Addis Ababa University
School of Graduate Studies
Faculty of Informatics

BUSINESS RULES OBJECT-ORIENTED METHOD [BROOM]

FOR

BUSINESS RULES SYSTEMS DEVELOPMENT

BY

TADESSE TAREKE KEBEDE

A Thesis submitted to the School of Graduates Studies of Addis Ababa University in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

July 2004

BUSINESS RULES OBJECT-ORIENTED METHOD [BROOM]

FOR

BUSINESS RULES SYSTEMS DEVELOPMENT

BY

TADESSE TAREKE

Name, Role and Signature of Members of the Examining Board

No.	Name	Role	Signature
1.	Dr. Dida Midekso	Chairperson	
2.	Dr. Yirsaw Ayalew	Advisor	
3.	Dr. Osei Nana Adjei	External Examiner	

Dedication

📖 "What can we say about such wonderful things as these?
If God is for us, who can ever be against us?" (Rom. 8:31). 📖

IN MEMORY OF MY GRANDPARENT:

WOLDE ABAKOYEW

&

ASRESE KASSA

📖 And David had *success* in all his undertakings;
for the LORD was with him (1 Sam. 18:14). 📖

Acknowledgments

There are a lot of people whose participations made the completion of this thesis possible. First of all, I can never adequately express my appreciation to my advisor, Dr. Yirsaw Ayalew, whose tireless support, encouragement, and tactful guidance made all the difference. Without his leadership and teamwork, none of this would have been attempted.

I am also very much indebted to my beloved family members for their love, patience, and support during the last two solid years. I would also like to thank Ethiopian Airlines Enterprise for granting me the scholarship award to attend this M.Sc. program.

A *very special thank you* for the following people who directly or indirectly contributed to this research work is appropriate. From Ethiopian Airlines: Ato Mesfin Tassew, Ato Kassahun H/Gabriel, Ato Berhanu Bizualem, Ato Moges Delelegn, Ato Daniel Abebe, Ato Alemayehu Assefa, Ato Tariku Molla, Ato Getachew Teshome, Mr. Hakan Bali, Ato Negassi Dirar, W/t Bethelehem Mengistu, Ato Berhanu G/Tekel, Ato Wondwossen Hailu, Ato Benyam Ayalew, Ato Theowodros Alemgenet, Ato Theowodros Asnake, W/ro Wubit Tesfaye, W/ro Almaz Tsegaye, W/ro Aster Fisseha, Ato Tensayberhan Teketel, Ato Tesfaye Yimer, Ato Getinet Tadesse, Ato Getahun Negatu, Ato Enkubahri Tsehai, and Ato Abdulhakim Meftuh. From my AAU colleagues: Ato Fikre Lema, Ato Semahegn Abebe, and W/t Addisalem Negash. From Cybersoft: Ato Tekesteberhan Habtu, Ato Mesfin Mulugeta, and Ato Assefa Dagne. From other places: Dr. Daniel Asrat, W/ro Meliha Mohammed, and Ato Zelalem Sintayehu.

Very heartfelt, sincere and special *thanks* to all of you for helping me in any kind to achieve this goal.

Above everything else, I praise God for His love, care, and incredible support in attaining this achievement!

Abstract

Business rules systems are automated systems in which the business rules are separated from the procedural codes and database constructs. In addition, these business rules are shared across data stores, user interfaces and applications. The formal way of developing such systems is known as Business Rules Approach (BRA). Any approach has its own tradeoffs. To that end, BRA and other approaches such as Object-Oriented Approach (OOA) have their own strengths and weaknesses in developing business rules systems. This research is targeted to bring about a new, lightweight, and viable method known as Business Rules Object-Oriented Method (BROOM, for short) by taking the strengths of both approaches.

Persistency of business rules is a prerequisite for separating business rules from the procedural codes and database constructs. This is, however, a less addressed issue by OOA although the concept of objects has been spread since 1967. Furthermore, this separation of business rules is a basic principle of BRA in which its proponents consider that OOA and the Unified Modeling Language™ (UML®) are incapable of addressing the development problem of a business rules system.

In this research, to alleviate the problems stated above, UML was extended with stereotype extension mechanism to model the business rules flavors (i.e., term, fact and rule). The support tools for the method were also prototyped. These support tools include Metamodel Management Tool (MMT), Rules Management Tool (RMT), and Data Management Tool (DMT). Moreover, a general architectural framework for the method was provided from which an extended version of the framework was drawn. This extended framework utilized Java, eXtensible Markup Language (XML), and eXtensible Stylesheet Language (XSL) in order to realize a design pattern for a lightweight Rule Engine.

Last but not least, BROOM was demonstrated with a system and the extended framework as a case study. The system rigorously used the prototype tools after it was modeled with the UML extensions made earlier. Finally, the results of the case study were analyzed.

Keywords: Method, Business Rules, Business Rules System, Rule Engine, Support Tools

Table of Contents

List of Tables ..... **iii**

List of Figures ..... **iv**

Listings ..... **v**

Acronyms and Abbreviations ..... **vi**

CHAPTER 1: INTRODUCTION **1**

1.1 OVERVIEW **1**

1.1.1 BUSINESS RULES SYSTEMS 1

1.1.2 DEVELOPMENT APPROACHES' TRADE-OFF 1

1.2 PROBLEM STATEMENT **3**

1.3 RESEARCH QUESTIONS AND HYPOTHESES **4**

1.4 OBJECTIVES **4**

1.5 ORGANIZATION OF THE THESIS **6**

CHAPTER 2: BACKGROUND **7**

2.1 BUSINESS RULES APPROACH **7**

2.2 OBJECT-ORIENTED APPROACH **7**

2.3 UNIFIED MODELING LANGUAGE (UML) **8**

2.3.1 ABOUT UML 8

2.3.2 ARCHITECTURE OF UML 8

2.4 DESIGN PATTERNS AND FRAMEWORKS **9**

2.4.1 ABOUT DESIGN PATTERNS AND FRAMEWORKS 9

2.4.2 SELECTING DESIGN PATTERNS 10

2.4.3 DOMAIN SPECIFIC DESIGN PATTERNS 11

2.5 SUMMARY **12**

CHAPTER 3: RELATED WORK **13**

3.1 UML EXTENSION MECHANISMS **13**

3.2 MODELING DESIGN PATTERNS **14**

3.3 PERSISTENCY USING DESIGN PATTERNS **14**

3.4 AUTOMATING DESIGN PATTERNS **16**

3.5 RULE PATTERNS **16**

3.6 REQUIRED TOOLS **17**

3.6.1 REPOSITORY SYSTEMS 17

3.6.2 RULES ENGINES 18

3.6.3 INTEGRATED SUPPORT TOOLS 19

3.5 SUMMARY **20**

CHAPTER 4: DEVELOPMENT OF BROOM 21
4.1 INTRODUCTION 21
4.2 EXTENDING UML FOR BROOM 21
4.2.1 EXTENDING UML 21
4.2.2 UML CLASS DIAGRAM ANALYSIS FOR SUPPORTING BROOM 22
4.2.3 MODELING THE FLAVORS OF BUSINESS RULES 25
4.2.4 METAMODEL FOR BUSINESS RULES 42
4.3 ARCHITECTURAL FRAMEWORK DEVELOPMENT 45
4.3.1 GENERAL FRAMEWORK FOR BROOM 45
4.3.2 IMPLEMENTING THE FRAMEWORK OF BROOM 47
4.4 IMPLEMENTATION PERSPECTIVES 48
4.4.1 PROGRAMMING LANGUAGES 49
4.4.2 RULE ENGINE 49
4.4.3 REPOSITORY SYSTEMS 50
4.4.4 BASIC DEVELOPMENT TOOLS 50
4.4.5 SUPPORT TOOLS PROTOTYPE DEVELOPMENT 52
4.5 SUMMARY 61
CHAPTER 5: CASE STUDY 62
5.1 PURPOSE 62
5.2 A CASE STUDY AND ITS BUSINESS RULES 62
5.2.1 SELECTION AND ITS RATIONALE 62
5.2.2 BUSINESS RULES IDENTIFICATION 63
5.3 APPLYING BROOM 66
5.3.1 MODELING THE PAYROLL SYSTEM 66
5.3.2 AUTOMATING THE SYSTEM WITH THE SUPPORT TOOLS 69
5.3.3 AMENDING THE BUSINESS RULES 80
5.4 ANALYSIS OF THE RESULTS 84
5.4.1 CLASS DIAGRAMMING ISSUE 84
5.4.2 VOCABULARY ISSUE 85
5.4.3 REPORTING ISSUE 85
5.4.4 PERFORMANCE ISSUE 86
5.4.5 CLIENT/SERVER ISSUE 86
5.4.6 SECURITY ISSUE 87
5.5 SUMMARY 87
CHAPTER 6: CONCLUSIONS 89
6.1 CONCLUSIONS 89
6.2 SUMMARY OF CONTRIBUTIONS 90
6.3 FUTURE WORK 91
BIBLIOGRAPHY 92
GLOSSARY 98
INDEX 103

LIST OF TABLES

TABLE 1- RESEARCH QUESTION, SUB-QUESTIONS AND HYPOTHESES....4
TABLE 2: LIST OF ENTITIES DATA MODEL DESIGN....29
TABLE 3: LIST OF ATTRIBUTES DATA MODEL DESIGN....29
TABLE 4: DESCRIPTION OF THE EXTENSIONS USED FOR TDM....30
TABLE 5: LIST OF FACTS DATA MODEL DESIGN....33
TABLE 6: DESCRIPTION OF THE EXTENSIONS USED FOR FDM....35
TABLE 7: TYPES OF RULES....36
TABLE 8: CONSTRAINT AND GUIDELINE DATA MODEL DESIGN....38
TABLE 9: ACTION ENABLER DATA MODEL DESIGN....39
TABLE 10: COMPUTATION DATA MODEL DESIGN....39
TABLE 11: DESCRIPTION OF THE EXTENSIONS USED FOR RDM....41
TABLE 12: EXPLANATION FOR THE GENERAL FRAMEWORK....46
TABLE 13: EXPLANATION FOR THE EXTENDED FRAMEWORK....48
TABLE 14: BASIC DEVELOPMENT TOOLS FOR DEVELOPING A BUSINESS RULES SYSTEM....51
TABLE 15: BASIC STEPS FOLLOWED AND PERFORMED IN DEVELOPING THE PROTOTYPES....52
TABLE 16: LIST OF ENTITIES, WHICH ARE SUBSET OF THE TERMS OF THE SYSTEM....63
TABLE 17: LIST OF ATTRIBUTES, WHICH ARE SUBSET OF THE TERMS OF THE SYSTEM....63
TABLE 18: LIST OF FACTS/RELATIONSHIPS OF THE SYSTEM....64
TABLE 19: LIST OF SAMPLE RULES OF THE SYSTEM....65
TABLE 20: LIST OF AMENDED RULES OF THE SYSTEM....83

LIST OF FIGURES

FIGURE 1: INTERACTIONS BETWEEN MODELS, FRAMEWORK, AND SUPPORT TOOLS 5
FIGURE 2: A SUMMARY OF THE MAJOR EVOLUTION OF UML [44] 8
FIGURE 3: OMG UML 4-LAYER ARCHITECTURE [60] 9
FIGURE 4: MODELING A TERM AT FIRST GLANCE 27
FIGURE 5: TERMS AND A BUSINESS RULES REPOSITORY SYSTEM 28
FIGURE 6: EXTENDED UML CLASS DIAGRAM FOR TERMS 30
FIGURE 7: FACTS AND BUSINESS RULES REPOSITORY SYSTEM 32
FIGURE 8: EXTENDED UML CLASS DIAGRAM FOR FACTS 34
FIGURE 9: RULES AND A BUSINESS RULES REPOSITORY SYSTEM 37
FIGURE 10: EXTENDED UML CLASS DIAGRAM FOR RULES 40
FIGURE 11: BUSINESS RULES METAMODEL USING EXTENDED UML CLASS DIAGRAMS 42
FIGURE 12: A GENERAL ARCHITECTURAL FRAMEWORK DESIGN FOR BROOM 45
FIGURE 13: EXTENDED FRAMEWORK OF BROOM USING BOGLAEV'S WORK 47
FIGURE 14: SIMPLE BUSINESS RULES METAMODEL: ENTITIES, ATTRIBUTES, FACTS & RULES 55
FIGURE 15: METAMODEL MANAGEMENT TOOL - ATTRIBUTE MANAGER 56
FIGURE 16: METAMODEL MANAGEMENT TOOL - ENTITY MANAGER PREVIEW WINDOW 57
FIGURE 17: METAMODEL MANAGEMENT TOOL - FACT MANAGER 57
FIGURE 18: DATA MANAGEMENT TOOL ALLOWING DATA ENTRY FACILITIES 58
FIGURE 19: RULE MANAGEMENT TOOL WHILE EDITING A RULE EXPRESSION 59
FIGURE 20: EMPLOYEE CLASS DIAGRAM 66
FIGURE 21: MONTHLY PAYROLL CLASS DIAGRAM 67
FIGURE 22: BASE CLASS DIAGRAM 68
FIGURE 23: CONTAINMENT/AGGREGATION CLASS DIAGRAM FOR THE PAYROLL APPLICATION 69
FIGURE 24: DEFINING THE ATTRIBUTES WITH THE ATTRIBUTE MANAGER 70
FIGURE 25: DEFINING THE FACTS/RELATIONSHIPS BETWEEN EMPLOYEE AND THE ATTRIBUTES 70
FIGURE 26: DEFINING EMPLOYEE AND THE *FACTS* WITH ATTRIBUTES VIA ENTITY MANAGER 71
FIGURE 27: DEFINING MONTHLYPAY WITH *FACTS AND* ATTRIBUTES VIA ENTITY MANAGER 71
FIGURE 28: FACTS MANAGER DEFINING THE RELATIONSHIPS BETWEEN THE TWO TABLES 72
FIGURE 29: RULE MANAGEMENT TOOL (RMT) IN ACTION 73
FIGURE 30: LIST OF RULES ENTERED IN THE REPOSITORY SYSTEM BY RMT 73
FIGURE 31: DATA MANAGEMENT TOOL (DMT) IN ACTION 77
FIGURE 32: ENTERING DATA OF MONTHLYPAY 78
FIGURE 33: APPLYING THE RULE STORED IN THE REPOSITORY SYSTEM 78
FIGURE 34: PARTIAL PREVIEW OF THE STATUS OF THE RULES EXECUTED 80
FIGURE 35: INCORPORATING THE CHANGES OF THE SYSTEM VIA ATTRIBUTE MANAGER 81
FIGURE 36: INCORPORATING THE *FACTS* OF THE SYSTEM VIA THE FACT MANAGER 82
FIGURE 37: LIST OF UPDATED RULES STORED IN THE REPOSITORY SYSTEM 83
FIGURE 38: EMPLOYEE DATA ENTRY SCREEN AFTER AMENDING THE CHANGES 84

LISTINGS

LISTING 1: CONVENTION FOR XML FILES FORMAT 54
LISTING 2: SAMPLE XSL FILE CREATED BY RMT 54
LISTING 3: RULES STORED IN AN XSL FILE GENERATED BY RMT 74
LISTING 4: SAMPLE XML CONTENTS GENERATED BY DMT 76

ACRONYMS AND ABBREVIATIONS

Acronym/ Abbreviation	Meaning
BEST	BROOM Essential Support Tools
BRA	Business Rules Approach
BROOM	Business Rules Object-Oriented Method
DBMS	Database Management System
DDL	Data Definition Language
DMT	Data Management Tool
FDM	Facts Data Model
GOF	Gang Of Four
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
MMT	Metamodel Management Tool
OO	Object-Oriented
OOA	Object-Oriented Approach
OOP	Object-Oriented Programming
ORDBMS	Object Relational Database Management System
RAD	Rapid Application Development
RDBMS	Relational Database Management System
RDM	Rules Data Model
RMT	Rules Management Tool
SQL	Structured Query Language
TDM	Terms Data Model
UML	Unified Modeling Language
VB6	Microsoft Visual Basic version 6.0
XML	Extensible Markup Language
XSL	Extensible Style-sheet Language
XSLT	Extensible Style-sheet Language Transformation

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

1.1.1 BUSINESS RULES SYSTEMS

Maintenance cost accounts for more than 50% of software resource expenditures [64]. In the case of a business rules system, this cost is basically manifested since business rules are commonly buried in procedural codes and database constructs that are inaccessible to the businesspeople (i.e., end users) [4, 26].

A business rules system is a computerized system that assists in the management and execution of the business rules logic in running the business of an organization [22]. Business rules represent that part of enterprise knowledge that we can codify [5, 21, 22, 23, 25, 49, 55, 58]. A business rule can be a *term* or a *fact* or a *rule*.

We say a business rules system is understandable and maintainable provided that business rules can be directly managed by the businesspeople of an organization [22, 49]. In doing so, businesspeople have control over the business rules.

Some of the areas of applications of business rules systems are: Payroll, Inventory Control, Order Entry, Order Tracking, and Claims Processing Applications. The formal approach of developing a business rules system is known as business rules approach [22, 49].

1.1.2 DEVELOPMENT APPROACHES' TRADE-OFFS

In the software industry, there are a number of approaches [10, 11, 22, 26, 49] that help us develop software. Two of them are: Object-Oriented Approach (OOA) and Business Rules Approach (BRA). The main advantage of using OOA is its

support for reusability of codes and/or components [10, 47] and as a result understandability and maintainability of code is improved for the developers. The rules of a system are implemented as operations of the respective classes of the system. Data (i.e., attributes) and functions (i.e., operations/methods) are all encapsulated in the respective classes of the system. Modeling languages such as UML are used to visualize, specify, construct and document the artifacts of Object-Oriented (OO) software system [39]. Moreover, programming languages such as C++, SmallTalk, and Java support OOA.

The disadvantages of applying OOA are, first, business rules are embedded or hidden in the procedural codes of the associated classes. Second, as a result of embedment of the business rules in procedural codes, understandability and maintainability of the business rules by the end users and sharing of these business rules to applications are almost impossible to achieve.

On the other hand, the main advantage of BRA is the separation of business rules from the procedural codes. This separation of business rules increases understandability and maintainability of a system by the businesspeople, and sharing the business rules to the participating applications [49]. Rule engines support BRA to automate a business rules system. Rule engines are used for firing or executing appropriate rules stored in a repository system. Versata and ilog commercial products are examples of rule engines.

However, the problems of BRA are, first, UML is not utilized for the fact that it does not adequately address the requirements of business rules system development [22]. Second, no lightweight architectural framework is clearly provided so that developers can easily develop systems using the approach. Moreover, there is a lack of lightweight and integrated support tools for the approach.

1.2 PROBLEM STATEMENT

As indicated above, in using object-oriented approach, business rules are implemented as methods/behaviors and attributes/properties of the respective classes of an application. On the other hand, as proposed by the proponents of the business rules approach, separation of business rules from a procedural code and treating them like data are believed to be the *best* strategy for developing an understandable and maintainable business rules system by the end users [22, 49].

The leading question for this research is then, "how can we develop an understandable and maintainable object-oriented business rules systems?"

In this research, a method is proposed by combining the strengths of object-oriented approach and business rules approach. The method is called **B**usiness **R**ules **O**bject-**O**riented **M**ethod (**BROOM,** for short).

By a method, we mean, a set of principles for selecting and applying a number of analysis and synthesis/construction techniques and tools in order to construct an efficient artifact, business rules software.

Considering the trade-offs of both approaches, we have the following requirements to be satisfied in order to arrive at the proposed method:

1. Treating business rules like data (this has been addressed by the Business Rules Approach)
2. A need for an appropriate object-oriented:
    i. Modeling Language
    ii. Design Patterns/Framework
3. A need for a lightweight integrated support tools for the proposed method.

1.3 RESEARCH QUESTIONS AND HYPOTHESES

Table 1 below depicts the main research questions, sub questions and hypotheses made for the research.

Table 1- Research Question, Sub-Questions and Hypotheses

Leading Question	Sub Question/Problem	Hypothesis
How can we develop an understandable and maintainable object-oriented business rules systems?	What is a feasible and appropriate Object-Oriented Method for understandable and maintainable business rules systems?	A lightweight and rigorous method using the strengths of both object-oriented approach and business rules approach is feasible.
	Does the current UML support for such understandable and maintainable object-oriented business rules systems?	The standard UML does not support modeling of business rules stored in a repository system. For this reason, UML needs to be extended to meet the stated requirement.
	Can we have an appropriate development tool for this method?	We can have a lightweight integrated support tool for the proposed method comprising the tools: Metamodel Management, Rules Management, Data Management, Code Generator, Reverse Engineering, etc.

1.4 OBJECTIVES

The main objective of the research is then to find out and develop the proposed method's components: Models, Architectural Framework, and Support Tools. Figure 1 below depicts the relationships between models, architectural framework, and support tools & techniques that make up the proposed method.

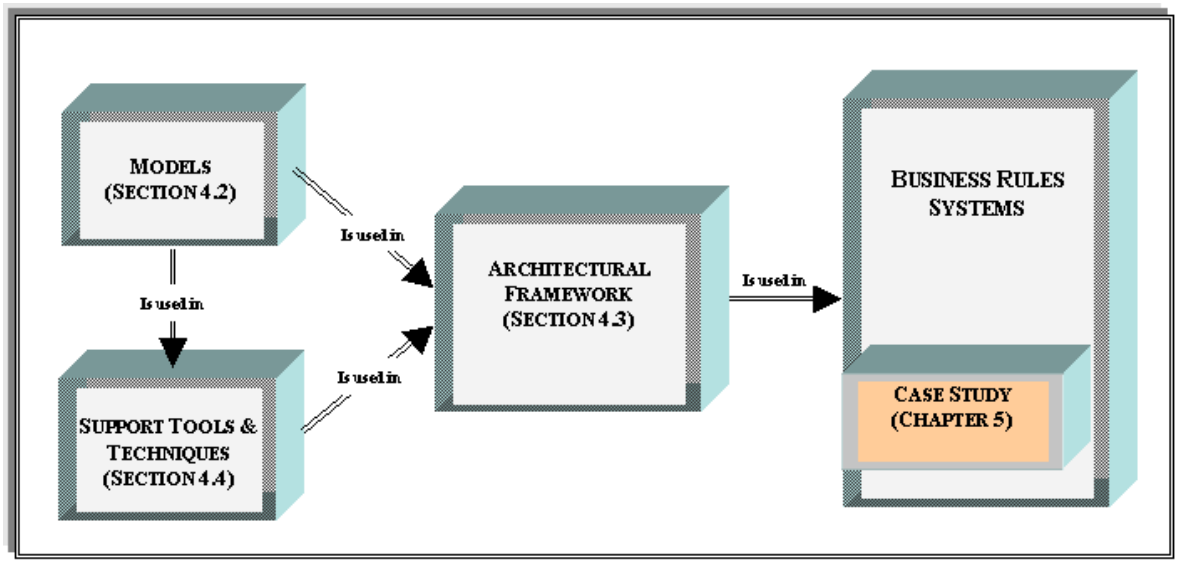


Figure 1: Interactions between Models, Framework, and Support Tools

The following are the detail objectives of the research.

1. To design models such as metadata models, and business rules Metamodel for the development of business rules systems. These models assure separation of business rules from the procedural codes which is the basic principle of BRA and require us to persist business rules in a repository system (Section 4.2).
2. To design an architectural framework for developing a business rules system. The framework allows us to utilize practically proven design patterns and help developers understand and apply the required elements in developing a business rules system (Section 4.3).
3. To develop prototype of the integrated support tools required for the proposed method. These include Metamodel Management Tool, Rules Management Tool, and Data Management Tool (Section 4.4).
4. To demonstrate the proposed method with a case study and evaluate the results obtained (Chapter 5).

1.5 ORGANIZATION OF THE THESIS

The rest of this thesis is organized as follows:

- Chapter 2, Background, gives preliminary information about the following:
  - The two approaches: Business Rules and Object-Oriented Approaches
  - Design Patterns and Frameworks
  - Unified Modeling Language (UML)
- Chapter 3, Related Work, reviews the state-of-the-art relevant to this thesis. These include UML extension mechanisms, modeling design patterns, persistency of objects, rule patterns, and required tools for developing a software system.
- Chapter 4, Development of BROOM, provides solutions that are believed to solve the problems stated in Chapter 1, focusing on business rules systems application development. In this chapter, the following activities are performed:
  - Extend UML Class Diagrams to support separation of business rules from the procedural codes and database constructs
  - Develop architectural framework for the proposed method – BROOM
  - Provide the implementation perspectives and settings of the method by selecting programming languages, a repository system, and other development tools
  - Develop prototype of the integrated support tools for the method
- Chapter 5, Case Study, presents a demonstration or validation work of the results of the research by selecting a case study, applying the solutions developed in Chapter 4, and analyzing the results of the case study.
- Chapter 6, Conclusions, presents some concluding remarks, list of the contributions made by the research, and the future work relevant to the research.
- Glossary, presents contextual definitions of important terms used in this thesis.

CHAPTER 2

BACKGROUND

2.1 BUSINESS RULES APPROACH

Business Rules Approach is a formal way of managing and automating the business rules of a business rules system so that the business behaves and evolves as its leaders intend [22, 49].

According to [22, 31, 49, 58], business rules approach proponents such as Ross [49] and Halle [22] propose the basic steps towards developing a maintainable and understandable business rules systems. These steps focus on separating the business rules from any code of an application or active database management system (DBMS) constructs such as triggers, which are standardized in Structured Query Language-3 (SQL-3) [27]. In other words, the business rules are treated like data of a system in a DBMS. Besides, these proponents comment that the current object-oriented methods do not support this separation of business rules [22].

2.2 OBJECT-ORIENTED APPROACH

The main goal of object-oriented approach is reusability and flexibility for developing object-oriented software systems [10, 47]. However, reusability and flexibility issues directly focus on the programmers' productivity. Moreover, the arguments for object-oriented approach and separation of business rules from the procedural codes can be twofold: First, the current modeling tools such as the Unified Modeling Language (UML) do not support modeling of a class containing a set of business rules that are separated from the procedural codes and stored in a repository system like a DBMS. Second, object-oriented programming languages do not adequately address separation of rules from the procedural codes [22].

2.3 UNIFIED MODELING LANGUAGE (UML)

2.3.1 ABOUT UML

Unified Modeling Language™ (UML®) is a standard modeling language used for visualizing, specifying, constructing, and documenting the artifices of a software-intensive system [8, 50]. UML has gained significant industry support from various organizations via the UML Partners Consortium and approved by the Object Management Group (OMG) as a standard in November 17, 1997 [39]. When this thesis was being written, UML was on the verge of version 2.0 [36, 39]. A summary of the major evolutions of UML is presented in Figure 2 below.

