

**INTEGRATED UNIFIED MODELING LANGUAGE (IUML)**

BY

**NASSER SALMAN KHASHAN**

A Thesis Presented to the  
DEANSHIP OF GRADUATE STUDIES

**KING FAHD UNIVERSITY OF PETROLEUM & MINERALS**

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the  
Requirements for the Degree of

**MASTER OF SCIENCE**

In

**COMPUTER SCIENCE**

**APRIL 2012**

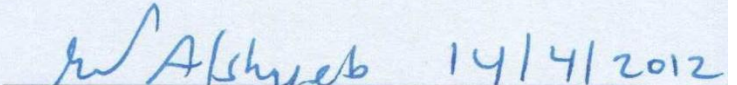


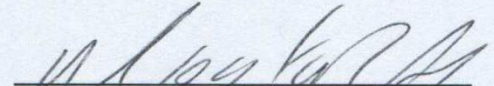
**KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS**  
DHAHRAN31261, SAUDI ARABIA

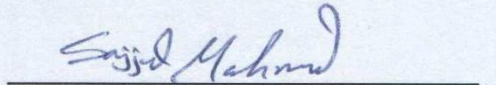
**DEANSHIP OF GRADUATE STUDIES**

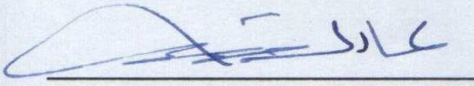
This thesis, written by **NASSER SALMAN KHASHAN** under the direction of his thesis advisor and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER SCIENCE**.

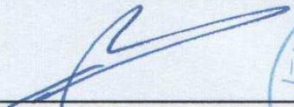
Thesis Committee

  
Dr. Mohammad Alshayeb (Thesis Advisor)

  
Dr. Moataz Ahmed (Member)

  
Dr. Sajjad Mahmood (Member)

  
Dr. Adel Ahmed  
(Department Chairman)

  
Dr. Salam A. Zummo  
(Dean of Graduate Studies)



16/4/12  
Date

# Dedication

*To my beloved family:*

*My inspirer, my parents, my brother, my sisters  
and my lovely fiancée.*

# Acknowledgements

I would like to take this chance to acknowledge King Fahd University of Petroleum and Minerals (KFUPM) for all support extended during this research.

Furthermore, I would like to deeply thank my thesis advisor, Dr. Mohammad Alshayeb, for his continuous support, unlimited help, patience, valuable guidance and advice since we first met. My gratitude is also due to the thesis committee members, Dr. Moataz Ahmed and Dr. Sajjad Mahmood for their help and enlightening comments.

I would like to thank my inspirer and my mentor (Ahmed) for inspiring me, encouraging me and for showing me the way.

I would also like to thank my parents (Salman and Ghefrah), my brother (Sameer), my sisters (Sallam and Rudinah), for their prayers, encouragement, and continuous support.

Last, but not least, I would also like to thank my fiancée (Hadeel) for her love, support, encouragement and her sincere belief in me.

# Table of Contents

	Page
<b>List of Tables</b> .....	<b>vi</b>
<b>List of Figures</b> .....	<b>viii</b>
<b>Abstract (English)</b> .....	<b>xiv</b>
<b>Abstract (Arabic)</b> .....	<b>xv</b>
<b>Chapter 1: Introduction</b> .....	<b>1</b>
<b>Chapter 2: Background</b> .....	<b>6</b>
2.1 UML.....	6
2.2 Meta-models .....	18
2.3 UML Extension Types.....	25
2.3.1 UML lightweight extension .....	25
2.3.2 UML heavyweight extension.....	28
<b>Chapter 3: Literature Review</b> .....	<b>30</b>
3.1 Class diagram.....	31
3.2 Sequence diagram .....	46
3.3 Use case diagram .....	50
<b>Chapter 4: Extension Integration</b> .....	<b>62</b>
4.1 The Integration Process .....	62
4.2 Applying the Integration Process.....	65
4.2.1 Integration of graphical symbols .....	66
4.2.2 Integration of the meta-model extensions.....	77
<b>Chapter 5: Tool Support</b> .....	<b>132</b>
<b>Chapter 6: Validation</b> .....	<b>137</b>
6.1.1 Case study # 1: Secured Health Care System .....	137
6.1.2 Case study # 2: Grade Recording System.....	150
6.1.3 Case study # 3: Meeting Scheduling System.....	159
6.2.1 Case study # 4: Elevator Control System .....	163
6.3.1 Case study # 5: E-Commerce System.....	168
6.3.2 Case study # 6: Elevator Control System .....	172
<b>Chapter 7: Conclusion</b> .....	<b>177</b>
7.1 Contribution.....	178
7.2 Threats to Validity .....	178
7.3 Future work.....	179
<b>References</b> .....	<b>181</b>
<b>VITA</b> .....	<b>184</b>

