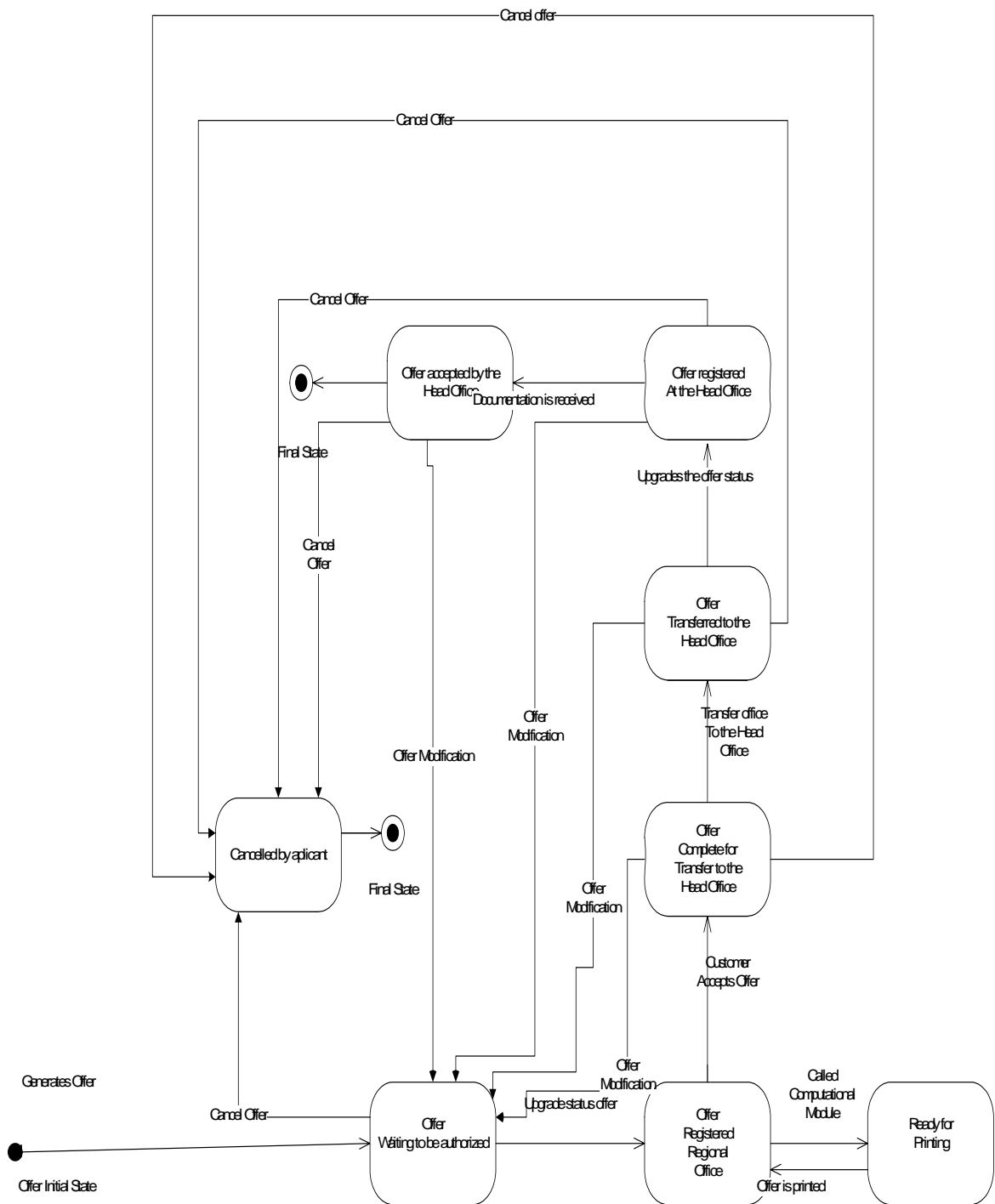
## Appendix E-7 Activity diagram for the Maintain Offer use case