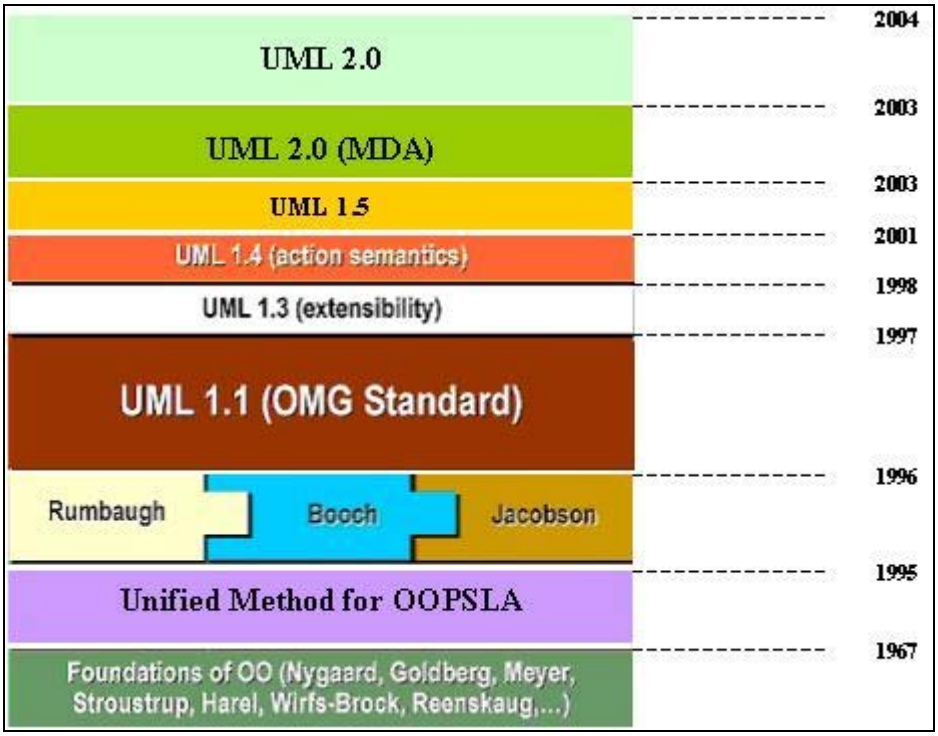


Figure 2: A summary of the major Evolution of UML [44]

2.3.2 ARCHITECTURE OF UML

To facilitate the description of the UML, it is important to understand the architecture that the OMG has chosen for its standards. The architecture has four layers, called M0, M1, M2 and M3 [37, 60].

Figure 3 below depicts the relationships between these levels or layers with the corresponding examples.

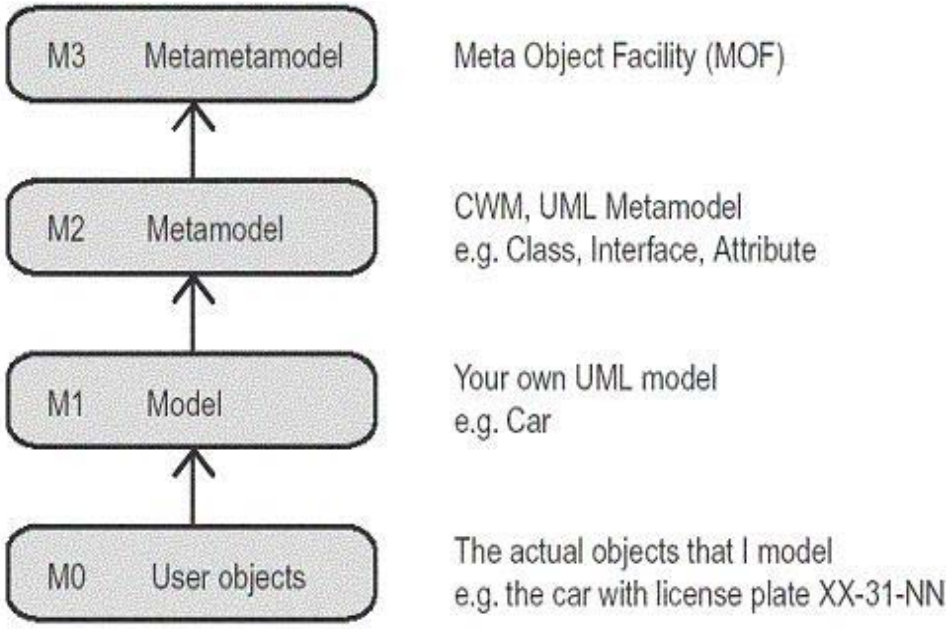


Figure 3: OMG UML 4-Layer Architecture [60]

An instance at a certain level is always an instance of something defined at one level higher. An actual runtime object at M0 is an instance of a class defined at M1. The classes defined in UML models at M1 are instances of the concept Class defined at M2. The UML metamodel itself is an instance of M3. Other meta-models that define other modeling languages are also instances of M3. Common data Warehouse Model (CWM) is one example.

2.4 DESIGN PATTERNS AND FRAMEWORKS

2.4.1 ABOUT DESIGN PATTERNS AND FRAMEWORKS

Design patterns are a form of best-practices documentation and not a language feature or idioms [18, 19]. Their intent is to communicate the observations, analysis, and recommendations of experienced developers with others in an effective way [18]. The authors of [18] are sometimes referred to Gang Of Four

(GOF). The twenty-three design patterns presented in the work of GOF are known as the Standard Design Patterns.

Despite the fact that design patterns and frameworks are related, it is important to recognize that frameworks and design patterns are two distinctly separate things: both of them are abstract designs that solve a given problem. Design patterns may be used in both the design and the documentation of a framework [18]. A framework is a highly reusable design for an application, or part of an application, in a certain domain and often defines the basic architecture of the applications that can be built by using it [33]. In fact, a framework can be viewed as the implementation of a system of design patterns.

2.4.2 SELECTING DESIGN PATTERNS

There are two ways of finding design patterns for a particular problem of a given domain: Problem-oriented and Solution-oriented[1] [65]. As indicated by Coplien [12], the sources of design patterns are numerous articles and papers in published forms and in the trade press. For a designer, it might be very difficult to select and refer to all the sources of design patterns. This is because there is no concerned body assisting in collecting proven design patterns in a standard way [15]. To that end, Tichy [56] presents over 100 general-purpose software design patterns categorized into nine top-level categories. Apart from its good collection and categorization of the design patterns, this work is merely used as a quick reference for the software designer searching for a design pattern. This is so because the focus is on cataloging the design patterns (the problems they solve).

[1] *Problem-oriented*: This mechanism first identifies the general design problems or structures like creating / decoupling objects or realizing hierarchical structures. Afterwards, an appropriate solution to this problem is developed. The design problem is ideally addressed by an existing design pattern, so that a well-designed solution can be reused. *Solution-oriented*: This focuses on the current design and implementation (solution), that is, it analyzes the existing design and implementation in order to find out where and how to improve it.

Similarly, David et al. [32] present the problem of finding or selecting design patterns for a specific design problem. This is because of the learning curve required for design patterns by software designers. For this issue, refer to the stages of awareness of design patterns presented in [59]. The lack of CASE tools supporting for the selection process of design patterns is also another reason for the problem. David et al. [32] also present a methodology on how to select design patterns for the specific design problem in addition to the guidelines, given in the work of GOF [18].

2.4.3 DOMAIN SPECIFIC DESIGN PATTERNS

GOF's work concentrates on object-oriented design problems and object-oriented programming languages such as C++ and SmallTalk. That means, it does not adequately address the problems of non-OO design. Moreover, the work of GOF does not address domain specific design problems such as separation of business rules from the procedural codes. It is intended to serve as a solution for general-purpose design problems.

However, after the work of GOF, some efforts were made to consider domain specific design patterns [16, 20, 53]. For example, Gustavsson et al. [20] attempted to propose a method for this need so as to assist developers pick design patterns for their specific domain design problems.

Ekstrom [16] discovers design patterns for simulation software. This is an indication of the need for domain specific design patterns. Gustavsson et al. show the need for domain specific design patterns, database for collecting them and a standardization body for proving their uses and precise specifications [15].

2.5 SUMMARY

The use of modeling language such as UML and reuse of codes are some of the advantages of using OOA. On the other hand, separation of business rules from a procedural codes and database constructs is the main principle of BRA [22]. This principle helps the businesspeople manage their own business rules themselves without the need for direct involvement of IT professionals [22, 49]. Consequently, the IT infrastructure becomes more stable, reliable, and productive in advancing its architecture and functionality [22, 49].

In general, as presented earlier, both business rules approach and object-oriented approach have their own strengths and weaknesses.

CHAPTER 3

RELATED WORK

3.1 UML EXTENSION MECHANISMS

Modeling business rules object-oriented systems did not find a direct support in the standard UML notations [38]. To that end, the work of Hay [24] presents metamodels for business rules artifacts using entity relation diagram (ERD). However, whereas metamodeling of business rules elements such as terms are useful, it does not give any idea on how to apply OO approach, or UML.

UML incorporates, in an integrated way, several notations for object-oriented modeling [8, 46, 47, 48, 50], although these notations are not enough for building business rules systems where business rules are stored in a repository system [14, 25, 38].

Though UML provides a rich set of notations, it is also intended to be extensible, adaptable, and modifiable [8, 46, 48, 50]. In other words, UML provides refining mechanisms of the standard semantics in a strictly additive manner provided that additional semantics do not contradict or conflict with the standard semantics. These mechanisms are called UML Extension Mechanisms and are used to capture semantics that are significant outside the scope of the language.

Prior to UML 2.0, the idea of extending the UML was limited to stereotypes and profiles [8, 41, 50, 52]. Stereotypes are localized extensions that take the form of adding adornments to existing modeling elements. Profiles, on the other hand, are the mechanisms, which were envisioned for larger scale changes, such as the addition of Specification and Description Language (SDL) [39].

In UML 2.0, there is a new extension mechanism called the Metamodel [37]. A metamodel is the underlying mechanism that defines the language for expressing a model (see M2 in Figure 3 above). Thus, in UML 2.0, there are three UML extension mechanisms to consider when choosing to extend the UML: Stereotypes, Profiles and Metamodel.

3.2 MODELING DESIGN PATTERNS

Design patterns are modeled or represented using UML. However, the standard UML does not adequately address the special aspects of a design pattern such as the variability of an algorithm in strategy design pattern. Fontoura et al. [17] propose an extension of UML using UML Extension Mechanisms such as stereotypes, which can identify the hot-spots, which are the aspects of the pattern that may vary.

The work of Fontoura et al. [17] focus on most of the standard design patterns called *configuration patterns.* This group of patterns is used for making systems more flexible and extensible. This same work also proposes two implementation techniques through Aspect-Oriented Programming (AOP) and Subject-Oriented Programming (SOP).

3.3 PERSISTENCY USING DESIGN PATTERNS

Design patterns can play different roles in different layers/tiers (i.e., presentation layer, business layer, and persistence layer) of a system development. For example, [2, 30, 51, 61, 62] demonstrate on how to apply design patterns to persistence layer and object persistency of a system. Persistence layer is a special case of building a layer to protect one from changes in the application or in the database [2, 51, 62]. Nevertheless, persistency of objects is one of the least

addressed issues in object-oriented programming until the work of some researchers such as Yoder et al. [62] and Kienzle et al. [30]. For example, Kienzle et al. use the standard design patterns such as Serializer, Factory Method, and Strategy to produce a framework for persisting objects. The work of Sauer [51] is typically a lightweight framework for object relational mapping to Java Database Connectivity (JDBC). Sauer uses a number of standard design patterns such as Template, Strategy, Command, and Composite. This same work uses non-OO Relational DBMS (RDBMS) for its persistency mechanism.

The work of Yoder et al. [62] is a pattern catalogue for connecting business objects to RDBMS (i.e., on how to persist objects to RDBMS). The paper selected RDBMS for a number of reasons: systems often need to store objects in RDBMS; systems often need the maturity of RDBMS; and other times, the corporate policy is to use RDBMS.

Ambler [2] also presents how to design persistence layer avoiding hard coding SQL in business classes since such an act results in code that is difficult to maintain and extend. According to this work, even hard coding SQL in separate data classes or stored procedures is only slightly better. Ambler recommends a persistence layer that maps objects to persistence mechanism, which the author calls it robust persistence layer.

A framework for relational database access layers is also provided in [61]. This same work is similar to the work of Sauer but the framework of [61] can be used for both OO and non-OO programming languages.

3.4 AUTOMATING DESIGN PATTERNS

The transformation of design patterns to an implementation language is sometimes difficult, and time consuming. Mel [35] provides a methodology for transforming design patterns into a particular programming language codes. However, precise specification of design patterns is a prerequisite to their analysis and the automation of their implementation [6]. Jaczynski et al. [28] use design patterns such as Abstract Factory, Composite, and Strategy with Rational Rose to design Case-Based Reasoners and automate same in Java code.

For our research, some of these works in transforming design patterns to implementation can be used in developing the support tools of BROOM.

3.5 RULE PATTERNS

To our knowledge, the work of Kappel et al. [29] is the first one to come up with a new concept called *rule patterns* in mid 1990s by making analogy to the concept of design patterns presented by GOF [18]. The same work introduces the general notion of rule patterns and illustrates the approach with an example. The paper critically comments on OO approach by stating, “coming from the programming and not from the database community, they do not consider the database aspects such as persistence of objects”. Though the work of Kappel et al. is a good start on business rules design problem, it is difficult to understand for it does not use any graphical representation to model the concepts and assumes that object-oriented approach does not solve the problem. That is, it is difficult to adapt these rule patterns for BROOM.

The works of Arsanjani [3] and Yoder et al. [63] are the latest researches in the area of rules patterns that serve the purpose of separation of business rules from

the procedural codes. Arsanjani and Yoder et al. provide Rule Object and Adaptive Object Models (AOM) respectively. AOM is related to Model Driven Architecture (MDA) of OMG [40]. These two works are based on the standard design patterns and are proven in various practical applications. For example, most of the design patterns of [63] are the standard ones (i.e., Type-Object, Properties, Composite, Strategy, Interpreter, and Builder).

Arsanjani [3] supported his work by presenting known application areas such as IBM San Francisco, If-Then-Else Framework [13], IBM WebSphere Application Server Enterprise Edition, and Component Broker's Manager Object Framework.

Implicitly, however, both Arsanjani and Yoder et al. assume that the standard[2] UML is adequate for representing the models of the pattern language. To that end, the work of Fountoura et al. [17] presents an extension of UML for better representation of the configuration design patterns.

3.6 REQUIRED TOOLS

This section presents a review of the available technologies such as repository systems, rules engines, and other support tools assisting the development of a business rules system with the very intent of separating business rules from the procedural codes and database constructs.

3.6.1 REPOSITORY SYSTEMS

Persistence of objects is not very well addressed in the OOA even though object concepts have been around since 1967 [44]. However, it has been worked in, for example, Object Relational DBMS (ORDBMS), which is a marriage of OOP and database technologies [1, 34, 62] and persistency of object less its operations (i.e.,

[2] Here, the term *standard* stands for the original OMG UML 2.0 Diagrams, before extended/modified

rules) has been addressed. Nevertheless, ORDBMS does not solve the problem of our research because it does not help us separate the objects (i.e., business rules) from the database constructs and *rules/methods* are not separated from codes. In this research work, however, RDBMS for storing the Metamodel and data of a business rules system is utilized.

Business rules can be separated from an application code with a rule engine design pattern presented by Boglaev [7]. In this work, a lightweight and less costly rules engine is produced using XML as its data repository and XSL as a rule engine functionality storing all the refactored business rules from an application code of a system. However, we need to refrain ourselves from using XML as a database for storing the huge amount of data of a system for its performance and other data processing maturity problems. This is because, mainly, XML is created for alleviating interoperability problems of organizations' data such as in importing or exporting data from/to a legacy DBMS of an organization [9].

3.6.2 RULES ENGINES

A rule engine is a key underlying technology that examines rules stored in a repository system, and seeks solutions consistent with the rules [6]. It comes from the Artificial Intelligence (AI) field of knowledge engineering [7]. A rule engine may be viewed as a collection of rules such as if/then statement interpreter.

There are a number of commercial rules engines in the market. Some of them are ilog, versata, Advisor, Haley's, Jess, and CommonRules. There are also some open source software such as CLIPS, Drools, Mandarax for building them. However, these are heavyweight and sometimes have high hidden costs for a business rules system development.

As indicated in the Repository Systems Section of this review, Java, XML and XSL are used for developing a simple and lightweight rule engine enabling us to separate business rules from the procedural code of an application [7]. XSL is originally created for transforming XML document into a formatted output such as HTML document [43]. Boglaev [7] does not use XSL as it was intended to be used by its architects. Instead, it is used as a repository system for the business rules separated from an application code.

Another possibility for having a lightweight rule engine is to use application programming interface (API) specification for a Java based rule engine, which is sometimes referred to as JSR-94 [6]. It is to be used for developers developing their rule engines enabling them interoperate one another.

3.6.3 INTEGRATED SUPPORT TOOLS

The idea of integrating the support tools for our proposed method was borrowed from work of Paige et al. [42]. Yet, BROOM differs for the following facts:

1. It focuses on business rules systems development by separating business rules from the procedural codes and database constructs.
2. The integrated support tools that can be used for developing business rules systems consist of the following:
    a. Business Rules Management Tool
        i. Metamodel Management Tool
        ii. Data Management Tool
        iii. Rule Management Tool
        iv. Report Generation Tool
    b. Reverse Engineering Tool. (For example, a tool that enables us to model the class diagrams using the extended UML class diagrams based on the Metamodel information stored in repository systems).
    c. etc.

3.5 SUMMARY

The standard UML does not support modeling business rules system with the intention of separating business rules from procedural codes. Thus, extension of the standard UML is required for modeling the artifacts of a business rule system in order to visualize, specify, construct and document them in an efficient and effective way.

The works of Arsanjani [3], Boglaev [7], and Yoder et al. [63] are the main candidates of the proposed method with regard to the separation of business rules.

Moreover, the proposed method also requires support tools in an integrated manner in order to facilitate the development of business rules systems [57]. The following are the potential candidates for the required tools & techniques: Java as the programming language; RDBMS as a storage mechanism to store business rules Metamodel and data that may be of huge amount; XML as an intermediate data place for validation or firing rules pertaining to that data using a lightweight rule engine for a business rules system development as demonstrated in [7].

Finally, the potential implementation strategy for a lightweight rule engine of a business rules system is either to use Boglaev's Design Pattern for a Rule Engine or use the JSR-94 (Rule Engine API). Both of them use Java as a programming language; Boglaev uses XML to store data that trigger the business rules stored in a repository system - XSL.

CHAPTER 4

DEVELOPMENT OF BROOM

4.1 INTRODUCTION

BROOM attempts to combine the strengths of the two approaches: Business Rules and Object-Oriented Approaches with a modeling language, an appropriate framework, and integrated support tools. It is supposed to be a lightweight software development method for building object-oriented business rules systems. Simply put, a business rules system is an automated system in which the "rules" are separated (logically, perhaps physically) and shared across data stores, user interfaces and applications [4, 22, 49].

The next sections are devoted to the development of the different components of the proposed method – an extension of the UML, framework, and support tools & techniques.

4.2 EXTENDING UML FOR BROOM

To extend UML, data modeling techniques are used for the fact that persistency of the business rules is the main concern of the method under development. That is, our strategy in modeling business rules for the proposed method is first to look at the business rules repository system level so as to come up with an extended UML class diagrams that helps us model the artifacts of a business rules software system.

4.2.1 EXTENDING UML

This process of extending UML is done after proper analysis of UML class diagram for its necessity, sufficiency, and consistency is carried out. The technique uses stereotyping extensively for modeling business rule via its flavors: **Terms, Facts**

and **Rules**. Ultimately, these models are systematically integrated to form or develop a Metamodel for BROOM.

The main reasons for choosing stereotype extension mechanism are it is lightweight and the fact that UML class diagrams do not require large-scale changes. In addition, we want to keep the interoperability with the UML Tools such as Rational Rose that can support user defined stereotyping. If a heavyweight extension mechanism is used, a developer may be forced to develop a modeling tool supporting the extended features from the standard UML. This is because modeling tools' vendors may not incorporate the UML extensions made by the developer to their tools.

4.2.2 UML CLASS DIAGRAM ANALYSIS FOR SUPPORTING BROOM

A class is a general concept having structural and behavioral features. Structural features, including attributes and relationships called associations, define what elements constitute the class [8, 41, 50]. An attribute is an element of information or data. A relationship is a connection between classes. Behavioral features, including operations and methods, define how the constituents of the class interact to provide the behavior of the class. An operation is a specification of a service or processing. A method is an implementation of a service or processing. The most crucial aspect of a class is its semantics.

Class diagrams are most familiar and popular diagrams in UML. They depict the static structure and behavior of a system. UML class diagrams can depict the attributes, operations, and relationships quite easily. The fundamental element of a class diagram is an icon that represents a class with three default compartments (i.e., Class Name, Attributes List, Operations List compartments).

Compared to UML 1.5 Class diagrams, not much has been changed in UML 2.0 Class diagrams. For a summary of the changes made, refer to [36]. Therefore, it suffices to check the necessity, sufficiency and consistency of UML 1.5 instead of UML 2.0 for our proposed method.

In modeling a system with the standard UML class diagram, it is well known that operations are modeled as though they are more stable or static (may not be changed without programmers interventions) part of the system. Furthermore, unlike attributes' values of a class, operations are also assumed to be stored/buried/embedded as methods in the implementation of OOP languages that are inaccessible for the businesspeople (i.e., end users of the system). However, the usual OO implementation reveals that values of an attribute are stored in a repository system such as RDBMS and these are the only dynamically changeable things by the businesspeople.

Therefore, at least, we need to devise a modeling mechanism for those changing business rules (including newly defined or deleted ones). For example, using a stereotype UML extension mechanism, the operations compartment of a class can be divided into two parts: «Dynamic Operations» and **«**Persistence Operations**»**. Note that **«** and **»** are single characters called left and right Guillemets respectively. Not all the operations are required to be changed by the businesspeople.

Thus, the usual modeling technique can still be applied for those persistence operations, which are static/stable by nature (i.e., the persistence layer's operations are stable). This is handled by the **«**Persistence Operations**»**. The dynamic part that is subject to change and can be managed by the businesspeople is modeled by **«**Dynamic Operations**»**.

For example, Save(), Delete(), and Get() database operations for an entity class of an application are considered to be «Persistence Operations». The reason is, they are rarely changed throughout the lifecycle of the application. These are common to most entity classes. Thus, a persistence layer class along with the necessary operations can be used to implement them [2, 51]. Whereas Tax and Pension calculations of the employee class are subject to change for various reasons. Hence, the corresponding operations in the class are said to be «Dynamic Operations».

Moreover, in a business rules system, not only the values of the attributes stored in a repository system are subject to be changed but also the characteristics of the attributes (for example, the *size* of an attribute) can be altered. Similar to an operation of a class, the characteristics of an attribute (not its values) can also be modified. Therefore, we are faced with the same problem that is encountered in modeling an operation of a class in a business rules system. Consequently, to indicate the dynamicity of the attributes, the Attributes compartment of a class requires to be represented by «Dynamic Attributes». Ideally, all the attributes of business rules are subject to change. In other words, all attributes are members of the «Dynamic Attributes».

For example, the size of an attribute, say, *FullName* of an employee class may change from 50 to 25 characters or changing the name from *FullName* to *LastName* may be mandatory. In other words, the attribute, which was originally named as *FullName* with size 50 characters, is considered to be a member of «Dynamic Attributes».

In a nutshell, the above analysis shows that the standard UML class diagram notation is insufficient to model the class (Attributes and Operations) of a business rules system since class diagram can only represent the static structures and behaviors of a system. Now, the main question is, can we use the UML class diagram for representing or modeling the dynamic structure and behaviors of a system? It is most probable that the answer to this question is yes. This is because UML, in general, class diagram, in particular, can be extended using the UML extension mechanisms so as to meet the modeling requirements of our domain – business rules systems. Note that this is one of the hypotheses made for the research.

4.2.3 MODELING THE FLAVORS OF BUSINESS RULES

One of the basic ingredients of the business requirements is the existence of a set of continually changing business rules. Moreover, based on the intent of the business rule statement [22, 49], a business rule can:

(1) Define a **Term**

(2) Connect **Terms** into meaningful **Facts**

(3) Produce the result of a Calculation,

(4) Test Conditions to produce a new fact, or } **Rules**

(5) Test a condition to initiate action.

The last three categories are called **Rules**. In other words, business rules are each one of the following kinds or flavors: **Terms** (definitions), **Facts** (connections), or **Rules** (calculations, constraints, conditional logic). In our context, the word “rules” is not equivalent to the phrase “business rules” (i.e., rules are one kind of business rules).

Terms and facts along with the derived rules form the main ingredients of business rules models. In the remainder of this section, we discuss issues of modeling business rules using UML class diagram after extending per the analysis results obtained in Section 4.2.2 of this thesis.

The following are the main assumptions that are made in modeling terms, facts and rules of a business rule:

1. The main assumption in modeling a term, fact or rule of a business rule is that *all* of them are stored in a repository system. Actually, this follows from the very principle of business rules approach: business rules should be separated from the procedural codes or database constructs and we treat them like we do for data.
2. The business rules are changing (i.e., new terms, facts or rules arrive, existing terms, facts or rules expire, and so on).
3. These changes will be managed by the businesspeople of an organization without IT professionals' direct interventions.

4.2.3.1 MODELING THE TERMS

A term is a basic word or phrase in English or another natural language that people recognize and share in the business [22]. In a system, at a time, a term represents either an entity or an attribute. Terms are always nouns or qualified nouns. Here are some examples:

Customer	Order	Manager
Employee	Salary	Income Tax
Employee Name	Aircraft	Airport

Traditionally, we can model Terms as Entities and Attributes [22, 49]. Consequently, object-oriented modeling of Terms is equivalent to building Entity Classes for the Terms of the business rules (it follows from assumption 1 above). This can be done by extending a class diagram to indicate the dynamicity of business rules as shown in Figure 4 below.

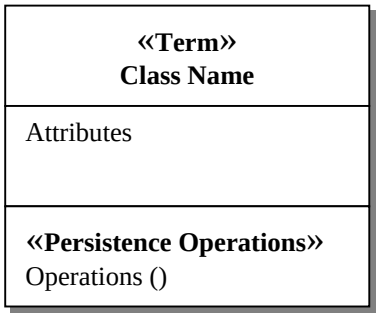


Figure 4: Modeling a Term at First Glance

In the above figure, «Term» and «Persistence Operations» are Stereotypes. Unfortunately, the above proposed model cannot serve for our purpose since it violates the assumptions set forth in the beginning of this section (especially, assumption 2 and 3).

Note that the standard UML class diagram depicts the static structures and behaviors of a system [8, 41, 47, 50]. Thus, they are not suitable for modeling the dynamic structures and behaviors of the business rules – in our case, terms as they stand. In other words, since terms are supposed to be managed by all users, including the businesspeople of the system, the above model is incapable of addressing the changes encountered in the terms of a business rule system.

Figure 5 below depicts the relationships among business rules system, terms, and a business rules repository system.