# List of Tables

Table	Page
Table 3.1: Summary of the new elements and their meanings proposed by Fontoura et al.[22] .....	31
Table 3.2: Stereotypes for Design Patterns proposed by Sanada and Adams [26] .....	35
Table 3.3: Tags in UML Profile for Design Patterns proposed by Sanada and Adams [26] .....	36
Table 3.4: Stereotypes for Frameworks proposed by Sanada and Adams [26] .....	36
Table 3.5: Tags in UML Profile for Frameworks proposed by Sanada and Adams [26] ..	37
Table 3.6: Proposed Stereotypes in Conceptual Process Design by Jantan et al. [27].....	38
Table 3.7: Summary of Navigational Access Stereotypes in ComHDM proposed by Jantan et al. [27] .....	40
Table 3.8: Mapping Rules between Navigation Design and User Interface Design in ComHDM proposed by Jantan et al. [27].....	41
Table 3.9: <<REcomponent>> as defined by Cortellessa and Pompei [32] .....	49
Table 3.10: <<REconnector>>as defined by Cortellessa and Pompei [32] .....	49
Table 3.11: <<REuser>>as defined by Cortellessa and Pompei [32] .....	49
Table 3.12: <<REservice>>as defined by Cortellessa and Pompei [32].....	49
Table 3.13: <<REhost>>as defined by Cortellessa and Pompei [32] .....	49
Table 3.14: WA-UML notations for use cases proposed by Djemaa et al. [17] .....	52
Table 3.15: UML extensions sorted by domain .....	54
Table 3.16: UML extensions sorted by type of extension.....	57
Table 3.17: UML extensions sorted by diagram .....	59
Table 4.1: Library of proposed graphical symbols (class diagram) .....	67
Table 4.2: Integrated graphical extensions.....	70
Table 4.3: Library of proposed graphical symbols (sequence diagram) .....	73
Table 4.4: Integrated graphical extension .....	74
Table 4.5: Library of proposed graphical symbols (use case diagram).....	75
Table 4.6: Integrated graphical extension .....	76
Table 4.7: The three extended functionalities proposed by Djemaa et al. 2006 [16].....	77
Table 4.8: The modeling elements of UML class diagram extensions .....	78
Table 4.9: Mapping iUML class diagram graphical symbols into the meta-model .....	100
Table 4.10: The modeling elements of UML sequence diagram extensions .....	106
Table 4.11: Mapping iUML sequence diagram graphical symbols into the meta-model	114
Table 4.12: The modeling elements of UML use case diagram extensions.....	118
Table 4.13: Mapping iUML use case diagram graphical symbols into the meta-model..	127
Table 6.1: Excerpt of iUML library .....	139
Table 6.2: Different types of Hospital’s records .....	142
Table 6.3: iUML security roles and levels .....	143
Table 6.4: Excerpt of iUML library .....	152
Table 6.5: iUML modeling elements (stereotypes).....	155

Table 6.6: Excerpt of iUML library .....	160
Table 6.7: iUML modeling elements (stereotypes).....	164
Table 6.8: Excerpt of iUML library .....	169
Table 6.9: iUML modeling elements (stereotypes).....	173

# List of Figures

Figure	Page
Figure 2.1: UML 2.4.1 diagrams .....	8
Figure 2.2: Class icon .....	10
Figure 2.3: Composition relationship .....	11
Figure 2.4: Aggregation relationship .....	11
Figure 2.5: Association relationship .....	12
Figure 2.6: Inheritance relationship .....	12
Figure 2.7: Dependency relationship .....	13
Figure 2.8: Class diagram for course registration and library systems .....	13
Figure 2.9: Object's lifeline .....	14
Figure 2.10: Messages between objects .....	15
Figure 2.11: Guarded message .....	15
Figure 2.12: Sequence diagram for Check handling system .....	16
Figure 2.13: Use case icon .....	16
Figure 2.14: Actor icon .....	17
Figure 2.15: Association link between Actor and Use case .....	17
Figure 2.16: Use case diagram for e-commerce website .....	18
Figure 2.17: Four-layered UML architecture .....	20
Figure 2.18: Class diagram meta-model .....	22
Figure 2.19: Sequence diagram meta-model .....	23
Figure 2.20: Use case diagram meta-model .....	24
Figure 2.21: Stereotype .....	26
Figure 2.22: Tagged value .....	26
Figure 2.23: GUI Profile proposed by Cabot et al. [20] .....	27
Figure 2.24: Example of using GUI profile proposed by Cabot et al. [20] .....	27
Figure 2.25: Dependencies between packages as presented by Przybylek [21] .....	29
Figure 3.1: UML-F extended class diagram proposed by Fontoura et al. [22] .....	32
Figure 3.2: Graphical representation of class meta-model element proposed by Byeon et al. [23] .....	33
Figure 3.3: Example of geo-referenced class presented by Byeon et al. [23] .....	33
Figure 3.4: Class diagram integrated with UMLpac for security features proposed by Peterson et al. [24] .....	34
Figure 3.5: RClass proposed by Mahmood and Lai [28] .....	43
Figure 3.6: CClass proposed by Mahmood and Lai [28] .....	43