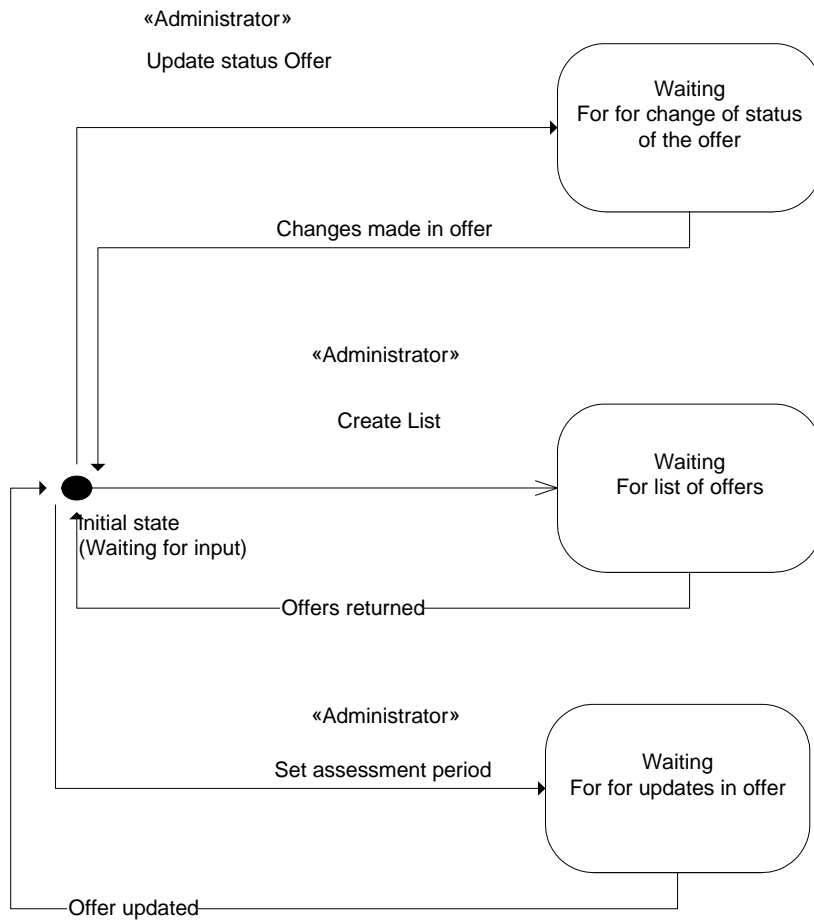
## Appendix E-8 Activity diagram for the Process Offer by Head office use case