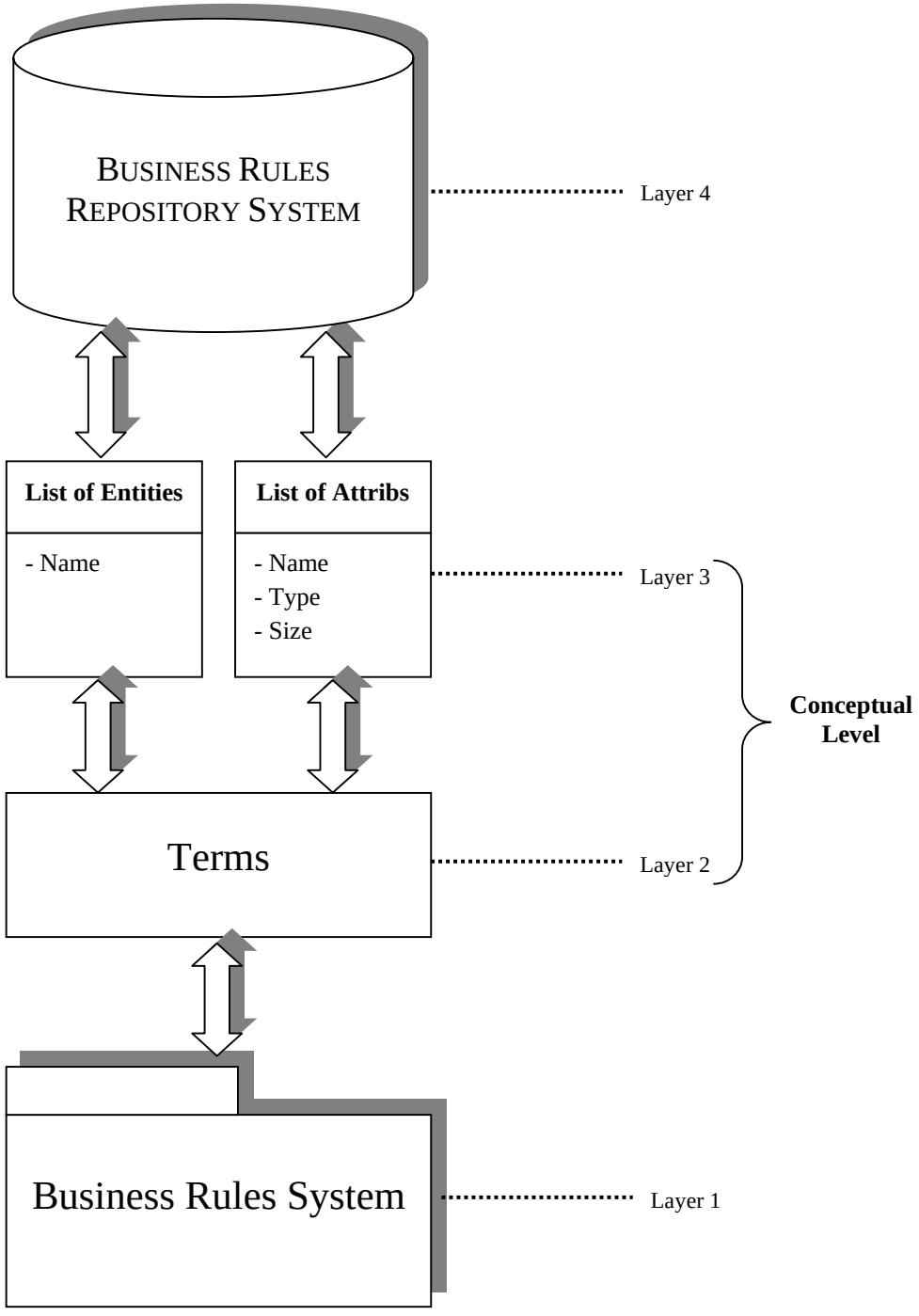


Figure 5: Terms and a Business Rules Repository System[3]

[3] Business rules repository system is a repository system whose purpose is to record and manage the information about the data models and the designs and implementations of the business rules systems of an organization [25].

The question is then, how can we model Terms of a business rules system in an object-oriented way using UML? Terms (Layer 2) in a business rules system (Layer 1) can be modeled by considering its components (Layer 3) - *List of Entities* and *List of Attributes* of the system along with the characteristics of these components.

TERMS DATA MODEL (TDM)

As proposed in the above section, to model the terms of a business rules system, the components (Entities and Attributes) found in Layer 3 of Figure 5 above are considered. Accordingly, there are two data models for relational database tables: (*Cf.* the business rules metadata model presented in Section 4.4.5.2 of this thesis)

Table 2: List of Entities Data Model Design

No.	Field	Remark
1.	TermID	This is a Primary Key
2.	Description	Describes the Entity

Table 3: List of Attributes Data Model Design

No.	Field	Remark
1.	TermID	This is a Primary Key.
2.	Description	Any description of the attribute.
3.	Type	Attribute type such as Text, Integer, Boolean, Date, and so on.
4.	Size	The size of an attribute (applies to text type).
5.	IsDervied	Shows if an attribute is derived one. (It is a Yes/No type).
6.	Default Value	Attribute's default value, if any.

EXTENDED CLASS DIAGRAM DEPICTING TDM

In order to model the TDM using the UML class diagram, a stereotyped entity class that has dynamic attributes and static operations for persisting the terms is designed. In effect, this process extends the standard UML class diagram using stereotype extension mechanism. Figure 6 below depicts the extended UML class diagram for the collection of terms of a business rules system.

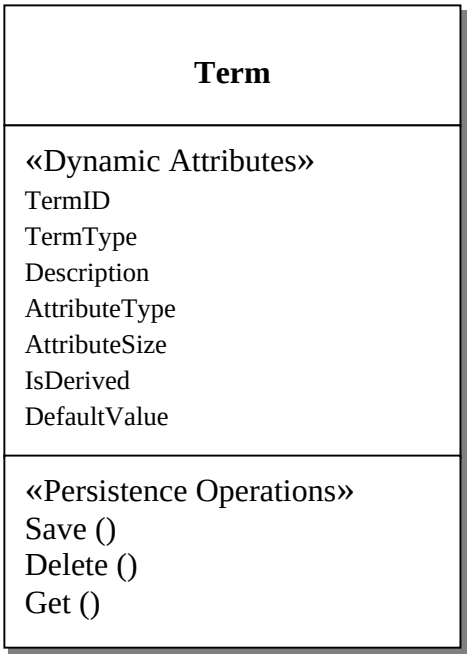


Figure 6: Extended UML Class Diagram for Terms

Table 4 below describes the stereotypes used to model the Terms Data Model.

Table 4: Description of the Extensions used for TDM

No.	Field	Remark
1.	«Dynamic Attributes»	This represents the union of all the attributes specified in the Terms Data Model presented above. TermType is either “Attribute” or “Entity”. TermType is dynamic in a sense that an attribute may become an entity or vice versa.

No.	Field	Remark
2.	«Persistence Operations»	As listed above (in the extended UML Terms class diagram), this stereotype represents all the default operations (save, get, delete) that can be used for managing the terms constituents data in a repository system, as an entity class.

REMARKS

1. The extended UML class diagram shown above is a template for the collection of terms in a business rules system.
2. The extended UML class diagram for Terms (entities and attributes) and the *values* of the attributes are managed by two of the Business Rules Management Tools known as Metamodel Management Tool (MMT) and Data Management Tool (DMT) respectively.

4.2.3.2 MODELING THE FACTS

Facts are about relationships or associations among the other flavors of business rules. The relationships between the pairs (X, Y) of terms of the business rules are classified into the following categories: “X Related To Y” means two tables are Related To one another. “X Contains Y” means containment fact (i.e., an entity contains attribute(s)).

Relationship/fact has been extensively addressed by the relational modeling technique with an extension (profile) of the standard UML [2] except that in the business rules approach the facts are stored in a repository system.

Figure 7 below shows the relationships among business rules system, facts, and a business rules repository system.

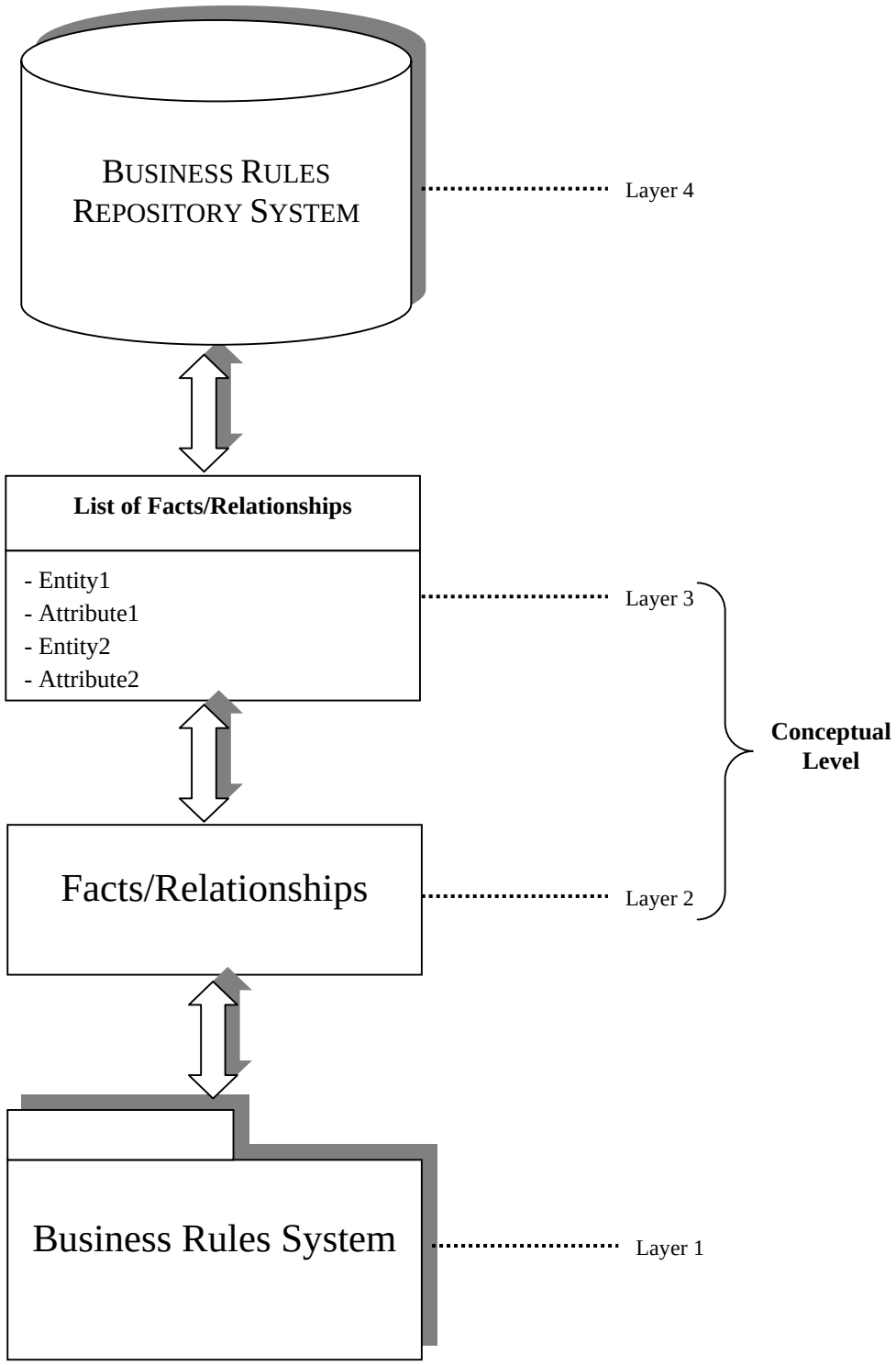


Figure 7: Facts and Business Rules Repository System

The question is then, how can we model facts/relationships of a business rules system in an object-oriented way using UML? Facts (Layer 2) in a business rules system (Layer 1) can be modeled by considering its component (Layer 3) - *List of Facts/Relationships* of a system along with the characteristics or properties of this component -Entity1, Attribute1, Entity2, and Attribute2. This handles containment of attributes by an entity as well as the relationships between two tables.

FACTS DATA MODEL (FDM)

Facts can be modeled as Relationships [22, 49] of the Entities and Attributes that are modeled for the terms. Thus, to model the facts of a business rule, we need to model its components shown in Figure 7 above. Table 5 below presents the data model for the collection of facts in business rules system.

Table 5: List of Facts Data Model Design

No.	Field	Remark
1.	FactID	This is a Primary Key.
2.	Entity1	The first entity where a relation starts.
3.	Attribute1	**For Related To**: A key of the Entity1 by which it is related to the Entity2 with a key of Attribute2. **For Containment:** This may also represent that Attribute1 is contained in Entity1. In that case, both Entity2 and Attribute2 are empty string.
4.	Entity2	A second entity which is related to the Entity1.
5.	Attribute2	Similar to 3 above.
6.	IsUnique	For **containment** relationship, an attribute (attribute1) of Entity1 may be unique or repeating. If IsUnique is **True,** Attribute1 is unique in Entity1.

EXTENDED CLASS DIAGRAM DEPICTING FDM

To represent the FDM using the UML class diagram, we need to develop a stereotyped entity class that has the above data model fields as dynamic attributes, and the default static operation (i.e., «Persistence Operations») for managing the facts constituent data of a business rule of a system in a repository system.

Figure 8 below depicts an extended UML class diagram that represents the collection of Facts of a business rules system.

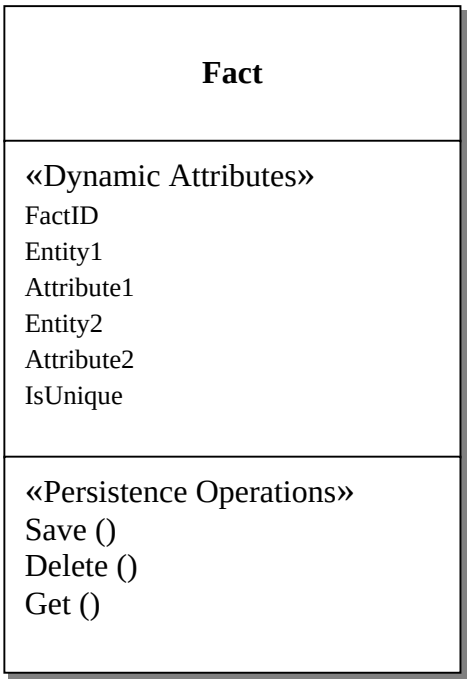


Figure 8: Extended UML class diagram for Facts

Table 6 below briefly describes the stereotypes used to extend the UML class diagram for Facts Data Model.

Table 6: Description of the Extensions used for FDM

No.	Field	Remark
1.	«Dynamic Attributes»	These are all the fields found in the above entity data models for the list of facts (Table 5), which can be changed by the user of the system.
2.	«Persistence Operations»	As an entity class (i.e., business rules should persist), it requires to have the following default methods such as Save (), Delete (), and Get () operations for managing facts data in a business rules repository system.

REMARKS

1. The «Dynamic Attributes» can be treated as built-in structures of the class for a business rules system.
2. Metamodel Management Tool (MMT) handles the relationships/facts concept of a business rules system.

4.2.3.3 MODELING RULES

Rules are elements or flavors of business rules that can be expressed as calculations, constraints, or conditional logic. In other words, rules can:

1. Produce the result of a Calculation,
2. Test Conditions to produce a new fact, or
3. Test a condition to initiate action.

A similar but more formal classification of rules can be found in the next table (Table 7). As presented in the table, there are five types or categories of rules [22]: *Constraints*, *Guidelines*, *Action Enablers*, *Computations* and *Inferences*.

Table 7: Types of Rules

S. No	Rules Type	Remark
1.	Constraints	A *mandatory* restriction or suggested restriction on the behavior of the business.
2.	Guidelines	A complete statement that expresses a *warning* about circumstance that should be true or not true allowing the human to make the decision.
3.	Action Enablers	A complete statement that *tests conditions* upon finding them true, *initiates* another business event, message or *other activity*.
4.	Computations	A complete statement that provides an algorithm for arriving at the value of a term where such algorithms may include sum, difference, product, quotient, count, maximum, minimum, average.
5.	Inferences	A complete statement that tests conditions and upon finding them true, establishes the truth of a new fact.

EXAMPLES

1. Guideline: The maximum basic salary of an employee **should** be Birr 10, 000.00.
2. Constraint: The minimum basic salary of an employee **must** be Birr 200.00
3. Action Enabler: If an employee has a monthly income of less than Birr 200.00 then, display “The Employee is exempted from an income tax.”
4. Computation: Pension contribution of an employee is calculated as 4 % of the basic salary of the employee.
5. Inference: If an employee’s monthly income is less than Birr 200.00, then s/he is exempted from an income tax.

Figure 9 below presents the relationships among business rules system, rules, and a business rules repository system.

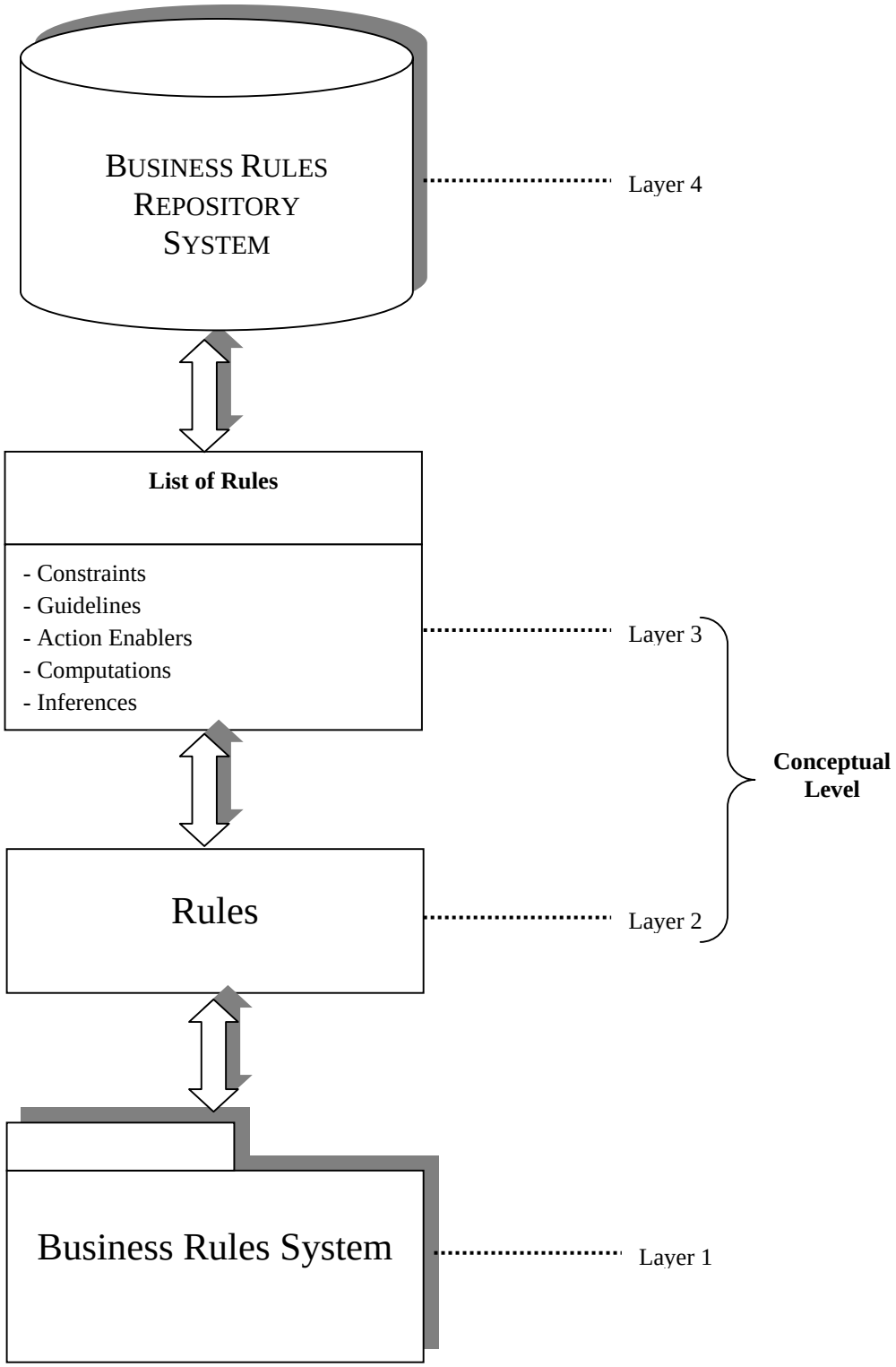


Figure 9: Rules and a Business Rules Repository System

The next and an important question is then, how can we model rules of a business rules system in an object-oriented way using UML? Rules (Layer 2) in a business rules system (Layer 1) can be modeled by considering its constituents (Layer 3) - *List of Rules* of a business rules system: Constraints, Guidelines, and so on. In this thesis, inferences are not treated for the fact that they require more analysis in the area of Inference Engine.

RULES DATA MODEL (RDM)

Referring to Figure 9 above, rules can be modeled by using data models of the rules types. Table 8 below depicts the data models for Constraints and Guidelines (MUST and SHOULD, respectively), normalized into one table.

Table 8: Constraint and Guideline Data Model Design

No.	Field	Remark
1.	RuleID	This is a Primary Key.
2.	Description	Describes a rule.
3.	Severity	A value representing an Error (for Constraint) or Warning (for Guidelines).
4.	Rule Expression	User understandable and maintainable rule in the form of a guideline or a constraint. The expression is composed of the restricted English language vocabularies.
5.	Message	The message that should be flagged to the user when the rule is violated. It is a free text.

Table 9 below presents the data models for Action Enabler in a business rules repository system.

Table 9: Action Enabler Data Model Design

No.	Field	Remark
1.	RuleID	Primary Key.
2.	Description	Describes a rule - Action Enabler.
3.	Rule Expression	User understandable and maintainable rule that enables us to parse an action enabler expression, which is understood by the repository system. The Rules Management Tool with restricted English language vocabularies manages this expression. This expression may refer to already defined rules.

Table 10 below gives the data models for Computations in a repository system.

Table 10: Computation Data Model Design

No.	Field	Remark
1.	RuleID	This is a Primary Key.
2.	Description	Describes the purpose of the rule.
3.	Type	The result of the computation as Integer, Text, Currency, Date, and so on.
4.	Rule Expression	User understandable and maintainable rule that enables us to parse a computational expression, which is understood by the repository system. The Rules Management Tool with restricted English language vocabularies manages this expression.

EXTENDED CLASS DIAGRAM DEPICTING RDM

A stereotyped entity class type is designed. The class has the Rules Types characteristics specified above (i.e., the dynamic attributes and dynamic operations representing rule types of the class).

Figure 10 below depicts the Extended UML class diagram for the collection of Rules of a business rules system.

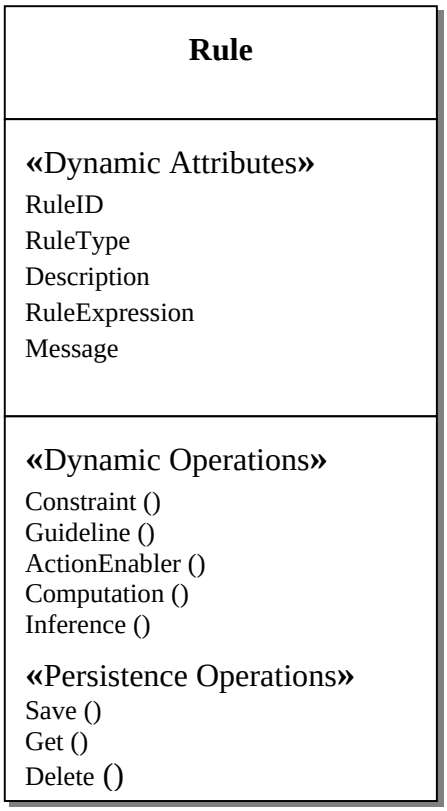


Figure 10: Extended UML Class Diagram for Rules

Table 11 below describes the stereotypes used to extend the above UML class diagram for representing or modeling the collection of rules in a business rules system.

Table 11: Description of the Extensions used for RDM

No.	Field	Remark
1	«Dynamic Attributes»	These are the union of all the attributes found in the above entity models for the different rules types, which can be changed by the user of a system. RuleType represents the types of rule as identified earlier in Table 7 above. It is dynamic in its nature for the fact that a rule may change from one type to the other. For example, a guideline may be revised to a constraint rule type or vice versa.
2.	«Dynamic Operations»	This represents the union of all the five types of rules, which can be changed by the end users of a system, dynamically without programmers' interventions. In other words, these operations are implemented to parse the information entered through the Rules Management Tool. Object-oriented programmers interpret the rule expressions, which are made up of the restricted English vocabularies, into predefined algorithms. The group of operations is implemented in business classes of the application being developed.
3.	«Persistence Operations»	As an entity class, it requires to have the default methods such as *Save* (), *Delete* (), and *Get* () operations for managing rules data in a business rules repository system.

REMARKS

1. The Dynamic Stereotypes specified above (i.e., «Dynamic Attributes» and «Dynamic Operations») can be treated as built-in characteristics of the class.
2. Rules Management Tool (RMT), which is one of Business Rules Management Tools, manages the implementation of this class.

4.2.4 METAMODEL FOR BUSINESS RULES

To model business rules as a whole, the extended UML class diagrams obtained in the previous sections for the different flavors/elements of business rules (i.e., Terms, Facts and Rules) are utilized. After using these diagrams and the relationships between these flavors, the following extended UML class diagram as shown in Figure 11 below is obtained.

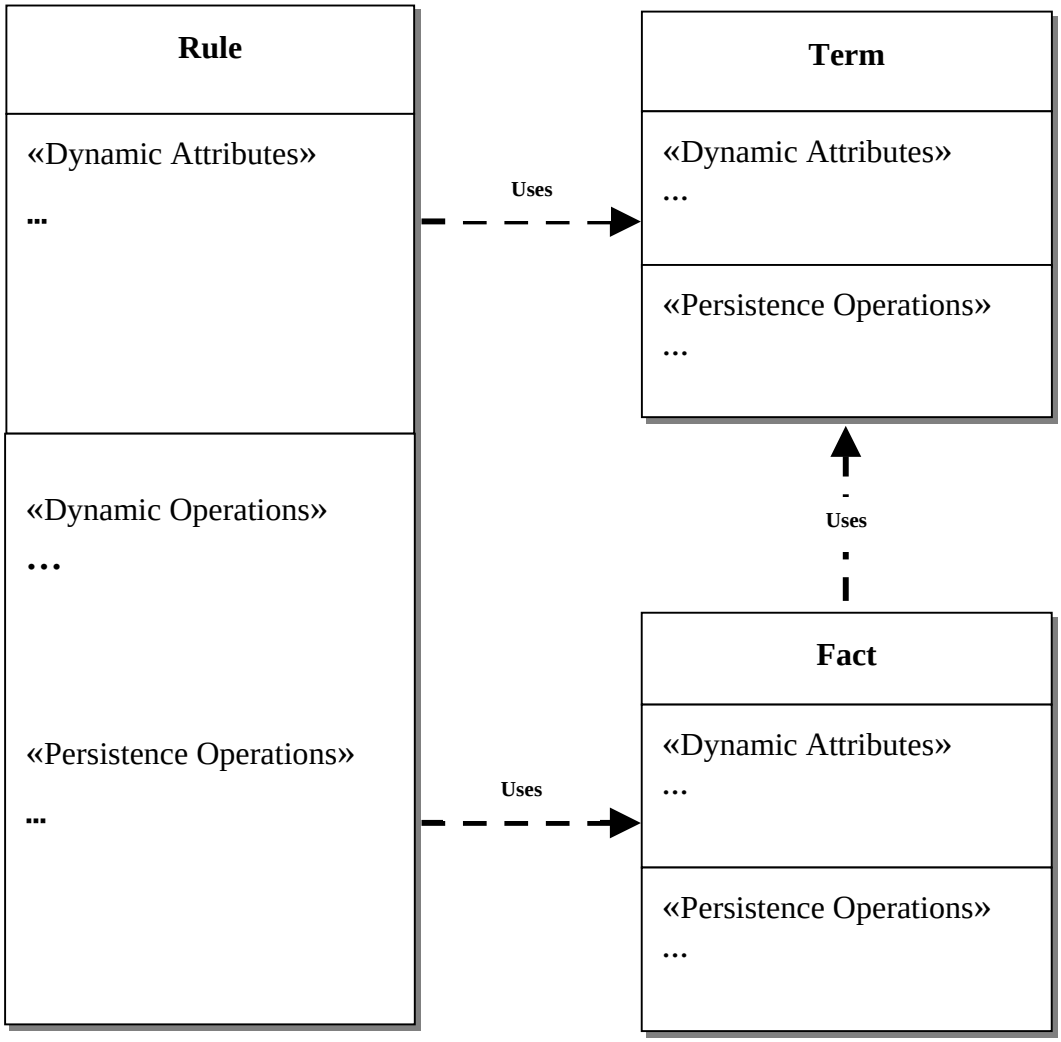


Figure 11: Business Rules Metamodel using Extended UML class diagrams

This diagram is referred to as Business Rules Metamodel.