Figure 3.7: Satisfy mapping relationship proposed by Mahmood and Lai [28].....	44
Figure 3.8: Crosscutting Bar and Invocation with Crosscutting Bar proposed by Zhou et al. [14] .....	47
Figure 3.9: Alarm use case proposed by Fei and Yan [16] .....	51
Figure 3.10: Actors of WA-UML proposed by Djemaa et al. [17].....	52
Figure 4.1: First part of original UML class diagram meta-model elements and integrated elements.....	81
Figure 4.2: Second part of original UML class diagram meta-model elements and integrated elements.....	82
Figure 4.3: Third part of original UML class diagram meta-model elements and integrated elements.....	83
Figure 4.4: Stereotype and Tagged Value categories applied to the first part of modeling elements.....	85
Figure 4.5: Stereotype and Tagged Value categories applied to the second part of modeling elements.....	86
Figure 4.6: Stereotype and Tagged Value categories applied to the third part of modeling elements.....	87
Figure 4.7: Meta-classes defined in the first part of modeling elements .....	89
Figure 4.8: Meta-classes defined in the second part of modeling elements.....	90
Figure 4.9: Meta-classes defined in the third part of modeling elements .....	91
Figure 4.10: The first part of integrated domain model elements .....	93
Figure 4.11: The second part of integrated domain model elements .....	94
Figure 4.12: The third part of integrated domain model elements .....	95
Figure 4.13: Crosscutting Feature derivation .....	96
Figure 4.14: Przybyłek’s UML heavyweight extension mechanism in [21].....	97
Figure 4.15: UML lightweight extension mechanism by Sharafi et al. in [28].....	98
Figure 4.16: Excerpt of iUML class diagram meta-model.....	99
Figure 4.17: First part of iUML class diagram meta-model.....	103
Figure 4.18: Second part of iUML class diagram meta-model .....	104
Figure 4.19: Third part of iUML class diagram meta-model .....	105
Figure 4.20: First part of original UML sequence diagram meta-model elements and integrated elements.....	108
Figure 4.21: Second part of original UML sequence diagram meta-model elements and integrated elements.....	109
Figure 4.22: Categorizing first part of elements as Stereotypes and Tagged Values.....	111
Figure 4.23: Categorizing second part of elements as Stereotypes and Tagged Values ..	112
Figure 4.24: Boolean type classifier.....	113
Figure 4.25: First part of iUML sequence diagram meta-model.....	116
Figure 4.26: Second part of iUML sequence diagram meta-model .....	117
Figure 4.27: First part of original UML use case diagram meta-model elements and integrated elements.....	120
Figure 4.28: Second part of original UML use case diagram meta-model elements and integrated elements.....	121
Figure 4.29: Third part of original UML use case diagram meta-model elements and integrated elements.....	122