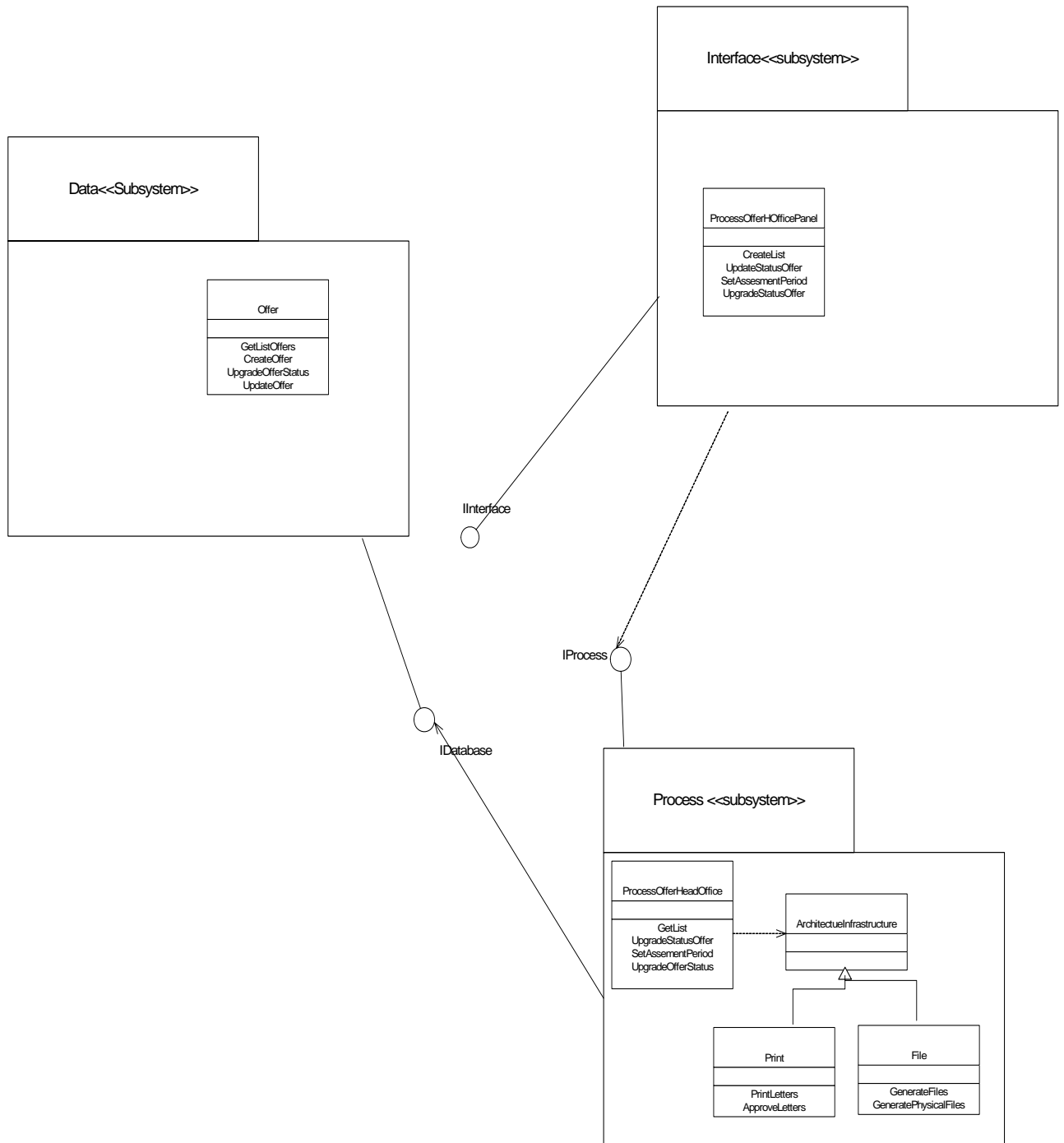
## Appendix E-9 State diagram for the Offer Object