REMARKS

1. Rules Management Tool (RMT), which is an essential tool and one of the Business Rules Management Tools, manages the Rule class (i.e., «Dynamic Attributes» and «Dynamic Operations» of the rule class).
2. The «Dynamic Operations» are stored in a repository system (for example, first in the RDBMS, and then in XSL).
3. The «Dynamic Attributes» for the Terms and Facts are managed by MMT
4. «Persistence Operations» is similar to operations implemented at the MMT – Attribute Manager (Save, New, Refresh). See the prototype development at the implementation perspectives (Section 4.4) of this thesis. As indicated previously, the persistence layer of an application implements this set of operations.
5. Fact class is managed by using MMT. Entity is first entered and then the attributes contained in that entity are introduced. This formation of relationship is referred to as the Containment fact/relationship. The other type of relationship is Table Relationship, which is referred to as the Related To fact/relationship. Fact Manager of MMT manages these relationships/facts.
6. The Persistence Operations are handled by the methods of the corresponding class of the business rules flavors (Terms, Facts or Rules). Actually, these can be factored out and put into a persistence class(es). Then, the operations of this common class or group of classes can be accessed by the classes of the business rules flavors.

To conclude this section, BROOM has got a modeling language after extending the standard UML class diagram. The extension is supposed to be a lightweight built using UML extension mechanism (stereotypes) atop of the subset of the standard UML 2.0.

Nevertheless, there are some issues that can be raised in implementing a business rules system using the extended UML class diagrams. Some of these issues are the following:

1. How can we implement those Dynamic Structures (attributes) and Behaviors (operations) of a business rules system in OOP languages? For example, how can we execute those object-oriented operations/methods modeled using the extended class diagram?
2. Is there any shortcoming/bottleneck such as performance, and security in using this model with a business rule repository system?
3. How can we provide a modeling tool for supporting efficient and effective modeling and automation of the extended class diagram (for reverse engineering purposes, for example)?

There can also be other important implementation issues that need to be raised. We will focus, however, on these issues in the next sections of this thesis.

4.3 ARCHITECTURAL FRAMEWORK DEVELOPMENT

The main advantage in implementing an application with an architectural framework is to provide a basic structure for the development of an application. An architectural framework is essential since it allows improving and accelerating the development process of a business rules system through architecture reuse.

The next sections provide frameworks, which show the physical and architectural design aspects of business rules systems development using BROOM.

4.3.1 GENERAL FRAMEWORK FOR BROOM

In implementing a business rules system, the following categories of technologies: Programming Languages, Rule Engines, Repository Systems, and Support Tools are employed. These components along with other techniques are bundled within a group to form the general framework fro BROOM.

In the following figure (Figure 12), an architecture that depicts the components and their interactions for a business rules system development is presented.

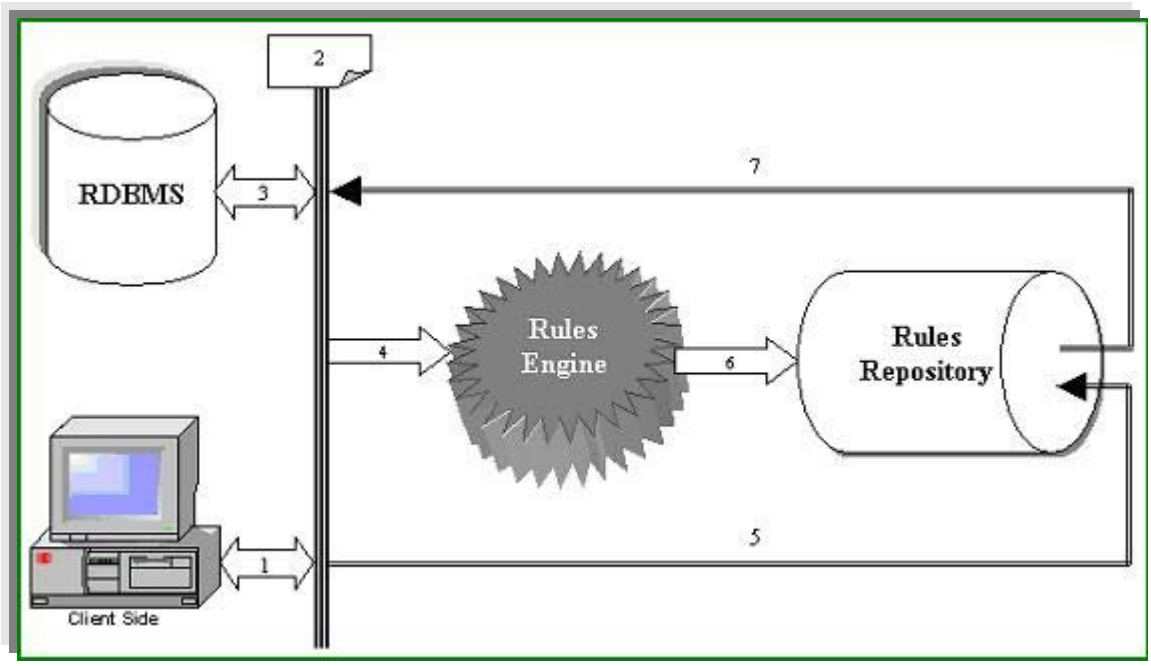


Figure 12: A General Architectural Framework Design for BROOM

The following table explains the *Paths* (1 through 7) indicated in the diagram presented in Figure 12 above.

Table 12: Explanation for the general framework

Path	Meaning
1	This is the client side and the interface communication. It applies while storing Metamodel, rules and data using MMT, RMT (to store the characteristics of a rule) and DMT respectively. MMT and RMT, for example, use the path 1→2→3→2→1 to manage the Metamodel and rules characteristics to be stored in the database
2	This is the Interface of the framework. It has the following contextual meanings: • If the interface communicates with the RDBMS or Client side, the interface acts as a persistence/data service layer (DSL). • If the interface communicates with a rule engine, the interface acts as a triggering mechanism of the rule engine. • If the interface communicates with a rule repository system, the interface acts as message and data sender (5) to or receiver (7) from rules repository system.
3	This is the RDBMS and interface communication. It is actually the communication between the data service layer and the RDBMS.
4	The interface and the Rule Engine communication. The rule engine is initiated by the interface, which is a form of triggering mechanism.
5	The interface and the repository system communication. The interface sends data and/or message to the rules repository system. This is required for updating the rules in the repository system by RMT. The path in updating rule repository is 1→2→5→7→2→1. That is, update is requested, data and message are sent by the client, message is sent back to the interface, and the interface sends message to the client.
6	This is a rule engine and a rules repository system communication. The rule engine activates appropriate rules stored in the repository system.
7	This is a rule repository system and interface communication. The response of the rules repository system is sent back to the requester through the interface.

In the above framework, it is to be noted that support tools such as MMT, RMT and DMT and some of the presentation layers' elements of a business rules system are not shown. However, we can assume as if all of them reside in the Client side of the framework. The arrows indicate the transfer of data and/or messages from one part to the other.

4.3.2 IMPLEMENTING THE FRAMEWORK OF BROOM

Figure 13 below depicts an example of the implementation of the above framework.

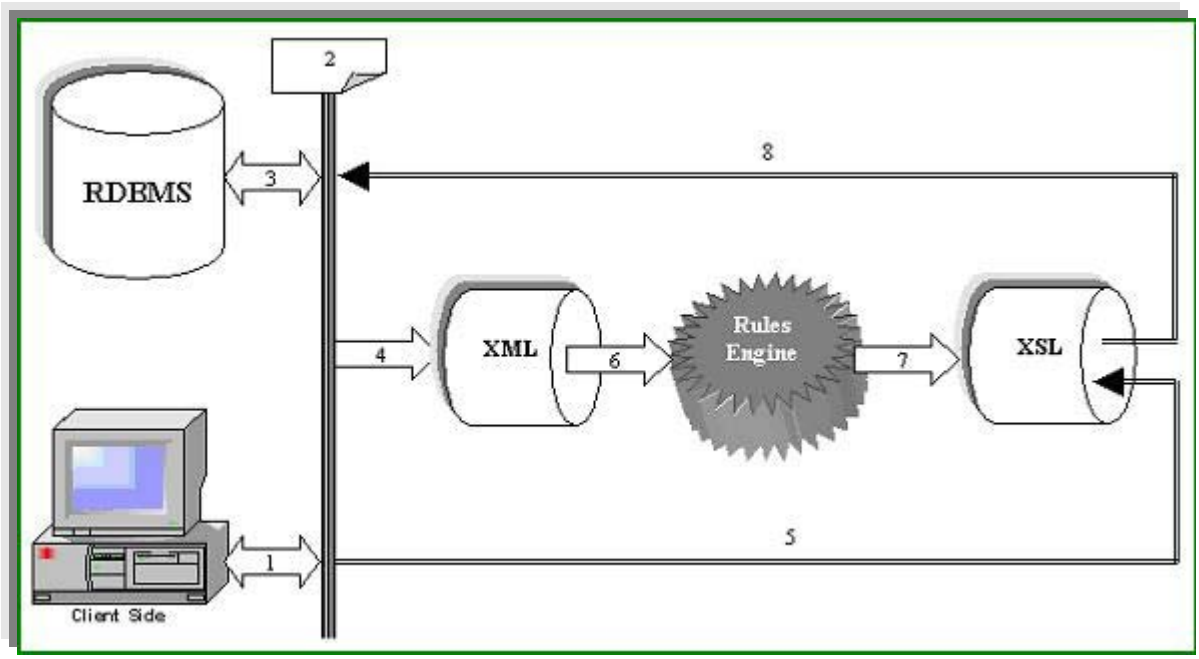


Figure 13: Extended framework of BROOM using Boglaev's work

The following table explains the *paths* (1 through 8) shown in the diagram presented in Figure 13 above.

Table 13: Explanation for the extended framework

Path	Meaning
1	Same as 1 of the General Framework.. The Client side will not directly update RDBMS. That is, record entered through the Client side is stored in the RDBMS via the XML file through the Interface (2)- data service layer.
2	Same as 2 of the General Framework. This is the Interface that acts as a mediator for the Client & DBMS data and the other parts of the architecture.
3	Business Rules Metamodel and Data are stored/read in/from an RDBMS permanently.
4	A record is tracked by the Interface (2), which updates the XML file by that record. Hence, there is a change event on the file. Note that the XML is used as an intermediate storage mechanism for a record to trigger an event for the execution of rules stored in XSL. See the XML convention and XSL sample used for the Design Pattern of the Rule Engine.
5	This applies to RMT while saving rules expressions, which are readable by the end users, and other characteristics such as RuleID, Rule Type, and Message of a rule in a database.
6	The change event in the XML file (6) triggers the Rule Engine that is coded using Java**.**
7	The Rule Engine then triggers the business rules stored in XSL file, which is managed by the Rules Management Tool (RMT).
8	Appropriate Rules stored in XSL files are executed. Data and/or message are returned to the calling program through the Interface (2).

4.4 IMPLEMENTATION PERSPECTIVES

In the previous sections, different models and framework for representing and implementing business rules systems were produced. However, these models and framework can only be realized if proper technologies (i.e., support tools and techniques) are chosen and utilized [57].

The following subsections describe the selection and development of some of the support tools and techniques for developing business rules systems using the proposed method.

4.4.1 PROGRAMMING LANGUAGES

The choice of a programming language is one of the important issues that need to be considered in a software development. An object-oriented programming language will be used for the implementation of a business rules system. To that end, we have a number of such programming languages that are supposed to satisfy the requirements of our method in managing business rules. Some of the languages are Java, C++, and SmallTalk.

In particular, Java is chosen for the fact that it can address interoperable problems that are prevalent in other languages. In other words, it is selected for its capability of platform independency.

In addition, for our demonstration purpose, Microsoft Visual Basic 6.0 (VB6) is also selected for implementing essential support tools known as Business Rules Management Tools, which contain Metamodel Management Tool, Data Management Tool, and Rule Management Tool for facilitating the development of a business rules system. The reason for choosing VB6 is its matured capabilities in rapid application development (RAD).

4.4.2 RULE ENGINE

We require a rule engine for parsing, selecting, and firing business rules refactored or separated from the procedural codes of an application and stored in a business rules repository system.

As presented in Section 4.3.2 of this thesis, we will be implementing the framework of BROOM using Boglaev's design pattern for a rule engine [7]. This rule engine is selected since it is lightweight and uses an object-oriented programming language, Java that is selected in the previous section of this thesis.

4.4.3 REPOSITORY SYSTEMS

To store Metamodel and data of a business rules system, any RDBMS supporting the standard SQL can be used. This is because we create different tables of a business rules system using Data Definition Language (DDL) of standard SQL with the help of the Metamodel Management Tool (MMT). MMT requests a user to provide information about business rules of the system under consideration. RDBMS is selected since it is a widely used DBMS [62].

For our demonstration purpose, Microsoft Access2000 is selected as the RDBMS for storing an organization's business rules Metamodel and data. We can employ XML as a means of storing the intermediate data (from RDBMS or user's data entry) to communicate with a rule engine and fire appropriate rule(s) stored in a repository system, XSL, for example, as indicated in [7].

4.4.4 BASIC DEVELOPMENT TOOLS

A system development can be carried out efficiently and effectively provided proper support tools are utilized. Some of the tools required for our method are (i) a modeling tool that should support the extensions made earlier, (ii) Metamodel Management Tool, which enables us to store a metamodel of a business rules system and generates the database based on the metamodel provided by the user and stored in a repository system, (iii) Data Management Tool for data entry

purpose for the entities created by MMT, and (iv) Rule Management Tool that helps us manage the rules stored in a repository system such as XSL.

Table 14 below presents some of the important development tools for the lightweight Rule Engine using XSL and business rules management tools such as Metamodel Management Tool (MMT), Data Management Tool (DMT), and Rules Management Tool (RMT).

Table 14: Basic Development Tools for Developing a Business Rules System

S. No	Tools	Description
1.	JDK1.4 or later	Java Compiler.
2.	Apache ant 1.5.2 or later	Ant is a Java based build tool. In theory it is kind of like "make" without makes wrinkles and with the full portability of pure java code.
3.	xalan-j_2_5_D1or later	Xalan processor from Apache Software Foundation. It builds on the Java API for XML Processing and implements the transformation API for XML.
4.	JUnit 3.8.1 or later	
5.	MS VB6.0	For developing the different prototypes of support tools (MMT, RMT, and DMT), for demonstration purpose.
6.	MS Access 2000	For storing the Metamodel of Business Rules and data or transactions of a system. This is also for demo purpose.
7.	XML and XSLT	XML will be used to persist the intermediate data for triggering the Rule Engine and, XSLT will be used to store or persist the separated actual Rules that are applied on the business terms and facts of the business rules systems.
8.	IDE	NetBeans (as Integrated Development Environment – IDE) will be used.

As indicated earlier, to demonstrate our work, VB6 will be used for automatic creation of (i) MS Access database for storing the Metamodel and data of business rules, (ii) XML files and style sheets (XSL) for storing the intermediate data and the rules respectively, and (iii) the GUIs of a business rules system for data entries purpose. The other tools specified in Table 14 above are used for developing a rule engine and a rules repository system.

4.4.5 SUPPORT TOOLS PROTOTYPE DEVELOPMENT

The UML class diagrams extensions presented in Section 4.2.3 are used for developing the support tools. Furthermore, the dynamicity of the business rules are supposed to be managed by the support tools themselves.

Table 15 below provides some of the basic steps to be followed and performed in developing prototype support tools, Business Rules Management Tool and a lightweight rule engine presented in [7].

Table 15: Basic steps followed and performed in developing the prototypes

STEP	PURPOSE/DESCRIPTION	REMARK
1.	Store Business Entities, Attributes, and Facts/Relationships in an RDBMS chosen for the case study.	Metamodel Management Tool (MMT) and Data Management Tool (DMT) Prototype Development
2.	Automatically create the corresponding tables of the database from the information stored using Step 1 above	
3.	Create GUIs for entering the data in the corresponding tables of the database created for the business rules system.	

STEP	PURPOSE/DESCRIPTION	REMARK
4.	Rule Editor that can convert some restricted English sentences with business rules vocabularies to XSL rule templates and Rule Modifier class that could insert or update at run time an XSL rule set and communicate with the rule editor [7].	This is in general known as, Rule Management Tool (RMT).
5.	Create automatically XML document from the data stored in the database and an agreed layout of the XML document.	
6.	Create an evolvable Rule Engine, which should *listen* to the events created by the change of XML files that serve as intermediate storage & triggering mechanism for the Rule Engine.	Application that will use Java, XML and XSLT
7.	Test the results with JUnit, again as presented in [7]	

4.4.5.1 XML FILES CONVENTIONS

Each client has its own XML files and these file serve as an intermediate file. XML should be automatically filled with data either from: a) The GUI of the data entry, or b) The database via the interface (Persistence/Data Service Layer). In both cases (a and b, above), the XML files are used to trigger the rule engine and execute appropriate rules stored in rules repository system called eXtensible Stylesheet Language (XSL) [43]. XSL file can be created with different rule types (Rule Sets):

i) Guidelines (Business Warning)
ii) Constraint (Business Error: Not Database error such as type mismatch)
iii) Computations (Business Calculations)
iv) Action Enablers (Business If Then Else Statement other than i) and ii))

Each of the above may represent rule sets for a business rules system.

Listing 1 below is the convention used for the layout or format of an XML file.

Listing 1: Convention for XML Files Format

```
<?xml version="1.0" encoding="UTF-8" ?>
 <Table>
  <Row>
        <Col1>value1</Col1>
        <Col2>value2</Col2>
        <Col3>value3</Col3>
        .
        .
        .

        <Coln>valuen</Coln>
  </Row>
</Table>
```

Note that XML files are created automatically by DMT. Listing 2 below presents sample XSL file created automatically by the RMT.