Figure 4.30: Stereotype category of the first part of elements .....	124
Figure 4.31: Stereotype category of the second part of elements .....	125
Figure 4.32: Stereotype category of the third part of elements .....	126
Figure 4.33: First part of iUML use case diagram meta-model .....	129
Figure 4.34: Second part of iUML use case diagram meta-model.....	130
Figure 4.35: Third part of iUML use case diagram meta-model.....	131
Figure 5.1: Environment of Dia .....	132
Figure 5.2: Properties of Class .....	133
Figure 5.3: iUML sheet .....	134
Figure 5.4: iUML integrated classes created using Dia .....	135
Figure 5.5: iUML class diagram example created using Dia .....	136
Figure 6.1: Excerpt of iUML class diagram meta-model.....	140
Figure 6.2: Excerpt of iUML class diagram meta-model.....	141
Figure 6.3: Hierarchy of users as suggested by Fernandez-Medina et al in [12].....	141
Figure 6.4: Levels of security as suggested by Fernandez-Medina et al. in [12].....	142
Figure 6.5: iUML security tile # 1.....	143
Figure 6.6: iUML security tile # 2.....	144
Figure 6.7: iUML security tile # 3.....	144
Figure 6.8: iUML security tile # 4.....	144
Figure 6.9: iUML security package (Secure Access).....	145
Figure 6.10: iUML security package (Secure Attribute Access) .....	145
Figure 6.11: iUML classes Admission created using Dia .....	145
Figure 6.12: Integrated UML class diagram (Secured Health Care System).....	147
Figure 6.13: iUML security package.....	148
Figure 6.14: iUML classes Admission created using Dia .....	148
Figure 6.15: iUML stereotypes .....	149
Figure 6.16: Excerpt from the integrated class diagram meta-model .....	153
Figure 6.17: iUML classes' design inspired by Byeon et al. [22] created using Dia .....	153
Figure 6.18: The GPA requirement class created using Dia .....	154
Figure 6.19: The Student component class created using Dia .....	154
Figure 6.20: RSatisfy relationship created using Dia.....	155
Figure 6.21: iUML UML class diagram (Grade Recording System).....	156
Figure 6.22: iUML student class created using Dia .....	157
Figure 6.23: Excerpt from the integrated class diagram meta-model .....	160
Figure 6.24: Class diagram for the Meeting system.....	161
Figure 6.25: Integrated UML class diagram (Meeting Scheduling System).....	162
Figure 6.26: Excerpt from the integrated sequence diagram meta-model .....	165
Figure 6.27: Sequence diagram for Select Destination System .....	166
Figure 6.28: Integrated UML sequence diagram (Elevator Control System) .....	167
Figure 6.29: Excerpt from the integrated use case diagram meta-model.....	170
Figure 6.30: E-commerce system environment.....	171
Figure 6.31: The verification process.....	171
Figure 6.32: Excerpt from the integrated use case diagram meta-model.....	174
Figure 6.33: Use case diagram for Select Destination System.....	174
Figure 6.34: Extended use case diagram .....	175

Figure 6.35: Integrated UML use case diagram (Elevator Control System)..... 175

# Abstract

**Name:** Nasser Salman Khashan  
**Title:** Integrated Unified Modeling Language (iUML)  
**Major Field:** Computer Science  
**Date of Degree:** April 2012

*The Unified Modeling Language (UML) is one of the most commonly used modeling languages in the software industry. It simplifies the complex process of design by providing a set of graphical notations which helps expressing the object-oriented analysis and design of software projects. Although UML is applicable to different types of systems, domains, methods and processes, it was found unable to express certain problem domain needs. Researchers realized that UML is not enough to model all aspects of software, therefore, many researchers proposed extensions to UML. In this thesis, we propose a framework for integrating the UML extensions and by using the framework we propose an Integrated Unified Modeling Language (iUML) that integrates the existing UML extensions into one integrated form. This includes an integrated diagram for UML class, sequence and use case diagrams and also includes modifications to the UML meta-model as a result of the integrated diagrams. In addition to that, a number of case studies were developed in order to validate the proposed iUML and build UML system models.*

## ملخص الرسالة

الاسم: ناصر سلمان خشان

عنوان الرسالة: لغة النمذجة الموحدة المتكاملة

التخصص: علوم الحاسب الآلي

تاريخ التخرج: أبريل ٢٠١٢

لغة النمذجة الموحدة هي واحدة من أكثر لغات النمذجة المستخدمة في سوق البرمجيات. تقوم هذه اللغة بتسهيل عملية التصميم المعقدة عن طريق توفير مجموعة من الرموز الرسومية والتي تساعد في نمذجة التحليل و التصميم للبرمجيات الموجهة نحو الهدف. و مع أن لغة النمذجة الموحدة يمكن تطبيقها على عدة نظم نطاقات, مناهج و عمليات, إلا أنها قد وُجدت غير قادرة على نمذجة متطلبات بعض النطاقات. فقد أدرك الباحثون أن لغة النمذجة الموحدة لا تكفي لنمذجة جميع جوانب البرمجيات, لذا, اقترح العديد من الباحثين ملحقات إلى لغة النمذجة الموحدة. في هذا البحث, نقترح إطار لضم ملحقات لغة النمذجة الموحدة وباستخدام هذا الاطار نقترح لغة النمذجة الموحدة المتكاملة والتي تربط ملحقات لغة النمذجة الموحدة الموجودة في نموذج واحد متكامل. وهذا يشمل اقتراح رسم متكامل لكل من رسم الصنفيات, التابع و وقائع الاستخدام الخاص بلغة النمذجة الموحدة والذي سيتضمن ملحقاتهم وكذلك يتضمن تعديلات مقترحة على نموذج الفوقية للغة النمذجة الموحدة كنتيجة للرسومات المتكاملة المقترحة. بالإضافة إلى ذلك, تم بناء عدة دراسات لحالات من أجل التحقق من صحة لغة النمذجة الموحدة المتكاملة وبناء نماذج نظم باستخدامها.