## Appendix E-10 State diagram for the ProcessOfferHOfficePanel Object

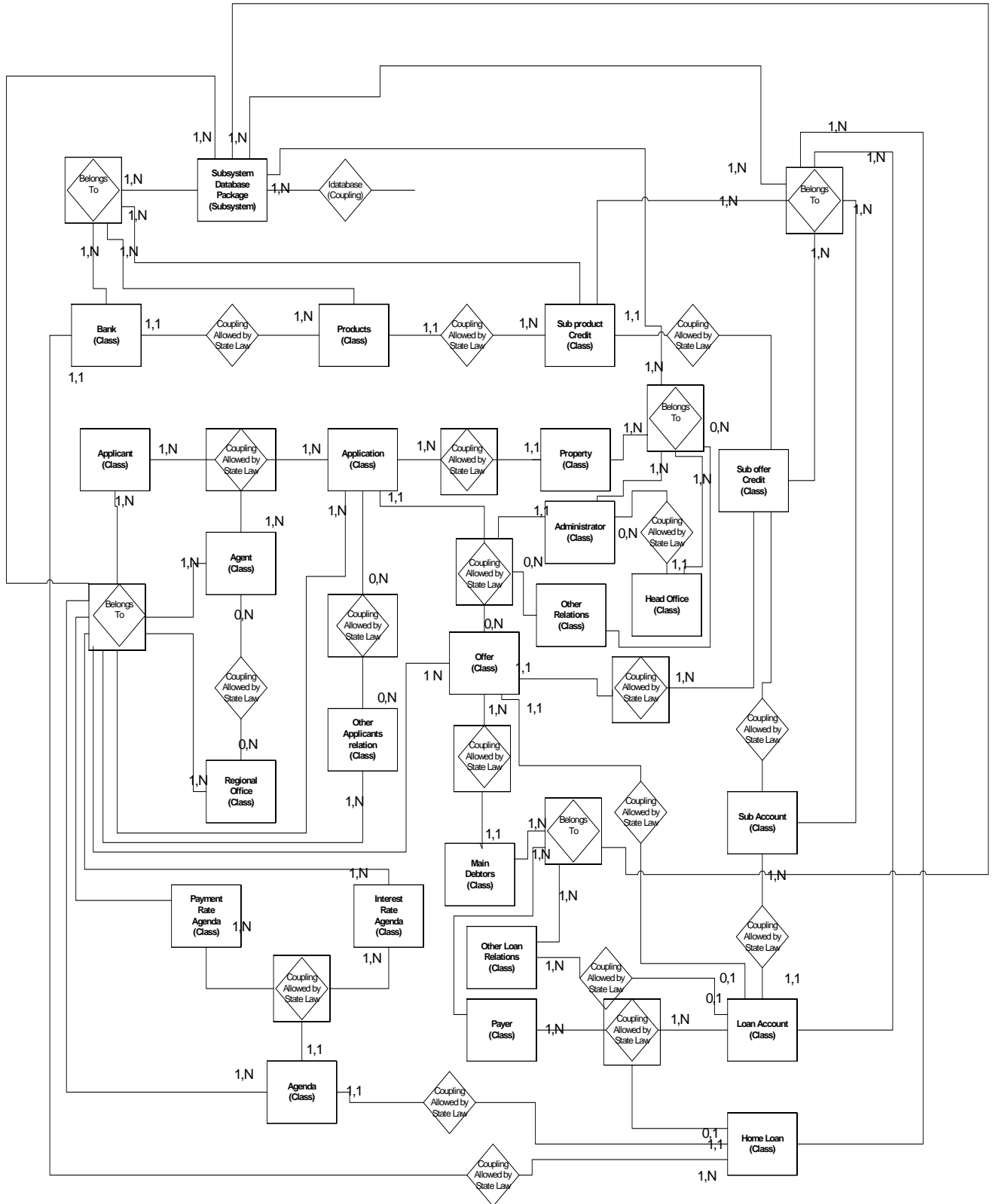


## Appendix E-11 Subsystem diagram with the classes required for the Process Head