Listing 2: Sample XSL file created by RMT

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:java="java" xmlns:action="simple.rules.actions.SendMessage">
<!-- Rule Type 1: Guideline  -->
<xsl:template name="rule1">
      <xsl:param name="sal"/>
      <xsl:param name="emp_id"/>
      <xsl:variable name="par">
         <xsl:value-of select="number($sal)"/>
      </xsl:variable>
     <xsl:if test="$par &lt; 150">
        <xsl:value-of select="action:sendMsg2Stdout(concat('Rule 1: Warning for Employee Id# ', $emp_id, ' The Minimum BasicSalary Should be 200.00', ' .'), $index)" />
     </xsl:if>
</xsl:template>
</xsl:stylesheet>
```

4.4.5.2 METAMODEL MANAGEMENT TOOL (MMT)

A prototype of the Metamodel Management Tool (MMT) is developed for the proposed method. Refer to the diagram. Figure 14 depicts the relationships of the tables of the business rules (Terms, Facts and Rules). It represents a simple Metamodel of business rules and this diagram is the basis for the MMT prototype development.

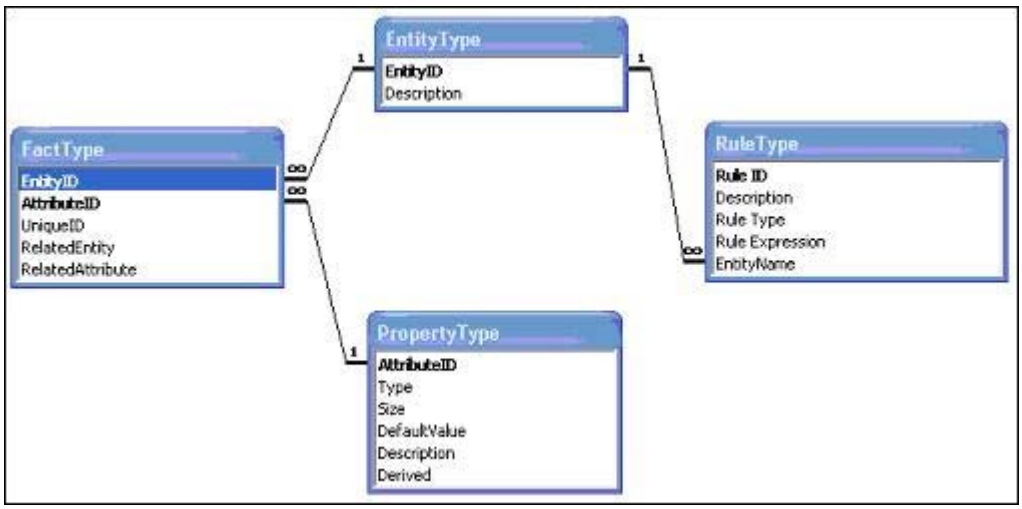


Figure 14: Simple Business Rules Metamodel: Entities, Attributes, Facts & Rules

The Metamodel is simple for the fact that the rule expressions are assumed to be derived from a single entity. The model presented in the above figure is obtained from the data models presented in Section 4.2.3 similar to the TypeSquare Model presented by Yoder et al. [63] for Adaptive Object Model.

MMT serves for managing terms (EntityType and PropertyType) and facts (FactType) of a business rules system as presented in Figure 14 above. The tool helps create the Metamodel and the corresponding database tables based on the information stored about the Metamodel in a repository system. That is, this tool covers the first two implementation steps presented in Table 15 above.

MMT has got three modules: Attribute Manager, Entity Manager and Fact Manager. Figure 15 below depicts the Attribute Manager.

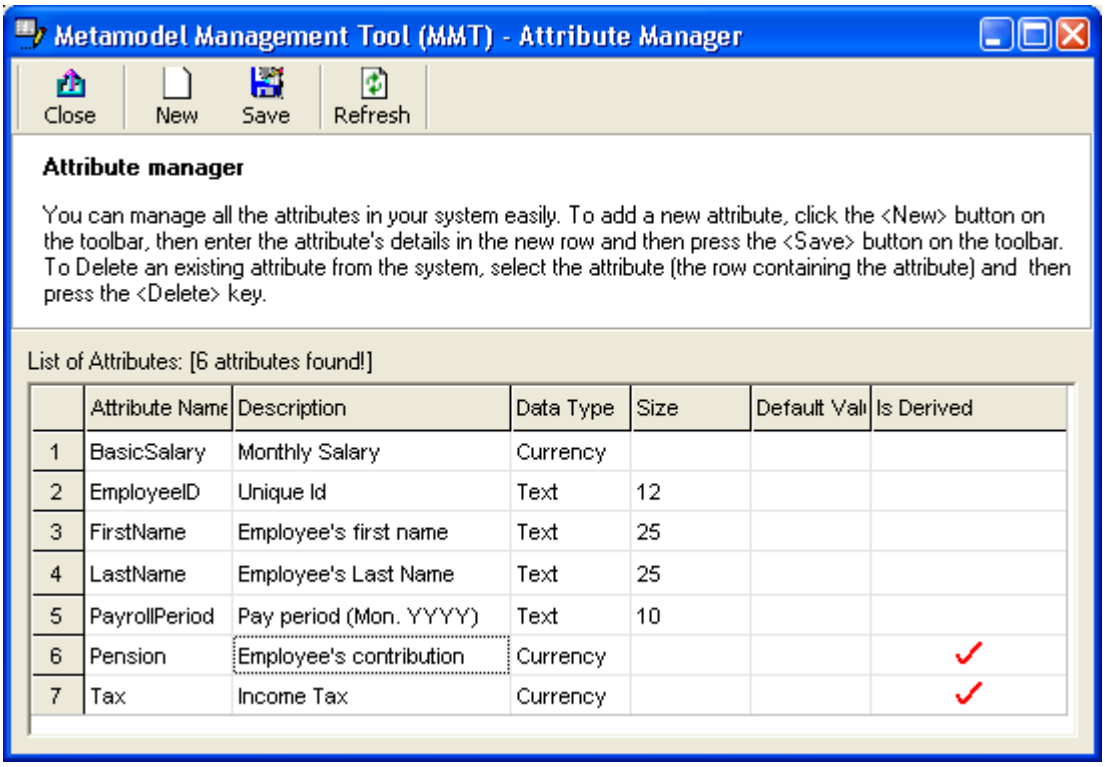


Figure 15: Metamodel Management Tool - Attribute Manager

Attribute Manager module manages the attributes of business rules systems by using PropertyType presented in Figure 14 above as its building block.

Similarly, Entity Manager manages the entities by using the EntityType presented in Figure 14 above. For example, Entity Manager shows the available Entities and the corresponding attributes along with their characteristics. Refer to Figure 16 below for a screenshot of the Entity Manger prototype.

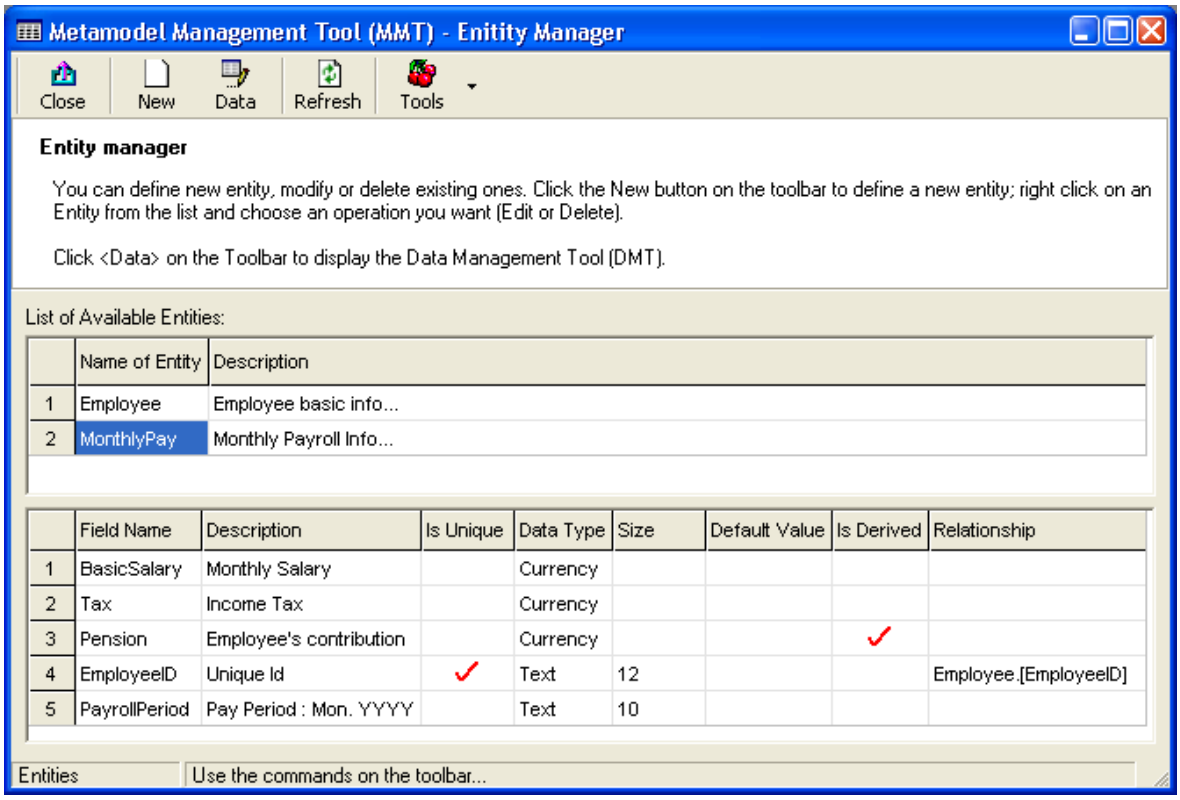


Figure 16: Metamodel Management Tool - Entity Manager Preview Window

New Entity can be defined using the “New” command found at the toolbar of the window.

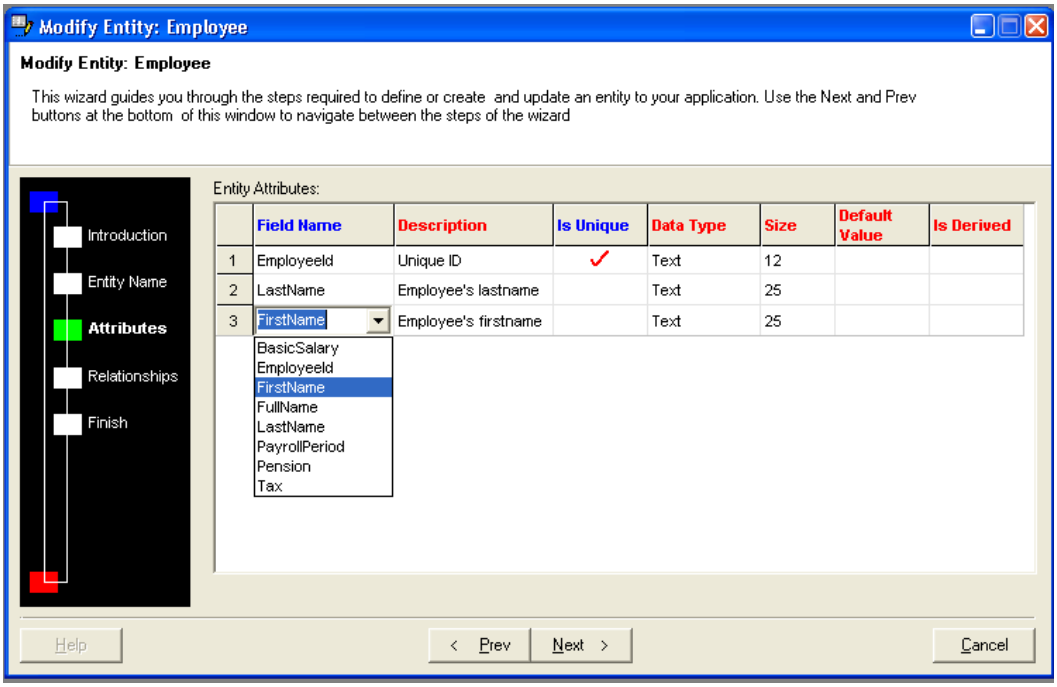


Figure 17: Metamodel Management Tool - Fact Manager

When an item of the Metamodel is modified, other related items should also be synchronized. For example, renaming an attribute of a business rules system entails us to ALTER the tables that refer the attribute being changed. Moreover, the data already contained by the attribute should not be lost.

The data type of an attribute may be one of the following types: Text, Currency, Date, Yes/No, and Integer. MMT maps this data types into the specific or physical database data types while executing SQL DDL statements.

4.4.5.3 DATA MANAGEMENT TOOL (DMT)

Data Management Tool (DMT) is supposed to address the third implementation step presented in Table 14 above (i.e., data entry facilities for the created entities and attributes). Figure 18 below depicts prototyped DMT's window while entering data of the attributes for the entity *employee*.

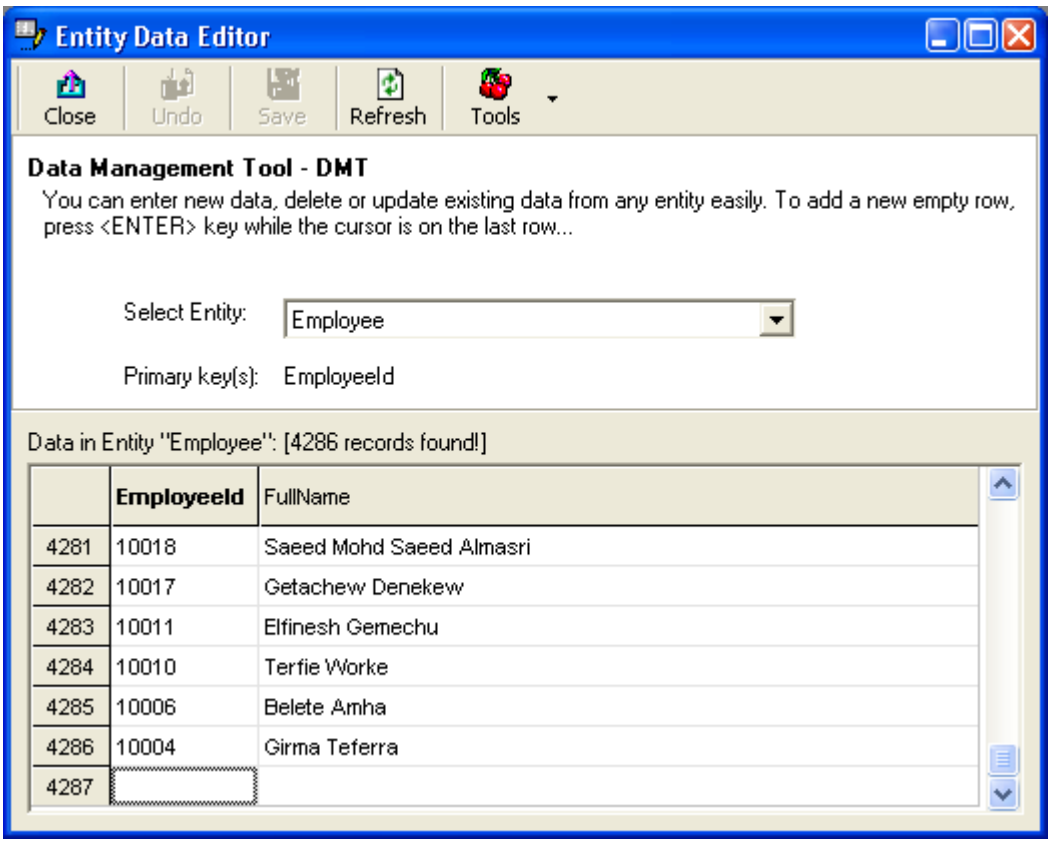


Figure 18: Data Management Tool allowing data entry facilities

In order to allow users to manage the data for the terms created by MMT, DMT uses the Metamodel, the actual tables created in a database, and a GUI element known as DynaGrid[4].

4.4.5.4 RULES MANAGEMENT TOOL (RMT)

Rules Management Tool (RMT) is supposed to address the fourth implementation step presented in Table 14 above. That is, RMT serves as a Rule Editor that can convert the English sentences constructed out of the restricted vocabularies provided by the tool. It manipulates the XSL files based on the information provided by the user. Figure 19 below presents a screenshot of RMT while editing a rule: guideline of rule id equals Rule001.

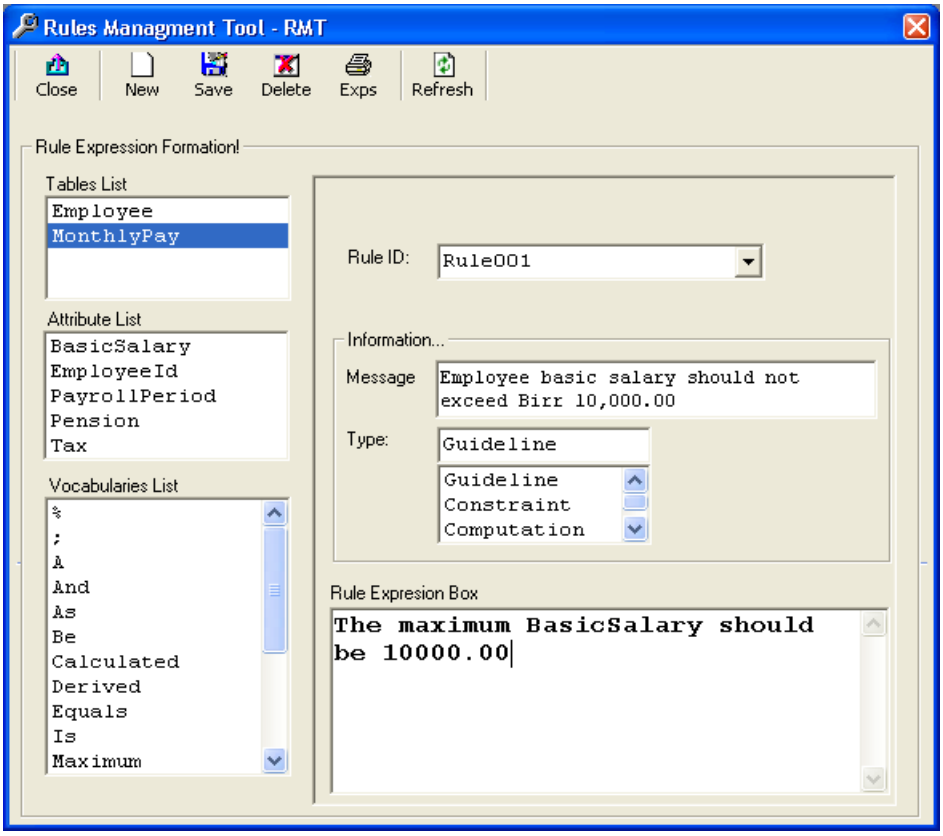


Figure 19: Rule Management Tool while editing a rule expression

[4] DynaGrid is a grid of ActiveX Control developed by Ato Kassahun H/Gabriel for Executive MIS Software Project of Ethiopian Airlines. It is a versatile grid that helps us manage data entry and other facilities in an efficient and effective way. I would like to highly appreciate his willingness in allowing us to use the grid and his kind assistance in most part of the VB codes associated with MMT and DMT.

When a rule is saved, RMT saves it in two *places*. The first one is, the rule along with its characteristics such as rule expression, stored in the database (RuleType table of Figure 14) for future reference (i.e., if a user wants to view a rule for changing it, for instance, it can be viewed as it was saved previously). The second place for saving the rule is in the rule repository, XSL, after converting the rule expressions into the correct syntax of XSL. That is, XSL files are amended whenever there is a change in a rule expression

There are some important issues to be addressed by the support tools such as RMT. For example, conflicting rules should be managed by RMT. Conflicting rules are those rules, which contradict each other. For example, "*The maximum age of an employee is 55 years"* and "*The maximum age of an employee is 60 years"* are conflicting rules, "*The maximum salary of an employee is BIRR 1,000.00"* and "*The minimum salary of an employee is BIRR 1,200.00"* are also conflicting rules.

In a RuleType table depicted in Figure 14 above, the field *Rule Expression* is unique, in addition to *RuleId*. Furthermore, the Rule Expression should have the following characteristics:

a. No conflicting rules between and among the expressions.
b. Every Rule Expression should be semantically unique.
c. A Rule Expression should have a meaning that can be applicable in the space of a business rules system.

4.5 SUMMARY

To sum up this chapter, a method is developed for developing an object-oriented business rules system where by business rules are separated from procedural codes and database constructs. To achieve our goal, a business rules system is modeled to visualize, specify, construct and document its artifacts with an extension of UML class diagrams. This extension is built using *Stereotype* extension mechanism atop of the standard UML class diagram. Since most UML tools such as Rational Rose support user-defined stereotypes, they can be used to model a business rules system.

Furthermore, the architectural framework, which is composed of BROOM Essential Support Tools (BEST), RDBMS, the interface, and the rule engine, was designed. Some of the tools that make up BEST are MMT, RMT and DMT.

CHAPTER 5

CASE STUDY

5.1 PURPOSE

The purpose of this chapter is to demonstrate on how to apply BROOM to a case study and analyze the results of the demonstration. The essential support tools of the method such as Metamodel Management Tool (MMT), Rules Management Tool (RMT), Data Management Tool (DMT), and Rules Engine will be used.

In the next sections, we select a system as a case study, identify its potential sample business rules, apply the method for the case study, and make analysis of the results obtained from the case study. Finally, we summarize the chapter with some important points of the chapter.

5.2 A CASE STUDY AND ITS BUSINESS RULES

5.2.1 SELECTION AND ITS RATIONALE

To demonstrate the output of this research with a practical example, a simple payroll application system is selected. The data used for this system is obtained from an existing and functional payroll system of Ethiopian Airlines Enterprise.

The application is selected for the following main reasons: (1) It is a common business rules system, and simple for demonstration. (2) It encompasses *rules* of all types: Guidelines, Constraints, Computations, and Action Enablers as described in the next Section. (4) Furthermore, the business rules are subject to change by the end users of the system.

5.2.2 BUSINESS RULES IDENTIFICATION

A payroll system of an organization has a number of business rules. In the following subsections, the business rules of the system are presented by their categories (i.e., Terms, Facts and Rules).

5.2.2.1 TERMS

A term is either an entity or an attribute.

ENTITIES

The following table depicts the entity of the system.

Table 16: List of entities, which are subset of the Terms of the system

No.	Entity Code	Remark
1.	Employee	This holds employee's basic information
2.	MonthlyPay	This is the entity that contains the monthly payroll transactions of the employees

ATTRIBUTES

Table 17 presents some of the attributes of the system.

Table 17: List of attributes, which are subset of the Terms of the system

No.	Attribute Code	Description
1.	EmployeeID	A maximum of 10 alphanumeric chars (Text).
2.	FullName	A maximum of 50 chars of length (Text format
3.	PayrollPeriod	A string of the form MON YYYY format (Text)

No.	Attribute Code	Description
4.	BasicSalary	The basic salary of an employee of the hypothetical organization. Its format is Currency
5.	Pension	It is a *derived*[5] attribute and has Currency format
6.	Tax	This is also a derived attribute and has Currency format

5.2.2.2 FACTS/RELATIONSHIPS

The next table presents the facts/relationships of the system.

Table 18: List of facts/relationships of the system

No.	Fact/Relationship	Remark
1.	Employee has unique Employee ID	Containment of Employee Id attribute in Employee Entity. Employee ID is unique
2.	Employee has FullName	Containment of Name attribute in Employee Entity
3.	MonthlyPay has Employee ID	Containment of Employee Id attribute in MonthlyPayEntity
4.	MonthlyPay has PayrollPeriod	Containment of PayrollPeriod attribute in MonthlyPayEntity. Both Employee ID and PayrollPeriod form a unique key for MonthlyPay
5.	MonthlyPay has Basic Salary	Containment of Basic Salary attribute in MonthlyPayEntity
6.	MonthlyPay has Pension	Containment of Pension attribute in MonthlyPayEntity
7.	MonthlyPay has Tax	Containment of Tax attribute in MonthlyPayEntity
8.	Employee is related to MonthlyPay with Employee ID	**Related To (Tables relationships):** Employee and MonthlyPay are related with an attribute key of Employee Id

[5] A derived attribute is an attribute whose values are obtained from the collaboration of other attribute(s) via some rules such as computation.

5.2.2.3 RULES

In Table 19, four sample rules categorized into the rule types (i.e., Guideline, Constraint, Computation, and Action Enabler) are given.

Table 19: List of sample rules of the system

No.	Rule Category	Rule Code	Rule Expression
1.	GUIDELINE	RULE001	Maximum BasicSalary Should [be] 10,000.00
2.	CONSTRAINT	RULE002	Minimum BasicSalary Must [be] 200.00[6].
3.	COMPUTATION	RULE003	Pension is calculated as 4 % of the BasicSalary
4.	ACTION ENABLER	RULE004	Tax is given by the Progressive Scheme[7]

Note the a guideline is used as a *warning message,* a constraint is used as an *error message*, a computation acts as an assignment of derived attributes, and an Action Enabler serves as an action triggering mechanism upon finding true of a condition. The action that can be triggered may be message, computation, etc.

We can also include additional business rules. However, for the sake of avoiding unnecessary details, only the above business rules are considered. In Section 5.3.3, a demonstration is presented on how to incorporate changing requirements of the system under study by the businesspeople themselves. For a formal way of analyzing business rules of a system, refer to [22].

[6] Ethiopian Government Law puts that the minimum basic salary of an employee of an organization to be Birr 200.00.

[7] The progressive tax rule is as follows: From 1 to 150 → 0%, 150 to 650 → 10%, 650 to 1400 → 15%, 1400 to 2350 → 20%, 2350 to 3550→ 25%, 3550 to 5000 → 30%, above 5000 → 35%.

5.3 APPLYING BROOM

In this section, the method is applied to the case study with the vital information gathered from Section 5.2.2 of this thesis and the prototype support tools developed earlier for the method.

5.3.1 MODELING THE PAYROLL SYSTEM

To model the payroll system, the extended UML Class Diagrams presented in Section 4.2.3 of this Thesis and the business rules of the payroll system identified earlier (about Terms, Facts and Rules) are used. Figure 20 below depicts the employee entity class diagram of the system.

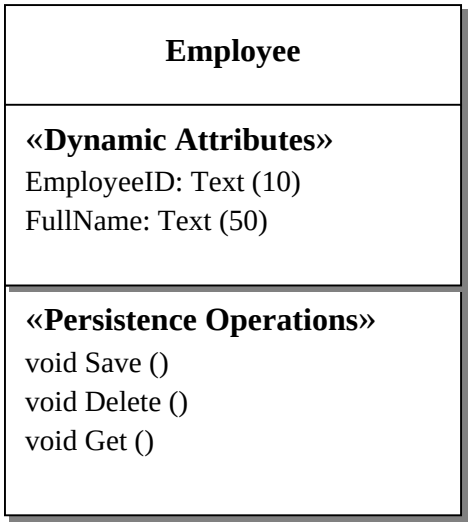


Figure 20: Employee Class Diagram

Employee class is the outcome of the Term and Fact classes. From the Term class and Fact class, the following characteristics of the Employee class are obtained:

1. The name of the class, Employee. It is obtained from the «Dynamic Attributes» of the Term (i.e., TermID, where TermType = “Entity”).
2. The attributes’ names along with their characteristics identified under «Dynamic Attributes» stereotype are obtained from the «Dynamic

Attributes» of the Fact class presented in Section 4.2.3.2 of this thesis and the list of facts identified for the system.

3. The «Persistence Operations». These are the basic operations used for managing the persistent data of the system (Save(), Get(), and Delete()). Figure 21 below presents another entity class diagram, MonthlyPay.

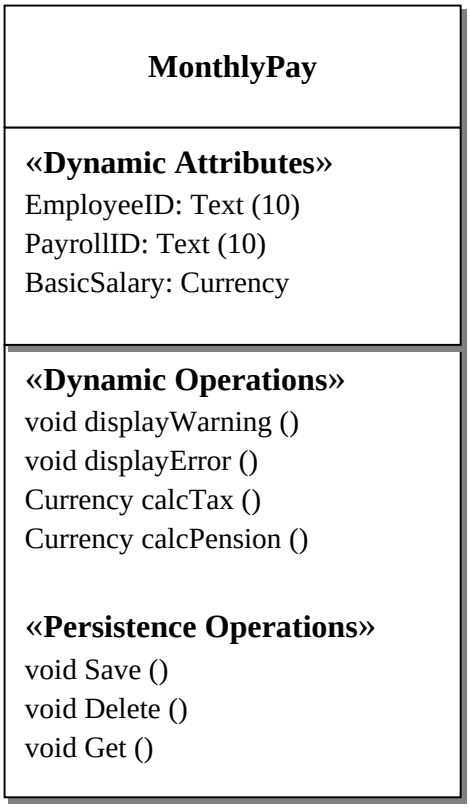


Figure 21: Monthly Payroll Class Diagram

Similarly, the MonthlyPay entity class diagram is obtained from our extension of UML class diagrams for Terms and Facts as in the following:

1. For the «Dynamic Attributes» and «Persistence Operations», the points 1, 2 and 3 specified above for the entity class **Employee** apply

2. The «Dynamic Operations» such as displayWarning () and calcTax () are obtained from the Rule class presented in Section 4.2.3.3 of this Thesis and the rules identified for the system

Figure 22 below illustrates the generalized entity class (base entity class) diagram from which the above entity classes (i.e., Eemployee and MonthlyPay) can be derived.

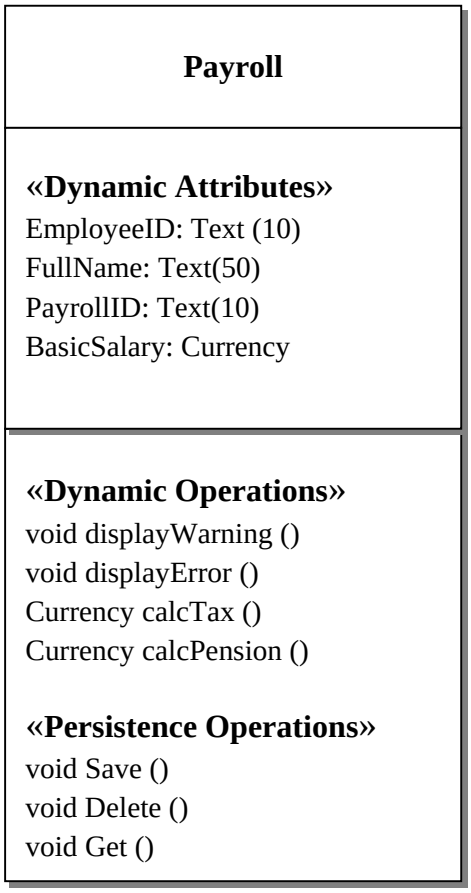


Figure 22: Base Class Diagram

The base class diagram, Payroll, is obtained from by simply taking the union of the attributes and the operations of the derived or sub classes, Employee and MonthlyPay, as indicated under «Dynamic Attributes», «Dynamic Operations» and «Persistence Operations» stereotypes.

Finally, Figure 23 below depicts the containment/aggregation class diagram to illustrate the relationships among the classes identified so far for the system under development.

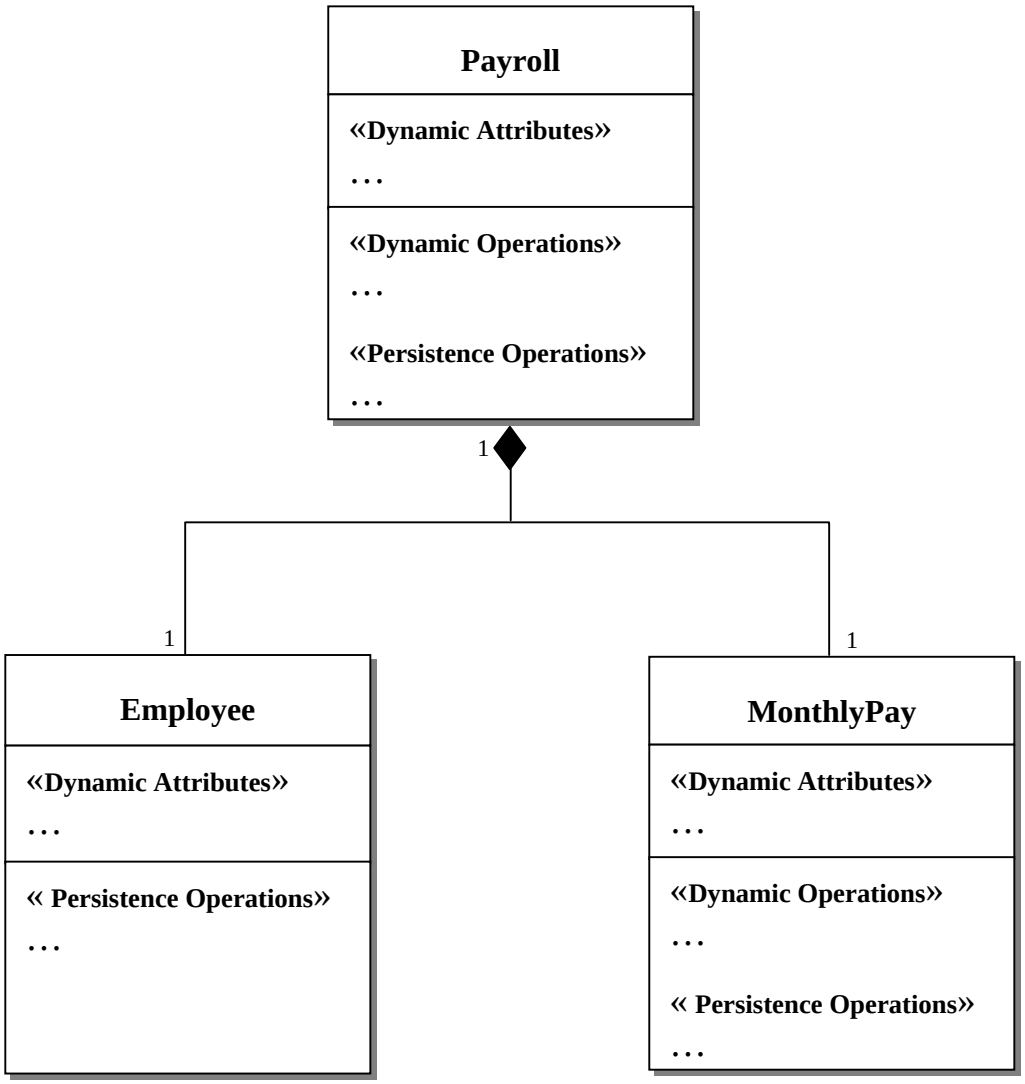


Figure 23: Containment/Aggregation Class Diagram for the Payroll Application

5.3.2 AUTOMATING THE SYSTEM WITH THE SUPPORT TOOLS

The implementation setting is provided in Section 4.4: Implementation Perspectives of this thesis.

5.3.2.1 DEFINING ENTITY, ATTRIBUTES AND FACTS

Figure 24 below depicts sample screenshot of the Attribute Manager for the business rules identified above.

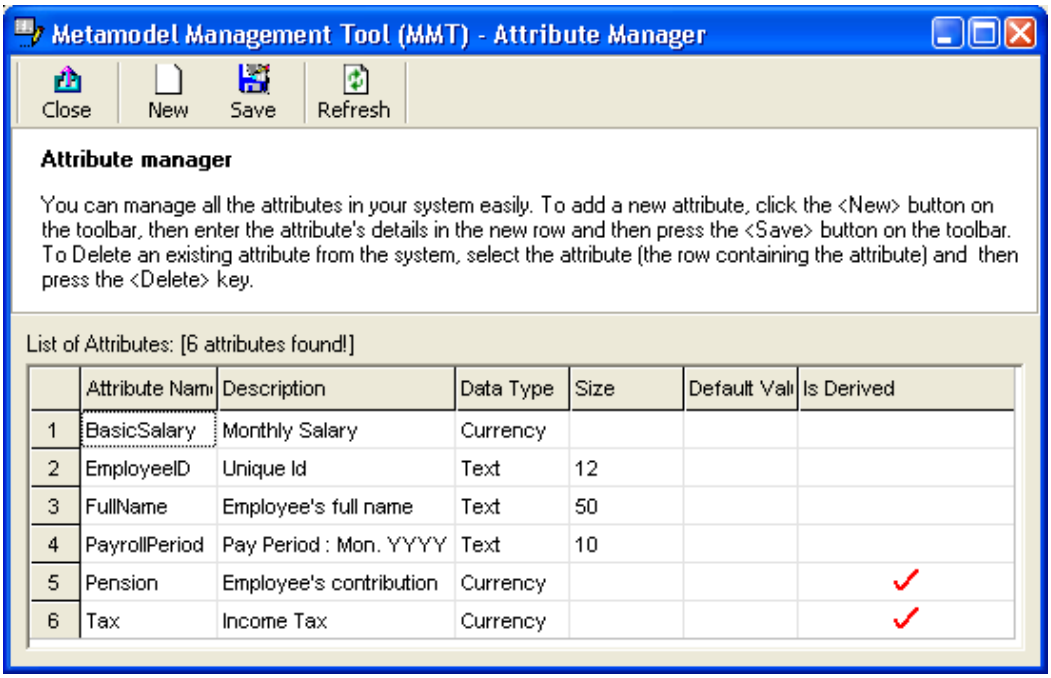


Figure 24: Defining the Attributes with the Attribute Manager

Figure 25 below a screenshot while defining the facts/relationships identified earlier for the employee entity and the available attributes.

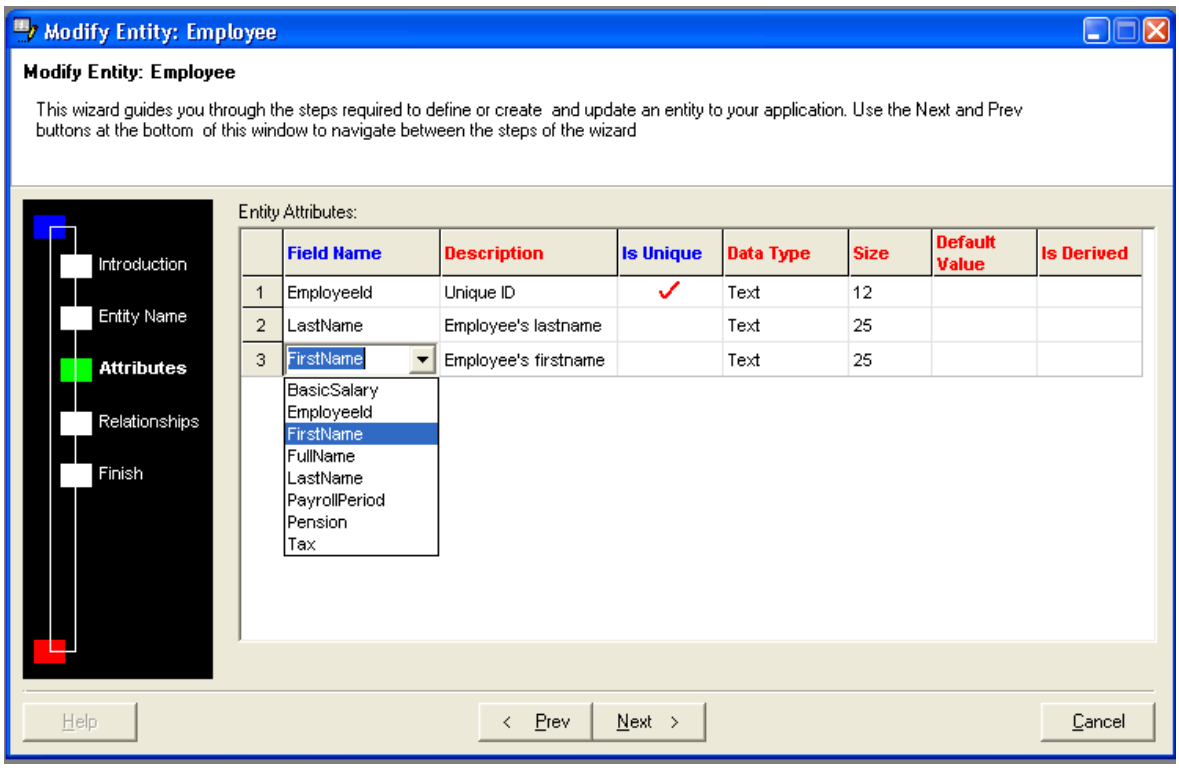


Figure 25: Defining the facts/relationships between Employee and the attributes

Figure 26 and Figure 27 below depict screenshots for the Entity Manager of the entities containing the corresponding attributes.

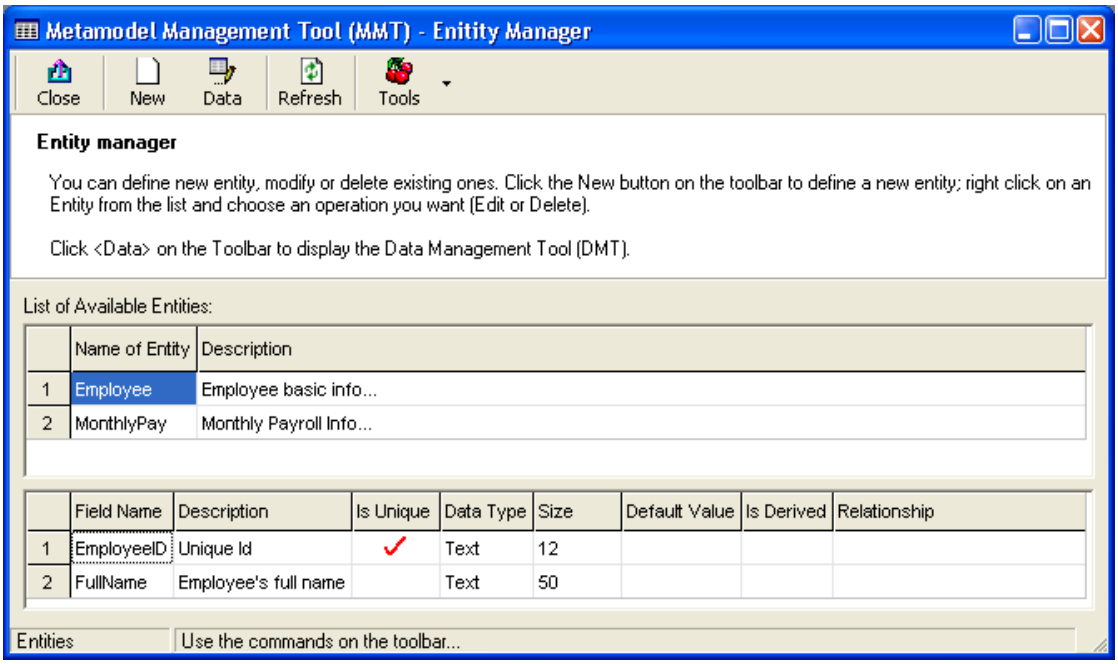


Figure 26: Defining Employee and the *facts* with Attributes via Entity Manager

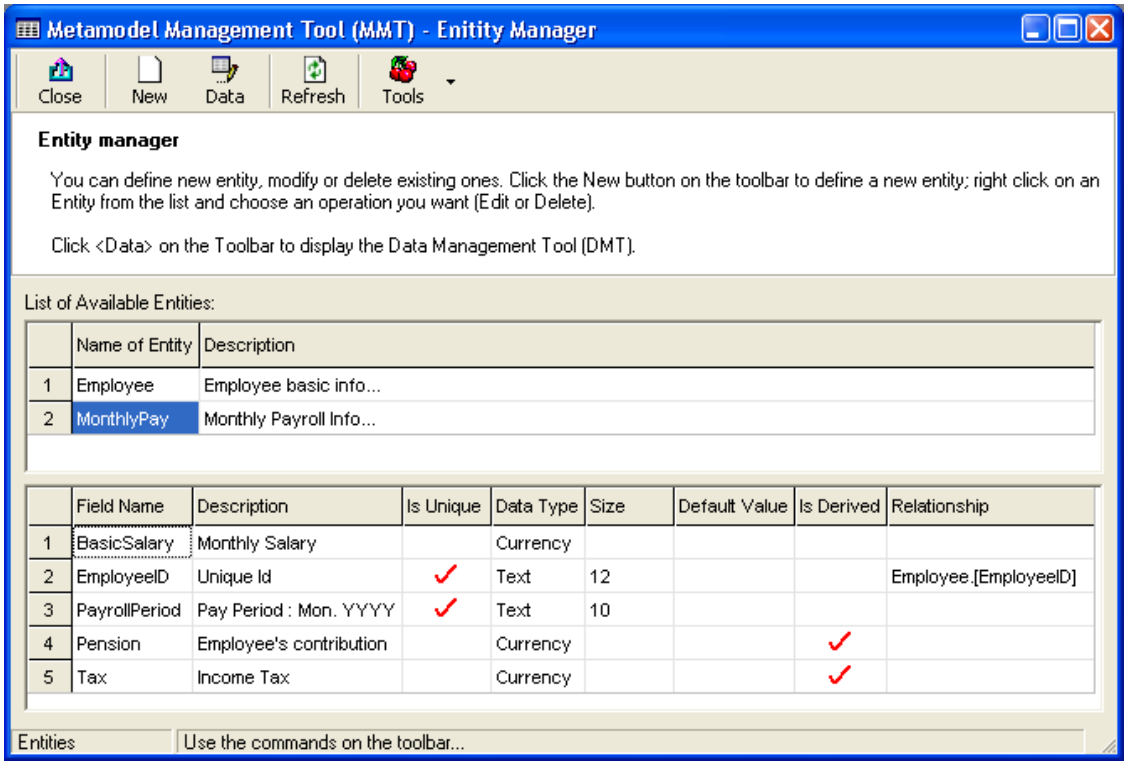


Figure 27: Defining MonthlyPay with *facts and* Attributes via Entity Manager

Figure 28 below is the Facts manager defining the relationship between the two tables (i.e., MonthlyPay and Employee).

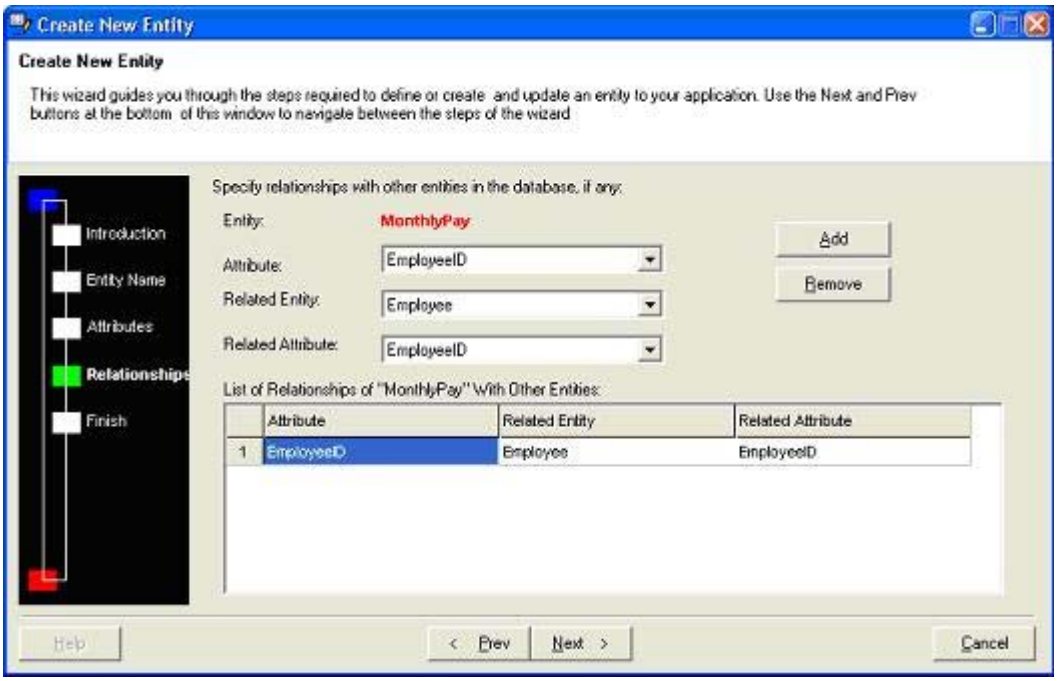


Figure 28: Facts Manager defining the relationships between the two tables

5.3.2.2 MANAGING RULES

Rules Management Tool (RMT) allows business rules systems' users to manage their rules of the system. With the help of RMT, users of the system enter a rule expression as presented in Section 5.2.2.3. RMT then searches for the words or vocabularies used in the rule expression.

RMT is aware of predefined and restricted English vocabularies such as *Minimum, Maximum, Should be, Must be, Calculated, Derived,* and so on. This is realized by storing these vocabularies along with the corresponding rule categories in RDBMS.

The other vocabularies known by the tool are the attributes of the system defined via the MMT. Figure 29 below is a screenshot of the RMT while managing (entering) a rule (Rule001) of the case study.

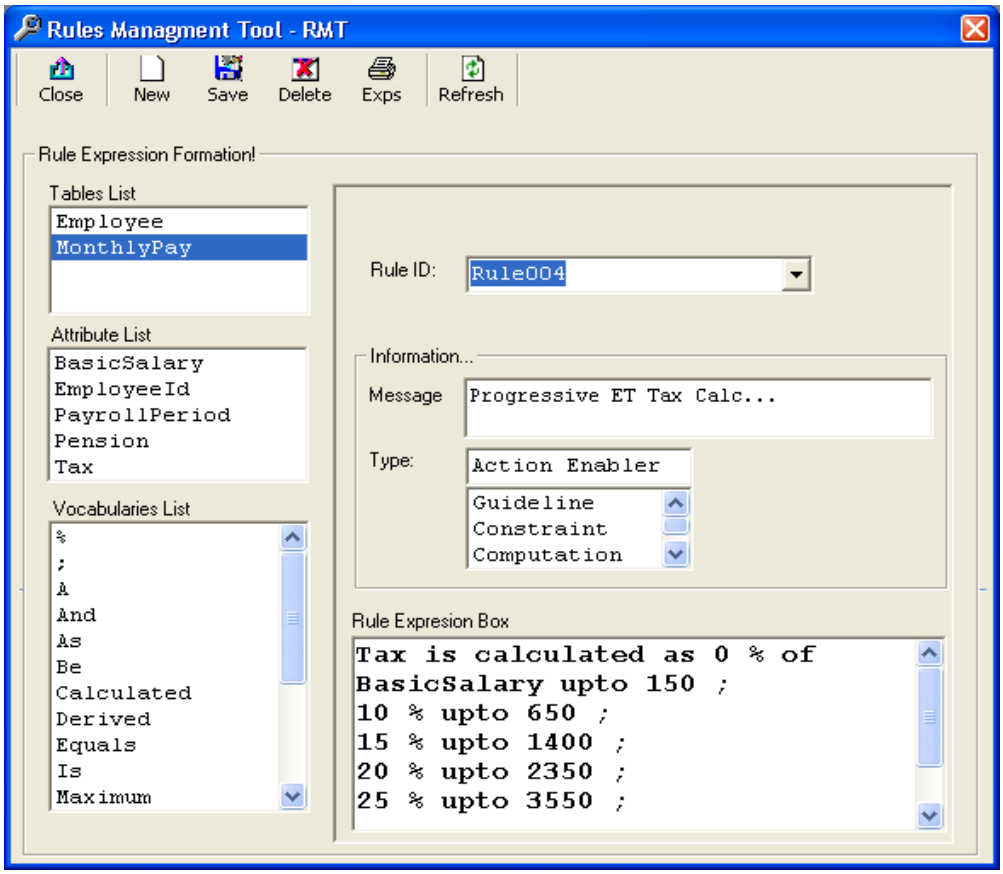


Figure 29: Rule Management Tool (RMT) in action

In other words, RMT helps store or save readable and maintainable rule expressions in the database for future use. Furthermore, it also parses and converts these expressions into a correct syntax of XSLT (XPath) so as to save them in the repository system's format known as XSL. That is, after finding the rule expression valid, it is changed into a form of correct syntax of XSL [43] and then saved to the XSL files. Figure 30 below shows the list of rules entered in the business repository system using RMT.

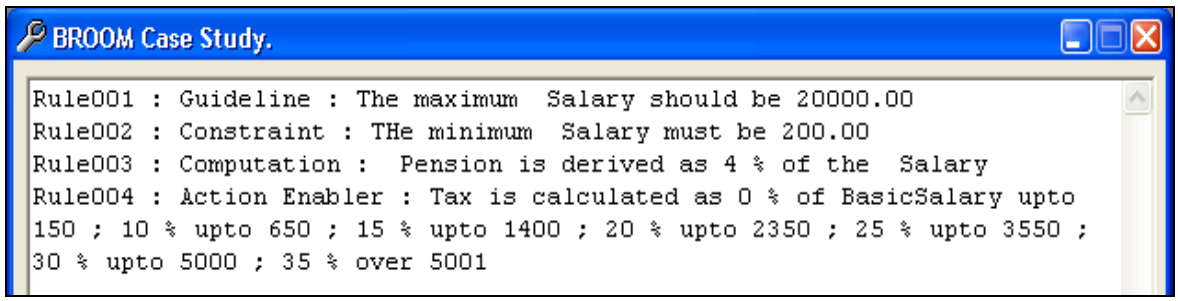


Figure 30: List of rules entered in the repository system by RMT

Listing 3 below is a sample XSL file created by the RMT for the above rules.

Listing 3: Rules stored in an XSL file generated by RMT

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:java="java" xmlns:action="simple.rules.actions.SendMessage">


<!-- Rule Type 1: Guideline  -->
<xsl:template name="rule1">
       <xsl:param name="sal"/>
       <xsl:param name="emp_id"/>
       <xsl:variable name="par">
          <xsl:value-of select="number($sal)"/>
       </xsl:variable>
      <xsl:if test="$par &gt; 20000">
         <xsl:value-of select="action:sendMsg2Stdout(concat('Rule001', ' Rule Type 1:
Warning for Employee Id# ', $emp_id, ' Enter the description...', ' .'), $index)" />
      </xsl:if>
</xsl:template>




<!-- Rule Type 2: Constraint  -->
<xsl:template name="rule2">
      <xsl:param name="sal"/>
      <xsl:param name="emp_id"/>
      <xsl:variable name="par">
         <xsl:value-of select="number($sal)"/>
      </xsl:variable>
      <xsl:if test="$par &lt; 200">
         <xsl:value-of select="action:sendMsg2Stdout(concat('Rule002', ' Rule Type 2:
Error for Employee Id# ', $emp_id, ' Enter the description...', ' .'), $index)" />
      </xsl:if>
</xsl:template>


<!-- Rule Type 3: Computations-->
<xsl:template name="rule3">
      <xsl:param name="sal"/>
      <xsl:param name="emp_id"/>
      <xsl:variable name="par">
         <xsl:value-of select="number($sal)"/>
      </xsl:variable>

      <xsl:value-of select="action:sendMsg2Stdout(concat('Rule003', ' Rule Type 3 for
Employee Id# ', $emp_id, '. The Salary is ', $par,' and the Rule is ', ' Pension is derived
as 4 % of the  Salary.',' Therefore the Pension is equal to ',  ($par * 4) div 100),
$index)" />
</xsl:template>
```

```
<!-- Rule Type 5: Action Enablers-->
<xsl:template name="rule5">
      <xsl:param name="sal"/>
      <xsl:param name="emp_id"/>
      <xsl:variable name="par">
         <xsl:value-of select="number($sal)"/>
      </xsl:variable>
      <xsl:choose>
        <xsl:when test='$par &lt; 151'>
            <xsl:value-of select="action:sendMsg2Stdout(concat('Rule004', ' Rule Type 4
for Employee Id# ', $emp_id, '. The Salary is ',  $par, ' and the Rule is ', 'Tax is
calculated as 0 % of BasicSalary upto 150 ;
10 % upto 650 ; 15 % upto 1400 ; 20 % upto 2350 ; 25 % upto 3550 ; 30 % upto 5000 ;
35 % over 5001 .',' Therefore the Tax is equal to ', $par  * 0), $index)" />
        </xsl:when>
        <xsl:when test='$par &lt; 651'>
            <xsl:value-of select="action:sendMsg2Stdout(concat('Rule004', ' Rule Type 4
for Employee Id# ', $emp_id, '. The Salary is ',  $par, ' and the Rule is ', 'Tax is
calculated as 0 % of BasicSalary upto 150 ;
10 % upto 650 ; 15 % upto 1400 ; 20 % upto 2350 ; 25 % upto 3550 ; 30 % upto 5000 ;
35 % over 5001 .',' Therefore the Tax is equal to ', ($par  - 150) * 0.1), $index)" />
        </xsl:when>
        <xsl:when test='$par &lt; 1401'>
            <xsl:value-of select="action:sendMsg2Stdout(concat('Rule004', ' Rule Type 4
for Employee Id# ', $emp_id, '. The Salary is ',  $par, ' and the Rule is ', 'Tax is
calculated as 0 % of BasicSalary upto 150 ;
10 % upto 650 ; 15 % upto 1400 ; 20 % upto 2350 ; 25 % upto 3550 ; 30 % upto 5000 ;
35 % over 5001 .',' Therefore the Tax is equal to ', ($par  - 650) * 0.15 + 50), $index)"
/>
        </xsl:when>
        <xsl:when test='$par &lt; 2351'>
            <xsl:value-of select="action:sendMsg2Stdout(concat('Rule004', ' Rule Type 4
for Employee Id# ', $emp_id, '. The Salary is ',  $par, ' and the Rule is ', 'Tax is
calculated as 0 % of BasicSalary upto 150 ;
10 % upto 650 ; 15 % upto 1400 ; 20 % upto 2350 ; 25 % upto 3550 ; 30 % upto 5000 ;
35 % over 5001 .',' Therefore the Tax is equal to ', ($par  - 1400) * 0.2 + 162.5),
$index)" />
        </xsl:when>
        <xsl:when test='$par &lt; 3551'>
            <xsl:value-of select="action:sendMsg2Stdout(concat('Rule004', ' Rule Type 4
for Employee Id# ', $emp_id, '. The Salary is ',  $par, ' and the Rule is ', 'Tax is
calculated as 0 % of BasicSalary upto 150 ;
10 % upto 650 ; 15 % upto 1400 ; 20 % upto 2350 ; 25 % upto 3550 ; 30 % upto 5000 ;
35 % over 5001 .',' Therefore the Tax is equal to ', ($par  - 2350) * 0.25 + 352.5),
$index)" />
        </xsl:when>
        <xsl:when test='$par &lt; 5001'>
            <xsl:value-of select="action:sendMsg2Stdout(concat('Rule004', ' Rule Type 4
for Employee Id# ', $emp_id, '. The Salary is ',  $par, ' and the Rule is ', 'Tax is
calculated as 0 % of BasicSalary upto 150 ;
10 % upto 650 ; 15 % upto 1400 ; 20 % upto 2350 ; 25 % upto 3550 ; 30 % upto 5000 ;
35 % over 5001 .',' Therefore the Tax is equal to ', ($par  - 3550) * 0.3 + 652.5),
$index)" />
```

```
        </xsl:when>
        <xsl:otherwise>
            <xsl:value-of select="action:sendMsg2Stdout(concat('Rule004', ' Rule Type 4
for Employee Id# ', $emp_id, '. The Salary is ',  $par, ' and the Rule is ', 'Tax is
calculated as 0 % of BasicSalary upto 150 ;
10 % upto 650 ; 15 % upto 1400 ; 20 % upto 2350 ; 25 % upto 3550 ; 30 % upto 5000 ;
35 % over 5001 .',' Therefore the Tax is equal to ', ($par  - 5002) * 0.35 + 1087.5),
$index)" />
        </xsl:otherwise>

      </xsl:choose>
</xsl:template>
</xsl:stylesheet>
```

Using DMT, when a data of the system is saved to the database, an XML file containing the data triggers the rule engine and appropriate rules that are stored in the XSL files by RMT are executed or fired for the data being saved. For a sample of XML file created, see Listing 4 below.

Listing 4: Sample XML contents generated by DMT

```
<?xml version="1.0" encoding="UTF-8"?>
<payrollData>
<ID number="3614" flag="0">
    <basicsalary>9587</basicsalary>
</ID>
<ID number="3925" flag="0">
    <basicsalary>1838.94</basicsalary>
</ID>
<ID number="5026" flag="0">
    <basicsalary>4938.12</basicsalary>
</ID>
<ID number="5159" flag="0">
    <basicsalary>15705</basicsalary>
</ID>
…
</payrollData>
```

During rule saving, RMT searches for the words that make up the rule expression being saved in the database. It also checks the appropriateness of the vocabularies in a given context.

5.3.2.3 MANAGING DATA

To manage (enter, edit, delete or browse) data of a business rules system, we use DMT. The next couple of figures illustrate the Data Management Tool while a user manages data of the two entities (i.e., Employee and MonthlyPay).

Note that the data are real data extracted from Ethiopian Airlines existing payroll system to demonstrate that the system works on a real environment.

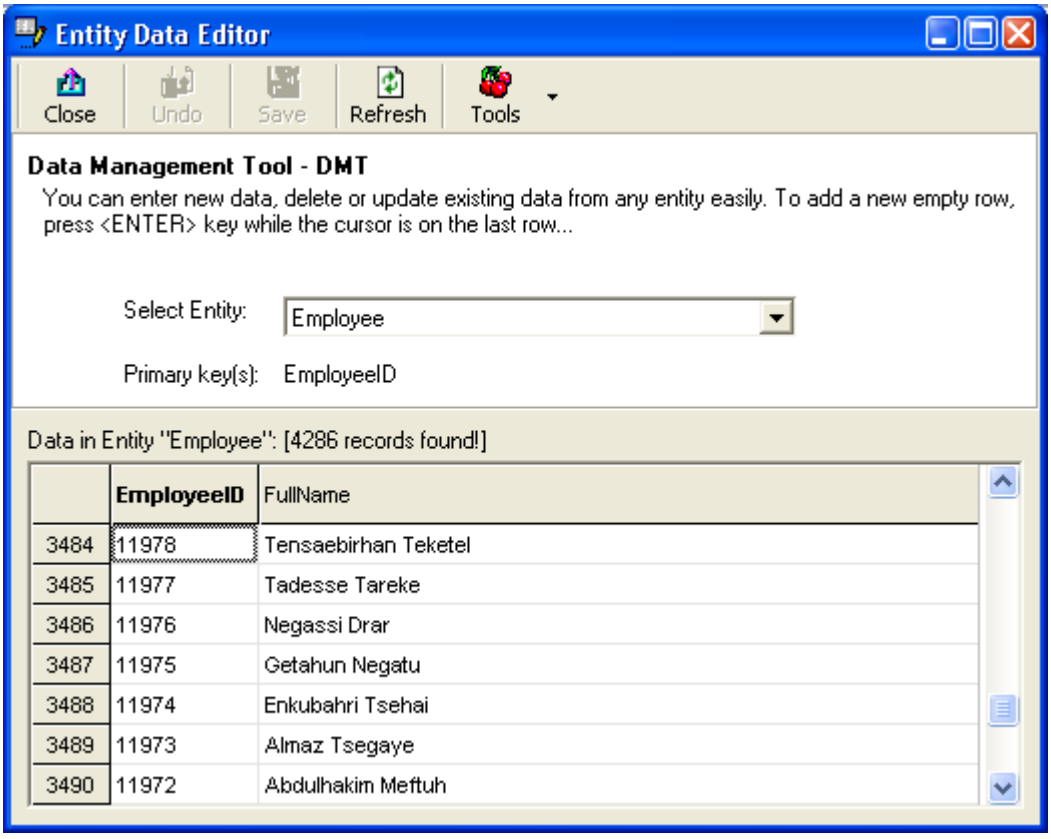


Figure 31: Data Management Tool (DMT) in action

Let us enter and save the corresponding payroll data of June 2004 as shown in the following figure.