Office use case



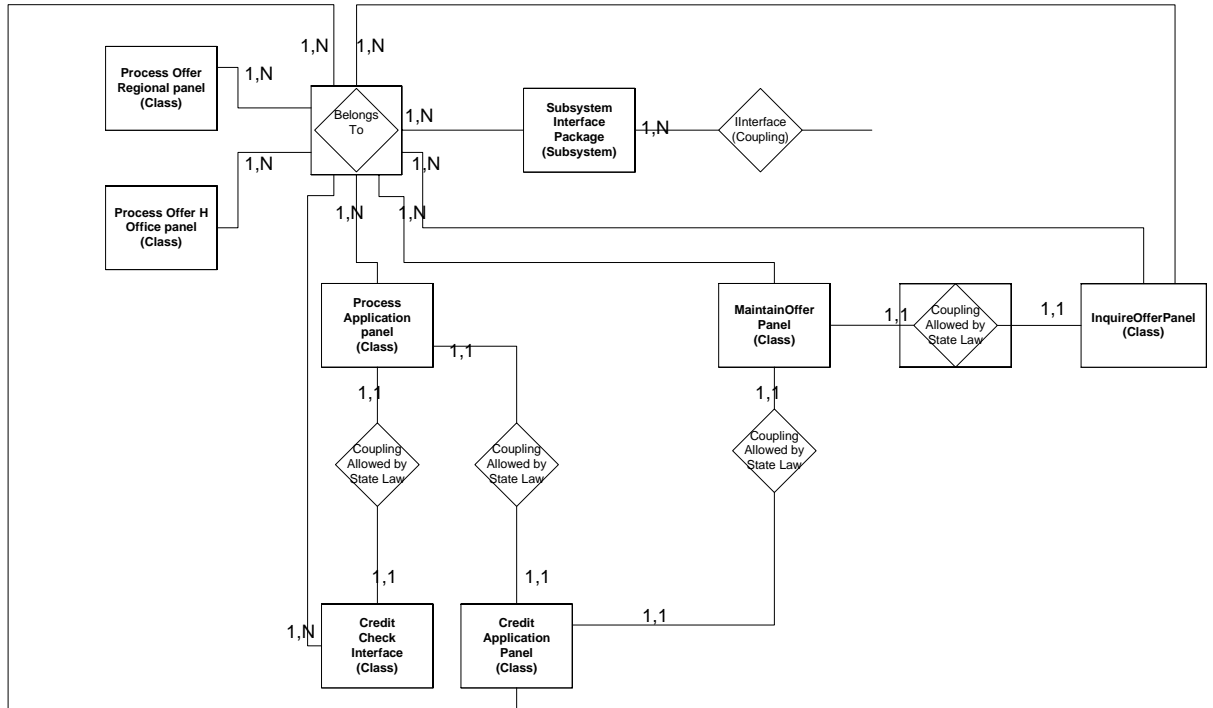
## Appendix F BWW Model

### Appendix F-1 BWW meta-model for the database package of the class diagram for the Process Offer Head Office

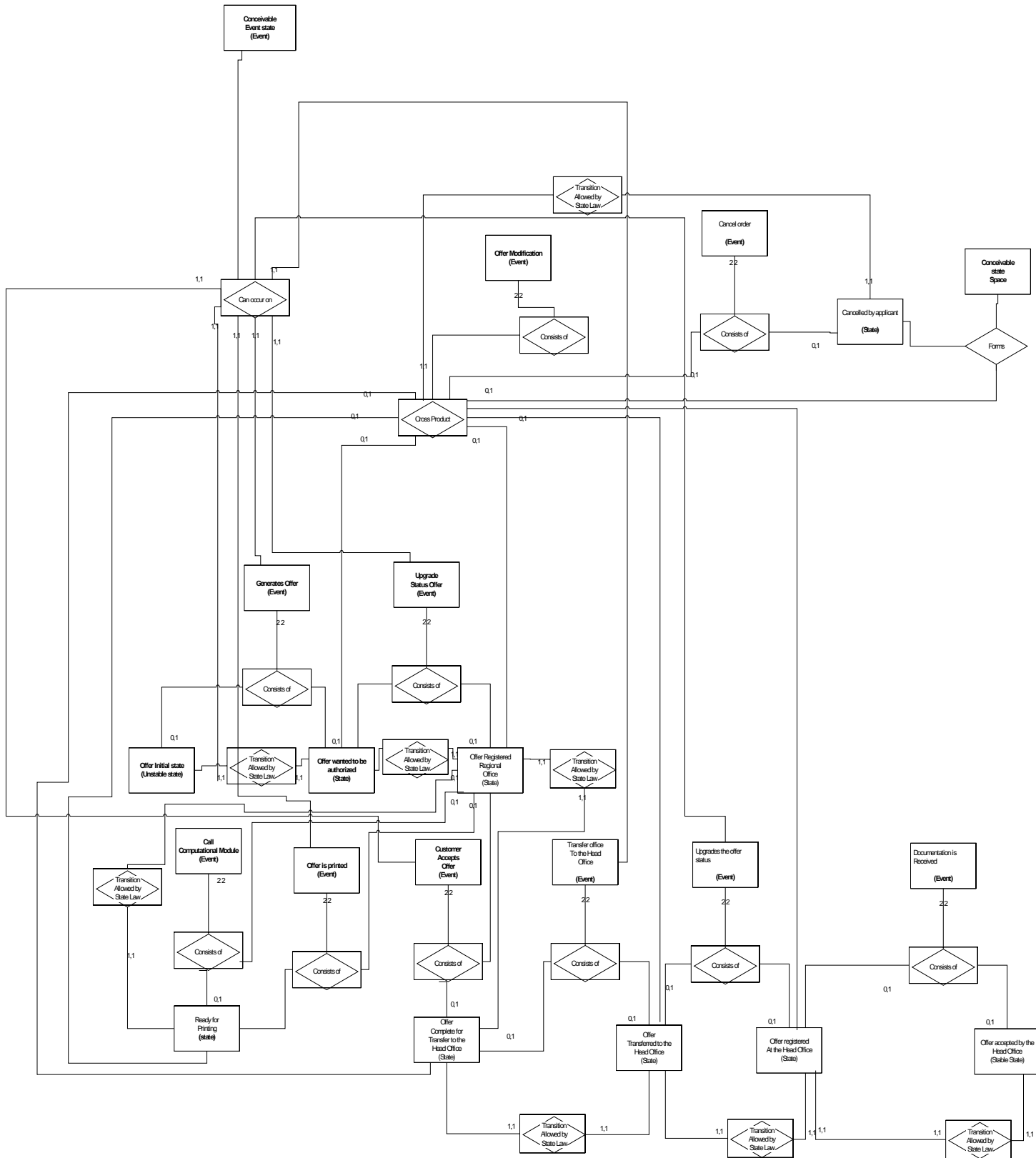




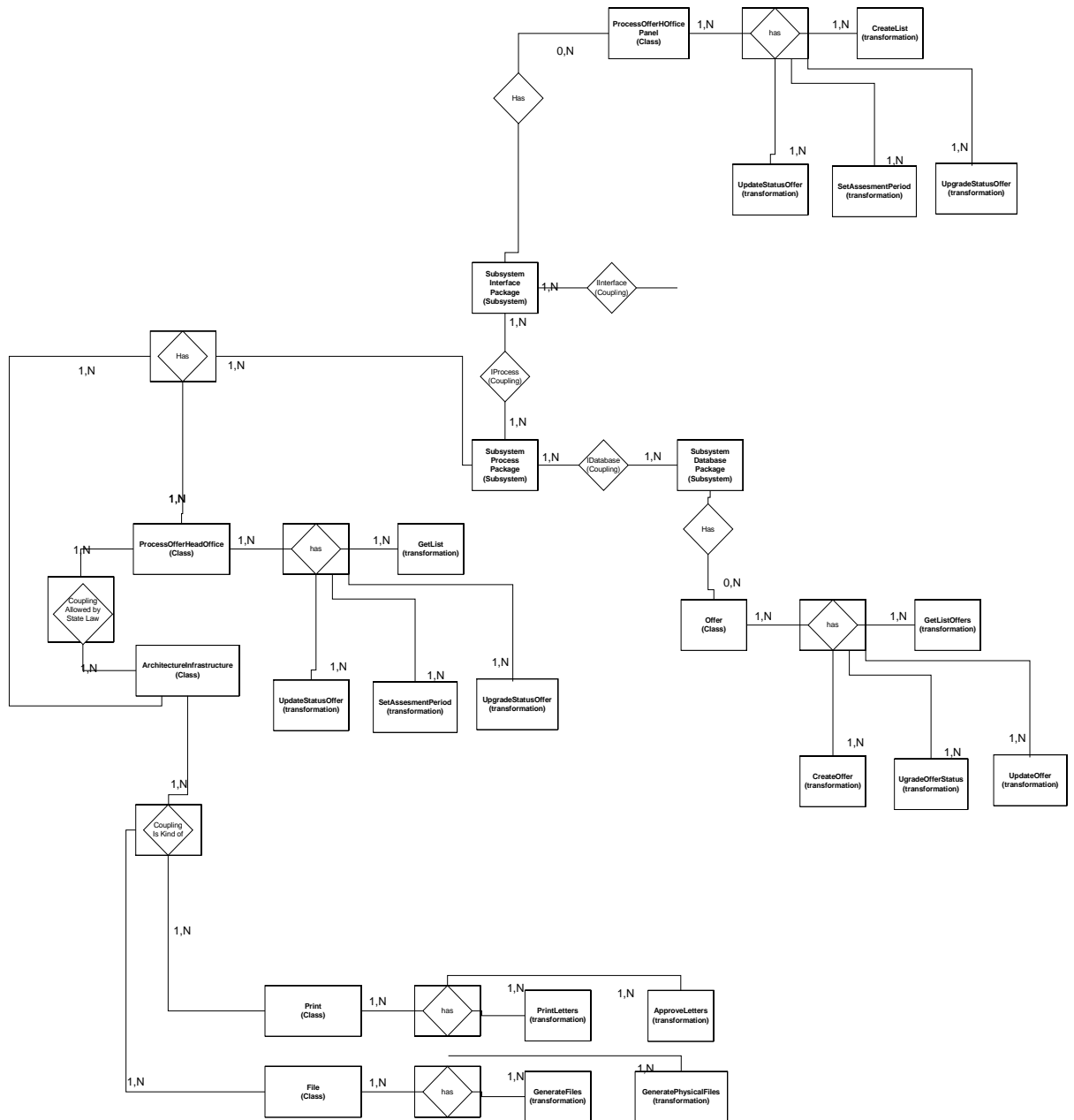
## Appendix F-2 BWW meta-model for class diagram of the interface package for the Process Offer Head Office