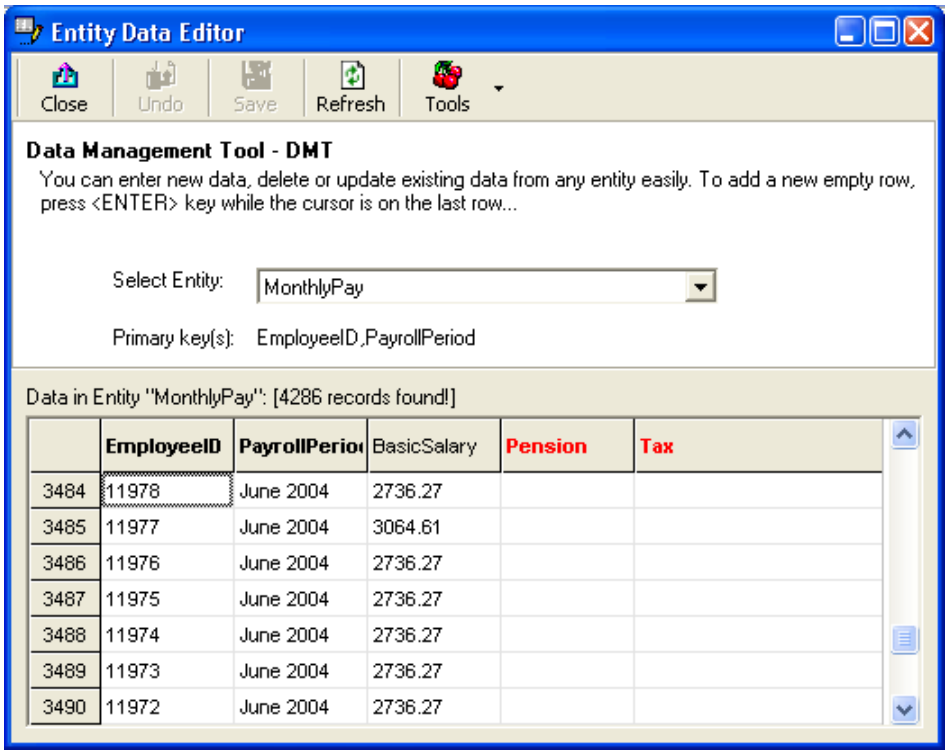


Figure 32: Entering data of MonthlyPay

After applying the rules stored in the repository system (XSL) by clicking on the **Tools/Apply Rules** and clicking on **Refresh** button, the following screen with updated Pension and Tax columns was received.

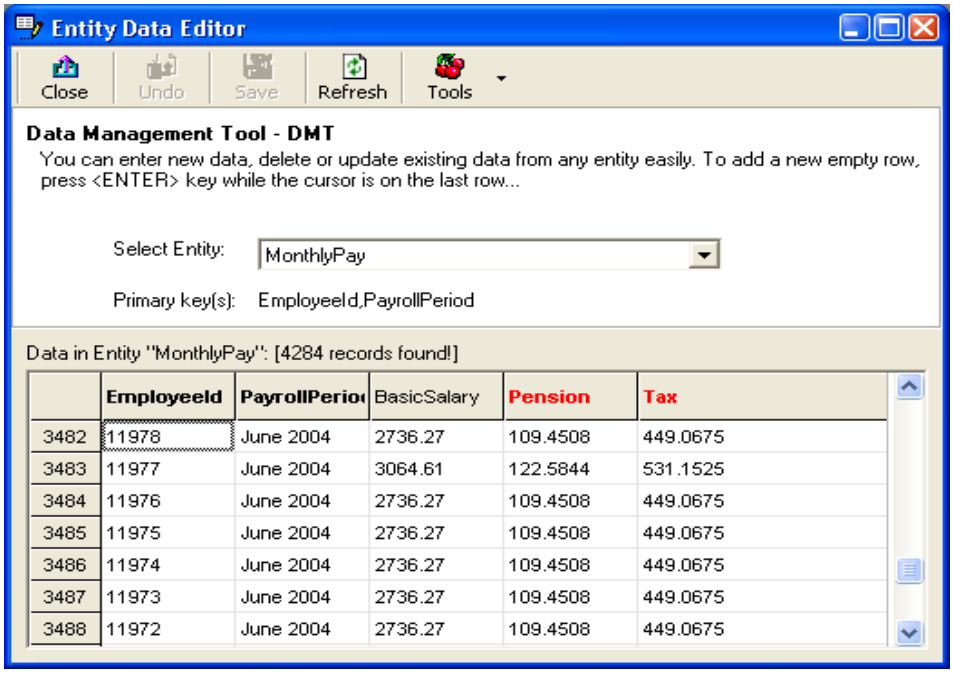
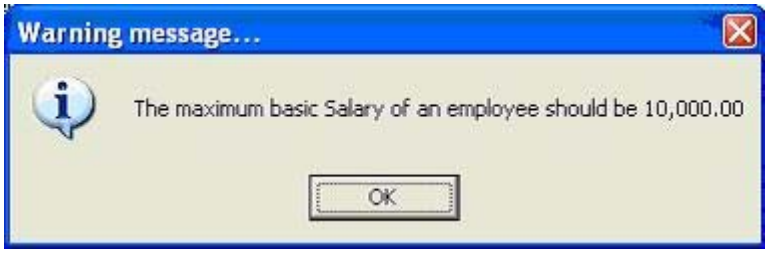


Figure 33: Applying the rule stored in the Repository system

The results (i.e., Pension and Tax calculations based on the rules stored in the repository system) are checked against the real payroll system of the Airline and found to be correct.

Warning messages were flushed by the system as presented below since there are employees who earn more than Birr 10,000.00 (by applying Rule001).



In addition, if a salary of an employee is found to be less than Birr 200.00, the following error message will appear while saving the employee record.

Pension and Tax columns of Figure 33 above are derived fields. That is, you cannot edit or change their values. They are handled by the DMT with the help of an XML file created by the rule engine via the rules stored in the XSL files created by RMT. Listing 3 and 4 above lists sample XSL and XML files that are automatically created by the RMT and DMT respectively.

The following figure shows the status of all the rules executed based on the data stored in the database.

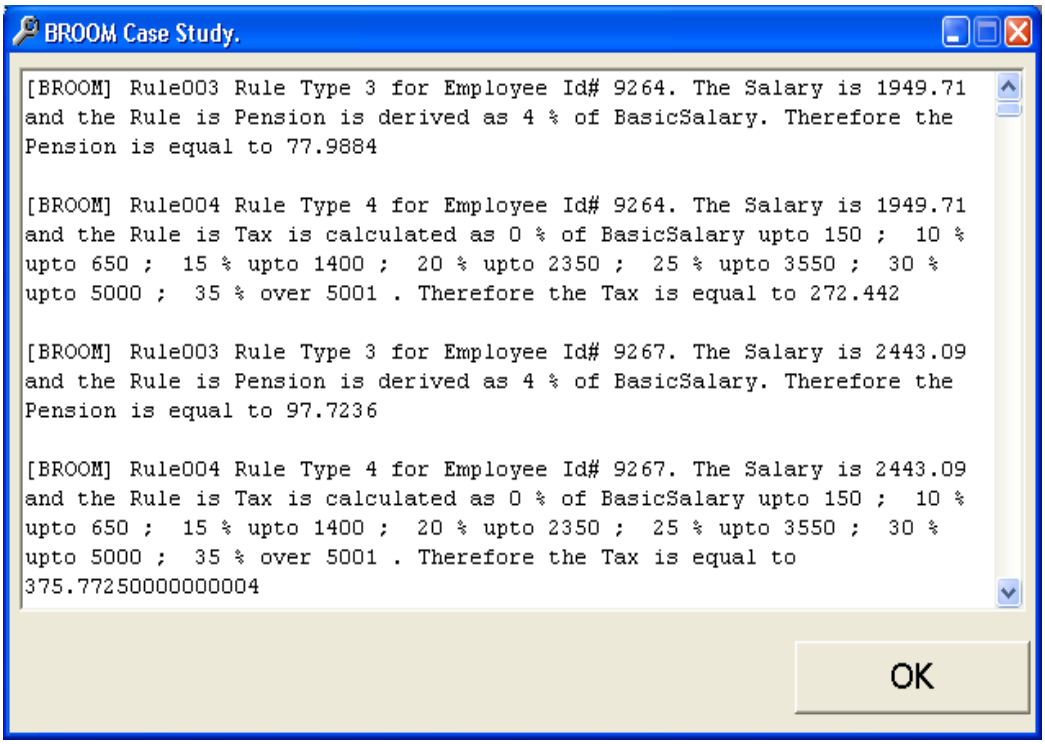


Figure 34: Partial preview of the status of the rules executed

5.3.3 AMENDING THE BUSINESS RULES

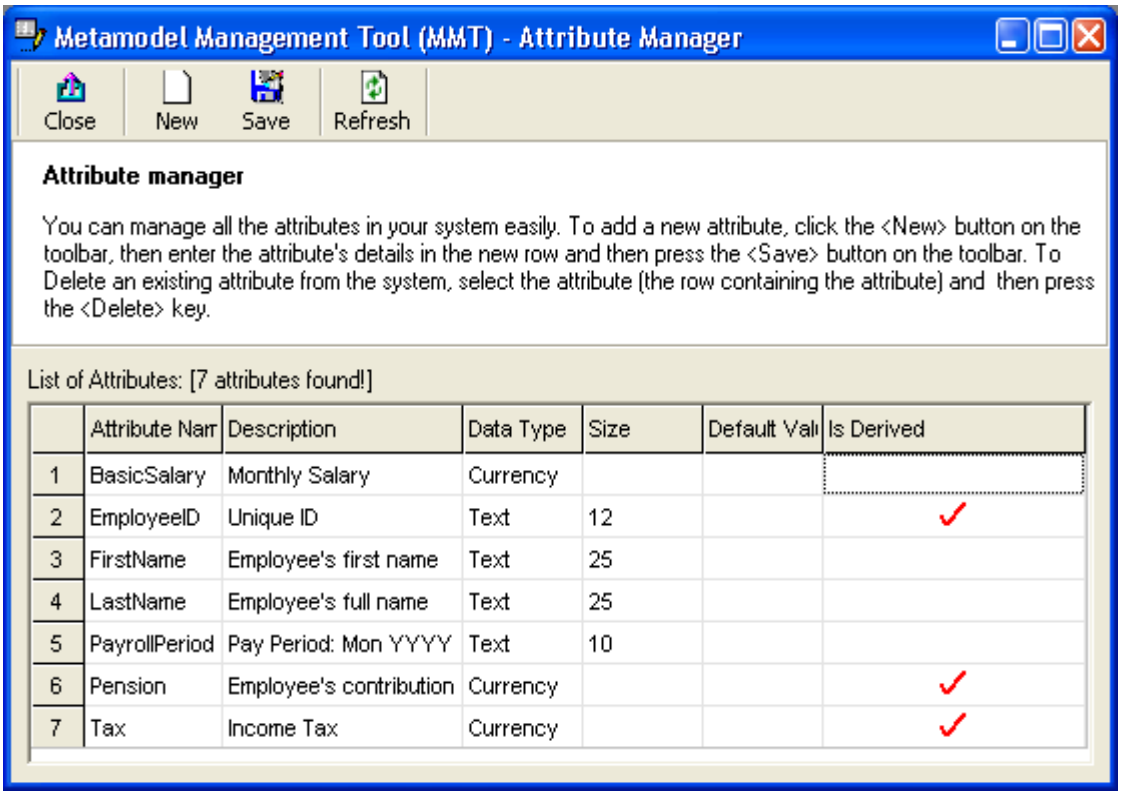
In this section, we try to see how to manage business rules requirement changes of the case study. These changes can be manifested by the change of the characteristics of a term, a fact, or a rule stored in the repository system. Two of the essential support tools are used of BROOM (i.e., MMT and RMT) to manage the requirement changes encountered in the system.

5.3.3.1 TERMS

For the sake of our demonstration, let us assume that the businesspeople found new attributes requirements for the term Employee of the system. These attributes are *Last Name* and *First Name* with length of 25 characters each while leaving out the attribute *FullName*, which was previously defined or created.

To accommodate these changes, the businesspeople of the system do not need to call a programmer so as to incorporate the changes, compile the source code and

install the object code to the users' machines. Instead, they simply use Attribute manager of the MMT to define the new attributes as shown below in Figure 35 (row 3 and row 4).



	Attribute Narr	Description	Data Type	Size	Default Vali	Is Derived
1	BasicSalary	Monthly Salary	Currency			
2	EmployeeID	Unique ID	Text	12		✓
3	FirstName	Employee's first name	Text	25		
4	LastName	Employee's full name	Text	25		
5	PayrollPeriod	Pay Period: Mon YYYY	Text	10		
6	Pension	Employee's contribution	Currency			✓
7	Tax	Income Tax	Currency			✓

Figure 35: Incorporating the changes of the system via Attribute manager

5.3.3.2 FACTS

For the next step, we use Entity Manager of MMT to define the facts (containment relationships) of the Entity and the new attributes defined above. Figure 36 below is the Entity manager with the new facts of the system. (i.e., Employee has Last Name and Employee has First Name are the new facts to be incorporated).

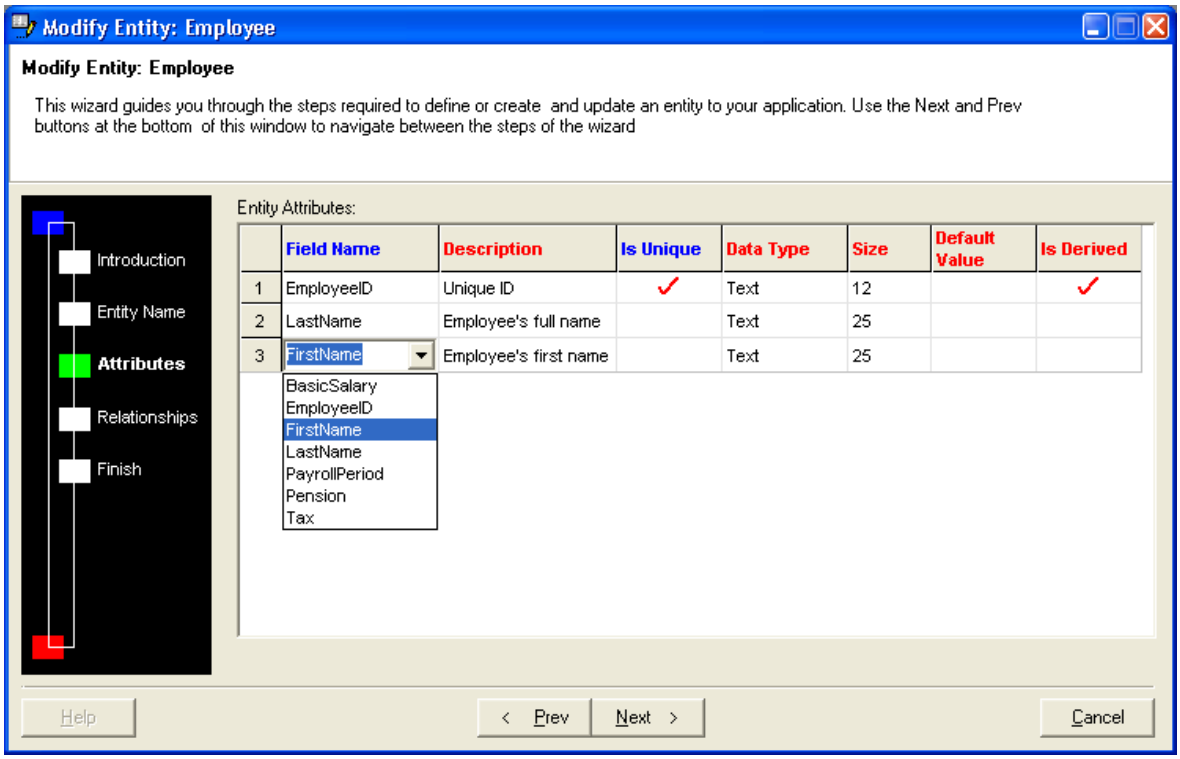


Figure 36: Incorporating the *facts* of the system via the Fact Manager

Note that attribute FullName is not shown from the facts of Entity. Furthermore, if we delete FullName and add the two fields from the fact manager, we loose the data entered previously into the attribute - FullName. However, if we rename the attribute to one of the new attributes being defined, the data will be retained. Renaming of an attribute is possible through the Attribute manager presented in Figure 35 above.

5.3.3.3 RULES

The last but not the least element of a business rule is a *rule*. Now, let us change the rules stored in the repository system and see the outcomes. Assume that the changes on the rules being made are based on the existing rules defined earlier using RMT.

The changed rules are shown in Table 20 below.

Table 20: List of amended rules of the system

No.	Rule Category	Rule Code	Rule Expression
1.	GUIDELINES	RULE001	*Minimum* BasicSalary Should be *200.00*
2.	CONSTRAINTS	RULE002	*Maximum* BasicSalary Must be *10,000.00*
3.	COMPUTATIONS	RULE003	Pension is calculated as *6 %* of the BasicSalary

In the above table, the changes from the previous rules are indicated by the italics text of the table. We use RMT to incorporate the changes of the rules of the system. The following figure shows the list of the rules of the system after adjusting the changes using RMT as in Figure 29 above.

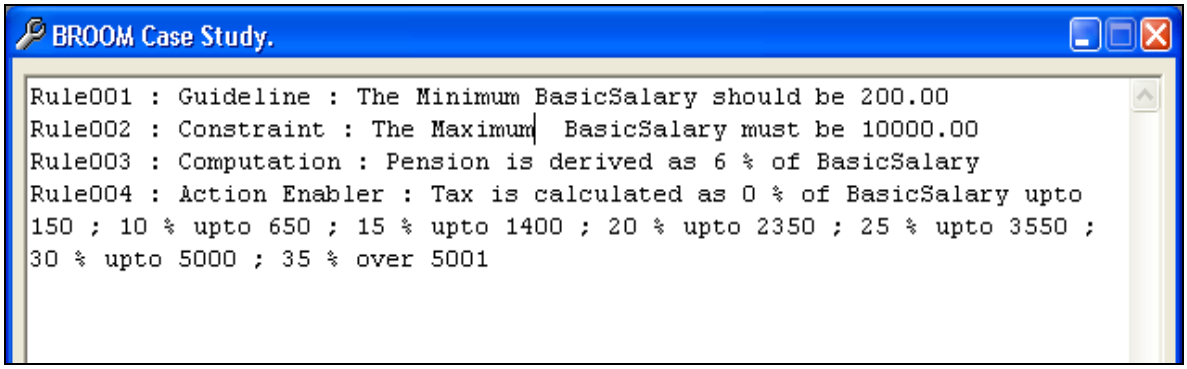


Figure 37: List of updated rules stored in the repository system

5.3.3.4 DATA

After renaming the previous attribute (i.e., FullName) to LastName, Figure 38 below illustrates the data entry screen with the new information applied to the Entity being changed.

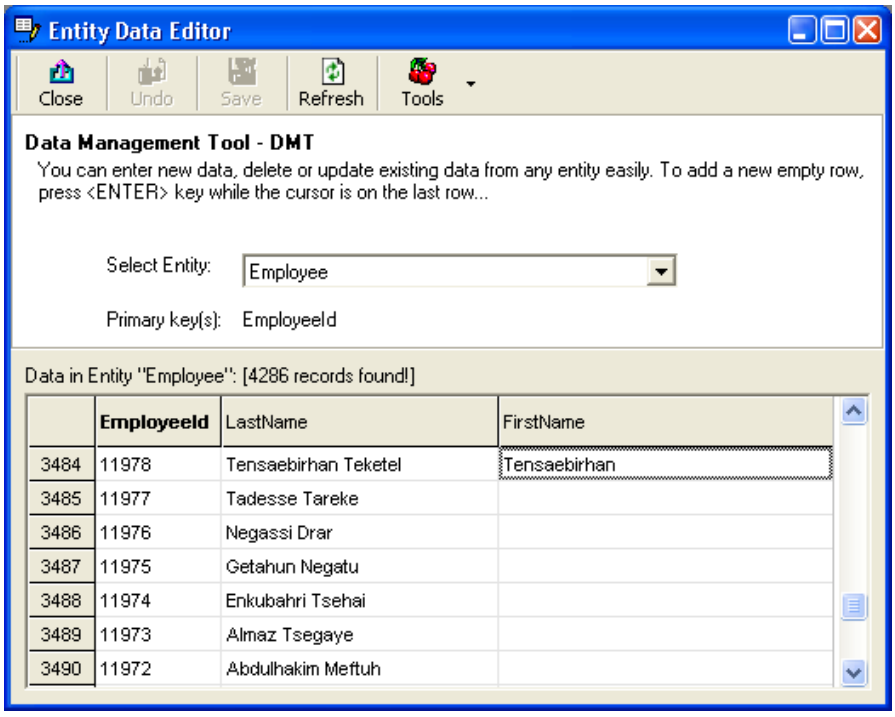


Figure 38: Employee Data Entry Screen after amending the changes

Note that data for the FullName attribute are not lost. They are rather given for the new attribute name, LastName. Therefore, users can amend the values for LastName and FirstName accordingly.

5.4 Analysis of The Results

5.4.1 Class Diagramming Issue

The UML class diagrams designed in Section 5.3.1 become obsolete whenever there is a change (i.e., addition, amendment, or deletion) in a business rule of the system since these class diagrams are based on the business rules of the system. Thus, the class diagrams illustrated in Section 5.3.1 will become useless in modeling the payroll system.

However, using the Metamodel information stored in the database, a modeling tool that can generate class diagrams similar to the diagrams such as Figure 20 and Figure 21 is feasible. This process can help us produce up-to-date class diagrams

of a system allowing the stakeholders of the system to visualize, specify, and document the artifacts of the system being developed by BROOM. It is to be noted that this tool is a member of the integrated support tools of the method.

5.4.2 VOCABULARY ISSUE

New vocabularies may arrive. These vocabularies are some business language words or phrases that are not part of the collection of terms (i.e., entities or attributes) of the system. By default, these collections of terms are the vocabularies of RMT. If a new vocabulary is required by the system, a developer (programmer) is required to introduce this vocabulary and its mapping or interpretation to the repository system syntax, as presented in Listing 3 above.

Thus, in this regard, RMT is maintained by the developer so as to meet the businesspeople requirements until it reaches its maturity. Note that MMT and DMT are rarely maintained. Basically, these tools are maintained during bug fixing or new features enhancement.

5.4.3 REPORTING ISSUE

Reporting requirements of a system is part of business rules of a system. In this research, the researcher did not try to address reporting requirements of the system. Especially those reports that are derived from two or more tables of a database. Note that we can produce a report for a table by sending the DynaGrid data to MS Excel or Internet Explorer.

Therefore, for those reports, which span more than two or more tables of a database, a reporting engine can be developed. This report engine helps us produce a required report from given tables and other required parameters. Nevertheless, until this tool is ready, a third party reporting tools such as Crystal Report can be utilized.

5.4.4 PERFORMANCE ISSUE

Rules are not compiled with the procedural codes. Besides, data is migrated from an old entity to a new entity whenever, for example, a rename of an entity (a table of the system). Hence, performance issue is prevalent for large amount of rules and data stored in the repository systems. In addition, a swapping storage space is required for the migration of such data.

The reason for applying this migration technique while we rename an entity is the inability to find SQL DDL command that helps us rename a table name of a MS Access database. If a facility for renaming a table of a database is provided by the vendor of the DBMS, the above performance issue problem will be solved. Note that this facility is provided by SQL-93 Standard.

5.4.5 CLIENT/SERVER ISSUE

The case study is tested on both a standalone PC and client/server architectures. The results of these architectures do not affect the functionality of the method. However, the issue of performance as discussed above can be a limitation while migrating data from a table of huge data.

A repository system, for example XSL, should also be placed in a server side for sharing the rules for other users and applications. The intermediate data are stored in XML files, which are normally placed in each client workstation. Thus, the XML and XSL files are referenced in the rule engine by giving the location where each one of them is residing (for example, XSL is on the Database Server and XML is a local machine where DMT is running). However, the detail results of this issue need to be further scrutinized.

5.4.6 SECURITY ISSUE

The database should only be managed by the support tools (i.e., MMT, RMT, and DMT). A DBA or programmer should not alter (add, delete, edit) any data or characteristics of a table. This is mainly because the Metamodel and the actual tables become inconsistent with each other.

Basically, to solve the stated problem, the Metamodel and the actual database tables' characteristics can be synchronized using an automated tool that traces the changes and update the metamodel database accordingly.

Nevertheless, this solution does not prevent the security record, audit trail (what, who, and when information). It can also create a performance bottleneck in checking and updating the inconsistent metamodel records of the system. Therefore, it is recommended not to alter any table outside the support tools provided by the method.

Consequently, to prevent such changes of the database of the system, there should be security mechanisms such as database password utilization and recording the audit trails for each record of the Metamodel. Moreover, these security mechanisms should be incorporated into the support tools (i.e., MMT, RMT, and DMT).

5.5 SUMMARY

In this chapter, the method that consists of the *models, framework, and support tools* (i.e., MMT, RMT, and DMT) was demonstrated with a simple payroll system of real data.

As a matter of fact, it was demonstrated that the end-users themselves could define business rules and manage business rules' changes.

In general, the results of the case study are positive since almost all the hypotheses made at the beginning of the research are supported. To remind, the hypotheses of the research made were the following:

- ✓ A lightweight and rigorous method using the strengths of both object-oriented approach and business rules approach is feasible.
- ✓ UML does not support modeling of business rules stored in a repository system. For this reason, UML needs to be extended to meet the stated requirement.
- ✓ We can have an integrated tool for the proposed method comprising tools: Metamodel Management, Rules Management, Data Management, etc.

CHAPTER 6

CONCLUSIONS

6.1 Conclusions

The area of business rules systems demands dynamic threads of business rules changes. Traditionally, developers use object-oriented approach to develop such systems in which case they take their advantages by enhancing reusability and maintainability of codes and/or components of the system. Moreover, use of modeling languages, proven design patterns and frameworks are the other advantages. However, first, the end users or businesspeople are not the direct beneficiaries of the approach. Second, sharing of business rules among the applications that require these same business rules is almost impossible.

In the meantime, to alleviate the two problems of the object-oriented approach, business rules approach has been advocated since mid 1990s. The main principle of this approach is the *separation of business rules from any procedural codes and database constructs*. Nevertheless, these advantages obtained from alleviating the problems stated above lead to some disadvantages such as unable to use UML and lightweight support tools that were not prevalent in object-oriented approach.