## Appendix F-3 BWW meta-model for the state diagram of the offer object



## Appendix F-4 BWW meta-model Subsystem diagram with the classes required for the Process Head Office use case



## Appendix G Consent letter



**THE UNIVERSITY OF  
SOUTHERN  
QUEENSLAND**

TOOWOOMBA QUEENSLAND 4350  
AUSTRALIA  
TELEPHONE +61 7 4631 2100  
FACSIMILE +61 7 4636 1762

**FACULTY OF BUSINESS**

Department  
of Information Systems

Tel. +61 7 4631 1630  
Fax. +61 7 4631 1631

29 March 2002

My name is Raul Valverde and I am currently a doctoral student of the Faculty of Business, University of Southern Queensland / Australia, I am undertaking research on component-based systems re-engineering of legacy systems. This research is important because it will help both practitioners as yourself and other researchers, understand better how to transfer business requirements embedded in legacy systems into modern re-engineered component-based systems.

I am seeking your interest and permission to become a participant (as a stakeholder) in my research to conduct a number of open interviews and conversations with you about your experiences and knowledge about the Home loan system that you currently maintain as part of your job duties. The interviews (2) will last about 1 hour, which will be taped and written up for analysis. Meeting place and time can be flexible to suit your needs.

The information you provide will be treated confidentially and will not be passed on to any other people in your organization, unless you want to do this. Individuals will not be identified in the research report. You have the right to decline or withdraw from the research at any time. No reasons for such a decision will be required nor will you be subject to any adverse consequences as a result of your withdrawal. If you do withdraw from the project, information collected will be used unless you state otherwise. If you don't want your information used, it will either be returned to you or destroyed. If you have a concern regarding the implementation of the project, you are welcome to contact the Secretary, Human Research Ethics Committee University of Southern Queensland, Toowoomba Queensland Australia 4350, or by telephone at 011 61(7) 4631 2956.

For any questions about the survey, or if you would like any additional information, please feel free to call me at (250) 8482424 EXT. 2989, or contact me by email [rvalverde@jmsb.concordia.ca](mailto:rvalverde@jmsb.concordia.ca). Thank you for your valuable contribution to this research effort.

**Raul Valverde**  
**Doctoral student**  
**University of Southern Queensland**



**THE UNIVERSITY OF  
SOUTHERN  
QUEENSLAND**

**FACULTY OF BUSINESS**

Department  
of Information Systems

TOOWOOMBA QUEENSLAND 4350  
AUSTRALIA  
TELEPHONE +61 7 4631 2100  
FACSIMILE +61 7 4636 1762

Tel. +61 7 4631 1630  
Fax. +61 7 4631 1631

**evaluation**  
**Participant Consent Form Research Project – The ontological**  
**of the requirements model when shifting from a traditional to a**  
**component-based paradigm in information systems re-**  
**engineering.**

**Re: Consent to Participate in Research**

I \_\_\_\_\_ have read the Information Sheet  
under the above research project and agree to become a participant in the study.

Organisation

Position

Signature

Date

Please keep a copy and return original to:

Raul Valverde  
Project Researcher  
University of Southern Queensland  
(514) 8482424 ext 2989  
rvalverde@jmsb.concordia.ca

## **Appendix H     Interview Protocol**

### **Interview Protocol for Participants**

Thank you for agreeing to participate in my doctoral research project. This interview should take about one hour.

The main aim of my research project is to contribute to generate better mechanisms for the transfer of requirements of legacy systems into re-engineered component based systems.

The research procedure will be based on a qualitative approach to data collection and analysis. This is mainly done through interviewing people involve in the maintenance of legacy systems. Specific comments you make are confidential, in terms of their source.

The information you provide will be treated confidentially and will not be passed on to any other people in your organization, unless you want to do this. Individuals will not be identified in the research report.

You have the right to decline or withdraw from the research at any time. No reasons for such a decision will be required nor will you be subject to any unfavourable consequences as a result of your withdrawal. If you do withdraw from the project, information collected will be used unless you state otherwise. If you don't want your information used, it will either be returned to you or destroyed.

Your organization and each participant will be entitled to access the research dissertation. The interview includes questions and discussion to gain your experience and views of the Home Loan Information System (HLIS). There are no right or wrong answers to the questions. I would like to make some notes during the interview.

If acceptable to you, interviews are taped to ensure I do not miss comments, and for subsequent analysis. Further interviews, if necessary, are a follow-up to see all areas are covered as part of my selected research process.

The expected benefits associated with your participation are:

- Identification of existing documents related to the HLIS.
- A better understanding of the operation of the HLIS.
- A better understanding of the architecture of the HLIS.

Do you have any questions about your participation in the project before we proceed?

Is it OK to proceed?

If no, can we discuss this further and agree on action?

## Interview Questions for Participant1

Start recorder.

Thanks for reading the interview protocol and agreeing to participate in my research project.

I'll begin the interview by recording some details, and then ask some questions.

Participant.....Interview #.....

Date.....Time.....Place.....

Role Title:.....

Time in Role.....

1. What are the documents available for the HLIS?

Participant's Response

2. Can you describe the content in general terms of each document?

Participant's Response

3. Can you describe the architecture of the HLIS in terms of programming languages, file systems, databases, networks and interfaces?

Participant's Response

4. Can you identify the actors that interact with the Offer and Application sub-system of the HLIS?

Participant's Response

5. What information do actors need to supply to the Offer and Application sub-system?

Participant's Response

6. What information do actors need to receive from the Offer and Application sub-system?

Participant's Response

7. What functionality does each actor expect from the Offer and Application sub-system?

Participant's Response

8. Can you describe the normal flow of events in a typical day of operations of the Offer and Application sub-system?

Participant's Response

9. Can you describe the normal flow of events in a typical day of operations of the Offer and Application sub-system?

Participant's Response

10. Do actors need to be informed about the events generated from the Offer and Application sub-system?

Participant's Response

Please let me know if you have any concerns about the conduct of this interview. Thank you for participating in the interview



## Follow-up Interview Questions for Participant

Start recorder.

Thanks for agreeing to meet with me again as a participant in my research project.

I'll begin the interview by recording some details, and then we can continue with our conversation.

Participant..... Interview #.....  
Date..... Time..... Place..... In the  
last interview we discussed:

To cover in this interview:

As advised in last interview, please let me know if you have any concerns about the conduct of this interview. Thank you for participating in interview.