To that end, in this research, a lightweight method is developed by combining the strengths (i.e., advantages) of both of the approaches. This method is known as Business Rules Object-Oriented Method, which is accompanied by different models, framework, and lightweight support tools.

To conclude the method is supposed to be useful for those business rules systems, which have frequently changing business rules. In that case, there is a need to conduct the analysis phase that deals with this separation of all the business rules by classifying into their categories (i.e., terms, facts and rules).

6.2 Summary of Contributions

The following are summary of the contributions made by the research:

1. As proposed in the beginning, the two approaches (i.e., Object-Oriented Approach and Business Rules Approach) collaborate each other to develop business rules software. The outcome of this collaboration brought up a new, viable and lightweight method known as Business Rules Object-Oriented Method (BROOM, for short).

2. An extension of the standard UML class diagrams was provided so as to visualize, specify, construct and document the artifacts of a business rules system with the intention of separating the business rules from the procedural codes and database constructs. In addition, a data model was provided in order to persist business rules in a repository system. A metamodel representing business rules was also generated out of the models of the flavors of business rules.

3. A prototype for an integrated development tools known as *BROOM Essential Support Tools* (BEST) is provided for the method. BEST facilitates the development of efficient and effective business rules systems using the method.

4. Developed a general architectural framework for the method so that developers can understand the method and apply to a real world business rules systems.

5. Developed prototypes for some of the participating tools (MMT, RMT, and DMT) of the integrated tool, BEST.

6. All these models, support tools, and techniques are utilized in the general framework of the method. This framework was extended to demonstrate the method with a case study of real data, in which positive results were obtained for developing an OO business rules system of an organization.

6.3 Future Work

In order to make our method applicable in an effective and efficient way, in addition to the work already done, some additional activities are required to be addressed in the future. The following are some of the future work of this research:

1. Only UML Class Diagram was analyzed for its *necessity*, *consistency* and *sufficiency* and extended for the purpose of BROOM. Other UML Diagrams such as use case diagrams may be required for modeling business rules system. Further study on the necessity, consistency and sufficiency of such UML diagrams can be made.

2. The prototype of the integrated development tools (MMT, RMT, and DMT) are developed using VB6, the Rule Engine is in Java, the Rules Repository is in XSL and terms and facts are stored in an RDBMS. The problem with this strategy is *platform dependency*.

   Thus, BEST can be developed using an interoperable programming language such as Java to acquire cross platform functionality of the integrated tool. In addition to the MMT, RMT, and DMT, the following are the tools that can be developed as part of BEST:

   2.1 A modeling tool for creating the class diagrams of a business rules system with the Extended UML Class Diagram and the metamodel information.

   2.2 Report Engine that satisfies the different reporting requirements of a business rules system.

3. Implementing the Rule Engine API specification (JSR-94), which is a specification for a lightweight Rule Engine, along with the work of Arsanjani, and some of the results of this research for designing and implementing a rule engine for the framework of the method. The rule engine will have some features such as a derivation of a rule from more than one entity. Conflicting rules can be managed by RMT.

4. Further and rigorous evaluation of the method with other business rules systems containing real data and all rule types including the *Inference* rule.

BIBLIOGRAPHY

[1]. C. Allison. Object Persistence with Relational Databases, the Information and Communication Systems Department of the Church of Jesus Christ, www.freshsources.com/pfx/ALLISON.HTM, 1997, accessed on October 26, 2003.

[2]. S. Ambler. Design of a Persistence Layer Series Software Development Magazine. The Design of a Robust Persistence Layer for Relational Databases, Building Object Application that work, Process Patterns, 2000.

[3]. A. Arsanjani. Rule Pattern Language 2001: A Pattern Language for Adaptive and Scalable Business Rule Design and Construction, PloP2001 Conference, pp. 1-33, 2001.

[4]. M. Bajec, and M. Krisper. Managing Business Rules in Enterprises, Electrotechnical Review, 68(4): 236-241, pp. 1-6, 2001.

[5]. M. Barnes, and D. Kelly. Forget COBOL and Database Triggers, Business Rules are Moving to Middle tiers and Simpler languages, http://www.byte.com/art/9706/sec7/art1.htm, 1998, accessed on November 14, 2003.

[6]. BEA Systems Inc. Java Rule Engine API – JSR-94, Java Community Process, http://java.sun.com/jcp, 2003, accessed on April 23, 2004.

[7]. Y. Boglaev. A Design Pattern for a Rule Engine, JavaPro, Fawcette Technical Publications, 2003.

[8]. G. Booch, J. Rumbaugh, and I. Jacobson. The Unified Modeling Language (UML) User Guide, Rational Software Corporation, Addison-Wesley, 1999.

[9]. R. Bourret. XML and Database, http://www.xml.org, 2001, accessed on May 15, 2003.

[10]. B. Bruegge, and A. H. Dutoit. Object-Oriented Software Engineering, Conquering Complex and Changing Systems, Pearson Edu. Inc., 2000.

[11]. E. Colbert. Choosing the Right Object-Oriented Method, Absolute Software Co., 1993.

[12]. J. Coplien. Software Design Patterns: Common Questions and Answers, AT&T Bell Laboratories, http://wuarchive.wustl.edu/languages/smalltalk/patterns/papers/PatQandA.ps, 1995, accessed on May 15, 2004.

[13]. P. Corazza. Using the if-then-else framework, Code maintainable branching logic with the if-then-else framework. http://www.javaworld.com/javaworld/jw-03-2000/jw-0324-ifthenelse.html, accessed on March 8, 2004.

[14]. J. Demey, M. Jarrar, and R. Meersman. A Markup Language for ORM Business Rules, Proceeding of the International Workshop On Rule Markup Languages for Business Rules on the Semantic Web, pp. 1-21, 2002.

[15]. A. Eden1, and A. Yehudai1. Patterns of the Agenda, http://www.eden-study.org/articles/1997/patterns_of_the_agenda.pdf, 1998, accessed on February 16, 2003.

[16]. U. Ekstrom. Design Pattern For Simulations in Erlang/OTL, Master's Thesis in Computing Science 178, ISSN 1100-1836, 2000-11-01, Information Technology, Computing Science Department, Uppsala University, Sweden, pp. 18-50, 2000.

[17]. M. Fontoura, and C. Lucena. Extending UML to Improve the Representation of Design Patterns. Software Engineering Laboratory (LES), Computer Science Department, Pontifical Catholic University of Rio de Janeiro, Brazil, pp. 1-13, 2000.

[18]. E. Gamma, R. Helm, R. Johnson, and J. Vlissides (The Gang of Four - GOF). Design Patterns – Elements of Reusable Object-Oriented Software, Addison-Wesley Professional Computing Series, 1995.

[19]. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Abstraction and Reuse of Object-Oriented Design", presented in ECOOP '93 Conference Proceedings, Springer-Verlog Lecture Notes in Computer Science, pp. 1-21, 1993.

[20]. R. Gustavsson, J.Ala-Kurikka, and S. Rulli. Domain Specific Design Patterns. A report in the course Object-Oriented Programming, Advanced Course, Mälardalen University, pp. 1-8, 2002.

[21]. P. Haley. Changing Rules Automatically in CRM Support Systems, Technology Review, http://www.brcommunity.com/cgi-local/x.pl/resources/n005.html, visited in Oct. 01, 2003.

[22]. B. Halle. Business Rules Applied, Business Better Systems Using the Businesses Rules Approach, John Wiley & Sons, Inc. 2002.

[23]. A. Hars, and J. Marchewka. Eliciting and Mapping Business Rules to IS Design: Introducing a Natural Language CASE Tool, pp. 1-14, 1995.

[24]. D. Hay. Modeling Business Rules: What Data Models Cannot Do, Part I and Part II, Essential Strategies, Inc., The Data Administration News Letter, 2004.

[25]. H. Herbst, and T. Myrach. A Repository System for Business Rules, In Proceedings of the six IFIP TC-2 Working Conference on Data Semantics, London: Chapman & Hall, pp. 1-18, 1997.

[26]. H. Herbst, G. Knolmayer, T. Myrach, and M. Schlesinger. The Specification of Business Rules: A comparison of Selected Methodologies, Presented at the IFTP Working Group 8.1 Conference CRIS 94. Method and associated Tools for the Information System Life Cycle, pp. 1-19, 1994.

[27]. IBM, DB2 Application Development Guide, Using Triggers in an Active DBMS, https://aurora.vcu.edu/db2help/db2a0/frm3toc.htm#ToC_458, 2003, accessed on February 20, 2003.

[28]. M. Jaczynski, and B. Trousse. An Object-Oriented Framework for the Design and the Implementation of Case-Based Reasoners, INRIA Sophia-Antipolis, Action AID, FRANCE, pp. 1-10, 1998.

[29]. G. Kappel, S. Rausch-Schott, W. Retschitzegger, and M. Sakkinen. From Rules to Rule Patterns, Institute of Computer Science, Department of Information Systems, University of Linz, AUSTRIA, pp. 1-14, 1996.

[30]. J. Kienzle, and A. Romanovsky. A Framework Based on Design Patterns for Providing Persistence in Object-Oriented Programming Language, 2000.

[31]. Knowledge Partners, Inc., A KPI Position Paper. Is a business rules approach the next paradigm? http://www.kpiusa.com/ReadingRoom/BusinessRulesToday.htm, 2002, accessed on September 20, 2003.

[32]. D. Kung, H. Bhambhani, R. Shah, and G. Pancholi. An Expert System for Suggesting Design Patterns – A Methodology and a Prototype, Department of Computer Science, University of Texas at Arlington, pp. 1-25, 2002.

[33]. T. Larsson, and M. Sandberg. Building Flexible Component Based on Design Patterns, Malardalen University, Department of Computer Science, Sweden, pp. 1-9, 2000.

[34]. G. McFarland, A. Rudmik, and D. Lange. Object-oriented Database Management Systems Revisited, Characteristics of Object-Oriented Databases, http://www.dacs.dtic.mil/techs/oodbms2/oodbms2.pdf, 1997 accessed on March 15, 2004.

[35]. Melocinneide. Automated Application of Design Patterns: A Refactoring Approach. A thesis submitted to the University of Dublin, Trinity College, for the degree of Doctor of Philosophy, Department of Computer Science, Trinity College, Dublin, pp. 93-159, 2000.

[36]. R. Miller. Borland White Paper: What is New in UML2.0?, http://bdn.borland.com/article/0,1410,30168,00.html, 2003, accessed on April 14, 2004.

[37]. Object Management Group (OMG). 3rd revised submission to OMG RFP ad/00-09-01: Unified Modeling Language: Infrastructure version 2.0, www.omg.org/docs/ad/00-09-01.pdf, 2003, accessed on April 3, 2004.

[38]. Object Management Group (OMG). Business Rules Expression Request for Proposal, Draft ad/03-02-01, http://www.omg.org, 2002, visited on March 01, 2004.

[39]. Object Management Group (OMG). Introduction to OMG's Unified Modeling Language (UML), http;//www.omg.org, 2004, accessed on October 15, 2003.

[40]. Object Management Group (OMG). Model Driven Architecture (MDA) Guide Version 1.0.1, omg/2003-06-01, 2003, http://www.omg.org/cgi-bin/doc?mda-guide, March 15, 2004.

[41]. Object Management Group (OMG). Unified Modeling Language Specification for UML 1.5 formal/03-03-01found at http://www.uml.org/, 2003, accessed on January 15, 2004.

[42]. R. Paige, and J. Ostroff. A Proposal for a Lightweight Rigorous UML-Based Development Method for Reliable Systems, Department of Computer Science, York University, pp. 1-15, 2002.

[43]. B. Randell, and A. Skonnard. A Guide to XML and Its Technologies, msdn.microsoft.com/archive/ en-us/dnarxml/html/xmlguide.asp, 1999, visited on May 10, 2004.

[44]. Rational Software. A Preview of UML 2.0, http://www.uml.org.cn/ UMLSearch/pdf/MDD30_U.pdf, 2002, accessed on March 18, 2004.

[45]. G. Regev, and A. Wegmann. Goals and Business Rules as the Central Design Mechanism, pp. 1-12, 2001.

[46]. G. Reggio, and E. Astensiano. An Extension of UML for Modeling the nonPurely-Reactive Behaviour of Active Objects, pp. 1-20, 2000.

[47]. C. Richter. Designing Flexible Object-Oriented Systems with UML, Software Engineering Series, TechMedia, 2000.

[48]. M. Riebisch, K. Böllert, D. Streitferdt, and B. Franczyk. Extending the UML to Model System Families, University of Essen, Germany, pp. 1-5, 2000.

[49]. R. Ross. Principles of the Business Rules Approach, by Business Rules Solutions, LLC, Addison-Wesley, 2003.

[50]. J. Rumbaugh, I. Jacobson, and G. Booch. The Unified Modeling Language (UML) Reference, Rational Software Corporation, Addison-Wesley, 1999.

[51]. F. Sauer. Applying Design Patterns to JDBC: Building a Lightweight Object-Relational Mapping Framework, pp. 1-8, 2001.

[52]. S. Si Alhir. Extending the Unified Modeling Language (UML), UML World, home.earthlink.net/~salhir/ExtendingTheUML-UMLWorld2001.PDF, 2001, accessed on November 10, 2003.

[53]. Simonm. Design Patterns Still Aren't, http://perlmonks.thepen.com/285065.html, 2003, accessed on February 13, 2004.

[54]. J. Sodhi, and P. Sodhi. Object Oriented Methods for Software Development, Addison-Wesley, 1996.

[55]. B. Theodoulidis, and A. Youdeowei. Business Rules: Towards Effective Information Systems Development, Department of Computer Science, UMIST, UK, pp. 1-8, 1995.

[56]. A. Tichy. A Catalogue of General-Purpose Software Design Patterns, In proceedings of Technology of OO Languages and Systems, IEEE Computer Society, 1998.

[57]. M. Tilman. Position Paper for the Tools and Environments for Business Rules' Workshop of the Ecoop '98 Conference, pp. 1-8, 1998.

[58]. J. Vanthienen. Ruling the Business: About Business Rules and Decision Tables, Department of Applied Economic Science, Belgium, pp. 1-16, 2002.

[59]. J. Vlissides. Designing with Patterns, IBM T.J. Watson Research, www.research.ibm.com/designpatterns/pubs/dwp-tutorial.pdf, 1999, accessed on January 5, 2004.

[60]. J. Warmer, and K. Objecten. The Future of UML, http://www.klasse.nl /english/uml/uml2.pdf, 2003, accessed on December 15, 2003.

[61]. K. Walfgang. A Design Cookbook for Business Information Systems, Software design & management, Post PLoP Version, pp. 1-28, 1996.

[62]. J. Yoder, R. Johnson, and Q. D. Wilson. Connecting Business Objects to Relational Databases, Department of Computer Science, University of Illinois at Urbana-Champaign, 1999.

[63]. J. Yoder, F. Balaguer, and R. Johnson. Adaptive Object-Models for Implementing Business Rules, Business Rules Workshop, OOPSLA, pp. 1-10, 2001.

[64]. W. Boehm. Software Engineering Economics, Prentice-Hall, Englewood Cliffs, NJ, 1981

[65]. W. Zimmer. Experiences using Design Patterns to Reorganize an Object-Oriented Application, Karlsruhe, Germany, pp. 1-10, 1995.

GLOSSARY

Action Enabler a complete business rule statement that tests conditions upon finding them true initiates another business event, message or other activity.

Approach It is a methodology with an appropriate technology by which software development is carried out.

Attribute a specialization of *Fact* that expresses a "has property of relationship between *Terms*, specifically an association between an entity type and a domain/abstract data type. For example, "an employee has a name."

Behaviors a behavior is the basic characteristics of a class. It can be invoked as a function by the instances (objects) of the class. Sometimes it is called a method or function of a class.

BROOM See Business *Rules Object-Oriented Method*.

Business Rule a statement that defines or constrains some aspect of the business. This must be a term or fact (described below as a structural assertion), a constraint (described below as an action assertion), or a derivation. It is "atomic" in that it cannot be broken down or decomposed further into more detailed business rules. If reduced any further, there would be loss of important information about the business.

Business Rules Approach is a formal way of managing and automating an organization's business rules so that the business behaves and evolves as is its leaders intend.

Business Rules Object-Oriented Method a proposed object-oriented method for Business Rules Systems that tries to combine the two approaches: Object-Oriented and Business Rules Approach with a *lightweight* modeling language built atop of the industry modeling language standard, UML2.0, framework, and support tools.

Business Rules Repository System is a repository system whose purpose is to record and manage the information about the data models and the designs and implementations of the business rules systems of an organization.

Business Rules System simply put, a business rules system is an automated system in which the "rules" are separated (logically, perhaps physically) and shared across data stores, user interfaces and applications. In other words, it is a computerized system that assists in the management and execution of the business rules (computational and integrity validation) logic in running the business

Businesspeople the people in charge of the business rules of an organization. Other terminology for this term is end users of a business rules system.

Class Diagram a diagram that shows a collection of declarative (static) model elements, such as classes, types, and their contents and relationships. It is a most familiar and popular diagrams in

the UML. They depict the static structure and behavior of a system. That is, the purpose of a class diagram is to depict the classes within a model.

Class it is a template for a collection of similar *real world* objects. It is a class is a general concept having structural features and behavioral features. Structural features, including attributes and relationships called associations, define what elements constitute the class. An attribute is an element of information or data.

Computation it is complete statement that provides an algorithm for arriving at the value of a term where such algorithms may include *sum, difference, product, quotient, count, maximum, and minimum, average.* In other words, it is the procedure of calculating; determining something by mathematical or logical methods.

Consistency conformity in the application of something, typically that which is necessary for the sake of logic, accuracy, or fairness. For example, UML Diagrams consistency with BROOM.

Constraint a condition that determines what values an attribute or relationship can or must have.

Data Management Tool (DMT) a tool provided by BROOM. It helps the businesspeople manage the data of a business rules system based on the definition of the Metamodel of business rules stored in a relational database. It allows the businesspeople to add, edit, delete, and view records of an entity defined by the businesspeople using MMT.

Design Pattern a description of a recurrent problem and of the core of possible solutions. The idea was originally conceived by the Gang Of Four (GOF): Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.

Dynamic Attribute an attribute, which has changing characteristics and can be managed by the businesspeople or workers or the owner of the business in an organization of a system.

Dynamic Operation an operation, which has changing or dynamic characteristics and can be managed by the businesspeople or end users or the owner of the business of a system of an organization.

Dynamic Structure the changing attributes of a system.

Entity that is perceived or known or inferred to have its own distinct existence (living or nonliving).

Entity Class a thing of significance to the enterprise, about which information about the business rules is to be held.

Fact the attribution of something to describe a thing: a role it plays, or some other descriptor. It is an associating of two or more terms. It expresses a potential for association ("can be" or "may be") rather than expressing a "must be" association.

Framework a framework is a highly reusable design for an application, or part of an application, in a certain domain and often defines the basic architecture of the applications that can be built by using it. In fact, a framework can be viewed as the implementation of a system of design patterns.

Guideline a complete business rule statement that expresses a *warning* about circumstance that should be true or not true. Allowing the human to make the decision.

Inference a complete statement that *tests conditions* and upon finding them *true*, *establishes the truth of a new fact*.

Lightweight that is not heavy to use but that may contain all the necessary and required features.

Metamodel an instance of the metametamodel. It defines the model representation UML or formalisms. Examples are class, attribute, operation for UML.

Metametamodel the most abstract level. It is the metamodel definition language. It defines the concepts underlying the representation of all the other levels as well as itself. Examples are MetaClass, MetaAttribute and MetaOperation in the case of UML.

Metamodel Management Tool (MMT) a tool provided by BROOM. It helps businesspeople manage (i.e., add, edit, delete, and view) the Metamodel of business rules such as terms (i.e., attributes and entities), facts and rues of a business rules system.

Method a method is a set of principles for selecting and applying a number of analysis and synthesis (construction) techniques and tools in order to construct an efficient artifact, here software (i.e. computing system). In other context, a method can be defined as an implementation of a service or operation of a class.

Methodology a methodology is the study of and knowledge about methods, one or more.

Model an instance of metamodel. It defines the representation language of the domain under consideration. Examples are employee, organization, client and mission.

Necessity the fact of being necessary of a particular UML Diagram for modeling Business Rules Systems using a proposed method, BROOM.

Object-Oriented Method (OOM) it is an object-oriented approach that applies a set of **principles** for *selecting* and *applying* a number of *analysis* and *synthesis* (*construction)* **techniques** and **tools** in order to *construct efficient* software.

Operation a *specification* of a service or processing.

Persistence Operations save, get,/read edit, and delete operations to/from a persistent storage such as database management system.

Procedural Codes a procedural code is any code that resides in a source code of a programming language or constructs such as triggers/stored procedures of a database management system

(DBMS). A method implementation of a class of an application is an example for a procedural code. It is not synonymous to procedural languages; however, their implementation use procedural codes.

Profile a stereotyped package containing model elements that are customized for a specific domain or purpose. It refines the standard semantics by adding further constraints and interpretations that capture domain-specific semantics and modeling elements. May not add any new foundational concepts.

Repository System See Business Rules Repository System.

Reverse Engineering it is a process of translating or reversing a computer programming code into its corresponding design specifications.

Rule it is part of a business rule statement that is derived from business *terms,* and business *facts.* It can be guideline, constraint, , computation, or action enabler, or inferences.

Rule Engine a rule engine is a key underlying technology that examines rules stored in a repository system, and seeks solutions consistent with the rules.

Rule Management Tool (RMT) a tool provided by BROOM. It helps the businesspeople manage (i.e., add, edit, delete, and view) the *rules* of a business rules system based on restricted English vocabularies introduced by the programmer/developer in consultation with the businesspeople and stored in a relational database.

Semantic in Semiotics (the study of signs and symbols and their use or interpretation) semantic means more narrowly, concerned with the relationship between the signs and the objects they represent.

Static Operation Contrast: Dynamic Operation.

Static Structure Contrast: Dynamic Structure.

Stereotype a new type of modeling element that extends the semantics of the metamodel. Stereotypes must be based on certain existing types or classes in the metamodel. Stereotypes may extend the semantics, but not the structure of pre-existing types and classes. Certain stereotypes are predefined in the UML, others may be user defined. Stereotypes are one of three extendibility mechanisms in the UML. See: Profile, Metamodel.

Stereotyped Class a class extended using a stereotype extension mechanism of the UML.

Sufficiency the condition or quality of being adequate or sufficient for a particular UML Diagram in modeling Business Rules Systems using a proposed method, BROOM.

Term the application of a single definition to a word or phrase used by the business. It is a word or phrase that has a specific meaning for a business in some designated context.

The Three Amigos the three prominent *Methodologist* who conceived the UML, namely: Grady Booch, James Rumbaugh, and Ivar Jacobson.

UML Extension Mechanisms the mechanism provided by the UML so as to capture semantics significant outside the scope of the language, UML.

Understandable and maintainable business rule these are business rules that are directly accessible/manageable by the businesspeople (end users) of the system under consideration.

Unified Modeling Language (UML) the OMG industry modeling language standard used to model software and non-software things in order to visualize, specify construct and document the structures, and behaviors of a system.

User Object the first level (M0) in the UML Architecture by OMG, to represent the actual objects that a user model. It is an instance of the model. They correspond to real-world objects that are being described. Examples are Paul Martin, My Car.

INDEX

A

Action Enabler 35, 36, 38, 39, 53, 62, 65, 98

Approach 1, 3, 7, 90, 94, 95, 96, 98

Attribute 29, 30, 33, 56, 63, 70, 81, 82, 98, 99

B

Behaviors 44, 98

BROOM 3, 6, 16, 19, 21, 22, 44, 45, 47, 50, 61, 62, 66, 80, 85, 90, 91, 98, 99, 100, 101

Business Rule 1, 3, 6, 7, 19, 21, 25, 28, 31, 32, 37, 41, 42, 43, 48, 49, 51, 52, 55, 62, 63, 80, 89, 90, 92, 93, 94, 95, 96, 97, 98, 100, 101

Business Rules Approach 1, 3, 7, 90, 96, 98

Business Rules Object-Oriented Method 3, 89, 90, 98

Business Rules Repository System . 28, 32, 37, 98, 101

Business Rules System 1, 51, 98, 100, 101

Businesspeople 98

C

Class Diagram 6, 30, 34, 40, 66, 67, 68, 69, 84, 91, 98

Class 99

Computation 36, 39, 65, 99

Consistency 99

Constraint 36, 38, 53, 65, 99

D

Data Management Tool (DMT) 31, 51, 52, 58, 62, 77, 99

Design Pattern ... 3, 6, 9, 10, 11, 14, 16, 20, 48, 92, 93, 94, 95, 96, 97, 99

Dynamic Attribute .. 24, 30, 35, 41, 43, 66, 67, 68, 99

Dynamic Operation 23, 24, 41, 43, 67, 68, 99, 101

Dynamic Structure 44, 99, 101

E

Entity 27, 29, 30, 43, 56, 57, 63, 64, 66, 69, 71, 81, 82, 83, 99

Entity Class 27, 99

F

Fact 43, 56, 64, 66, 67, 82, 98, 99

Framework 3, 4, 5, 17, 45, 47, 48, 94, 96, 100

G

Guideline 36, 38, 54, 65, 100

I

Inference 36, 91, 100

L

Lightweight 95, 96, 100

M

Metametamodel 100

Metamodel 4, 5, 14, 18, 19, 20, 22, 31, 35, 42, 46, 48, 49, 50, 51, 52, 55, 56, 57, 58, 59, 62, 84, 87, 88, 99, 100, 101

Metamodel Management Tool (MMT) 31, 35, 50, 51, 52, 55, 62, 100

Method 3, 4, 15, 89, 90, 92, 94, 95, 98, 100

Methodology 94, 100

Model 9, 17, 29, 30, 33, 34, 38, 39, 55, 95, 96, 100

N

Necessity 100

O

Object-Oriented Method (OOM) 100

Operation 99, 100, 101

P

Persistence Operation ... 23, 24, 27, 31, 34, 35, 41, 43, 67

Procedural Codes 100

Profile 101

R

Repository System . 17, 19, 28, 32, 37, 45, 50, 94, 98, 101

Reverse Engineering 4, 19, 101

Rule ... 2, 16, 17, 19, 20, 38, 39, 43, 45, 46, 48, 49, 51, 53, 54, 59, 60, 65, 67, 73, 83, 91, 92, 93, 94, 98, 101

Rule Engine 20, 45, 46, 48, 49, 51, 53, 91, 92, 101

Rule Management Tool (RMT) 101

S

Semantic 93, 101

Static Operation 101

Static Structure 101

Stereotype 61, 101

Stereotyped Class 101

Sufficiency 101

T

Term 25, 27, 66, 101

The Three Amigos 102

U

UML Extension Mechanisms 13, 14, 102

Understandable and maintainable business rule . 1, 102

Unified Modeling Language (UML) .. 6, 7, 92, 95, 96, 102

User Object 102

Declaration

I, the undersigned, declare that the thesis is my original work, has not been presented for a degree in any other university, and all sources of material used for the thesis have been duly acknowledged.

Name: Tadesse Tareke Kebede

Signature: _____

Place: Faculty of Informatics, Addis Ababa University

Date of Submission: July 10, 2004

This thesis has been submitted for examination with my approval as university advisor.

_____

Dr. Yirsaw Ayalew (PhD)