# CHAPTER 1

## 1. Introduction

The Unified Modeling Language (UML) [1] is a modeling language used to specify, visualize, construct and document the aspects of system-development process. UML gained a lot of popularity in the software industry due its unique ability to capture, communicate and model knowledge. UML also is applicable to different types of systems, domains, methods and processes which puts it on top of the modeling languages list. It was originally created by Grady Booch, James Rumbaugh and Ivar Jacobson from Rational Software Corporation [2]. The language then got approved by the Object Management Group (OMG) [3] as a standard in 1997.

Although UML provides a set of graphical notations that help in expressing the object-oriented analysis and design of software projects, yet some software engineers and designers found that UML was unable to cover some problem domains. For that reason, UML allows its users to customize it to address the desired problem domains. This is done by UML extensions mechanisms which enable UML to be more adapted to a variety of different systems, domains, methods and processes. These mechanisms allow the user to leverage the existing UML specifications to the desired level, hence, making modeling easier. In the meantime, the extension has to be sufficient and consistent in order to extend UML in a robust manner. In that sense, people behind this extension must understand the

accepted conceptual framework for modeling, UML's extension mechanisms and the governing rules and the proper application of such extensions.

There are two types of UML extension mechanisms; UML lightweight extension and heavyweight extension. UML lightweight extension involves using profiles. UML profile defines limited extensions to the meta-model elements. It uses three main constructs; stereotypes, tag definitions and constraints. This type of UML extensions mechanisms is a simple and straightforward mechanism for customizing existing UML modeling elements to a particular domain. It does not change UML behavior but it can add to or modify UML structure.

The second type is UML heavyweight extension; it involves the reuse technique of UML package. It also involves two steps; selecting the desired modeling elements that one wants to extend, and merging them with the elements from the targeted problem domain. It can customize UML behavior and operations but its development is difficult and costly.

Deciding whether to extend UML lightly or heavily depends on two issues: the nature of the problem domain and the intended use of the extended model. UML lightweight extension would be the perfect choice if the user wants simple customization to UML; adding new modeling elements, setting new properties or modifying existing ones, etc. On the other hand, if the user wants to extend the behavior of UML, restrict a set of modeling elements and other complex issues, then the heavyweight extension would be a better choice.

UML extensions, in general, add new terminologies, properties and define new semantics in order to make the language suitable to a specific problem domain. The problem is after

extending UML; it becomes only suitable for a specific domain, which may make it unusable for other domains even if they differ in small details. In this research we propose a framework for integrating the UML extensions and by using the framework we integrate the available UML extensions in the literature to form an integrated UML (iUML). The motivation for this research is to reduce the time and effort invested during modeling the targeted system using UML extensions. iUML saves a lot of time and effort when it comes to modeling since it provides one integrated form for all required problem domains, and secondly iUML provides the designers with a flexible way to model the targeted systems. iUML provides a one comprehensive set of graphical and meta-model concepts that is ready to model any domain or multiple domains at the same time.

The surveyed extensions address certain problem domains, hence, solve particular problems. Domains vary from security software designs, aspect-oriented modeling to component-based software systems and data warehouse modeling. Each extension proposed new modeling elements, properties, constraints and mapped them to UML specifications in order to make UML suitable for the targeted domain. This work integrates the introduced elements from the extensions into one diagram and one meta-model for each model type.

At first, the extensions, whether they were done lightly or heavily, were selected and studied carefully to make sure no problems will be caused from the integration process. In this research, we considered only the extensions that are made to three UML diagrams in the integration process. These UML diagrams are; class, sequence and use case diagrams. There are two reasons why those diagrams were the only selected diagrams; first, each UML diagram represents a different view of the modeled system. The class diagram

describes the system's structure, the sequence diagram describes the system's behavior and the use case describes the system functionality. The second reason is that most of the extensions found in the literature are applied to those diagrams.

The integration process is applied to two different types of extensions; the first type addresses UML extensions that provided graphical symbols only. The second type goes beyond the graphical representations in UML diagrams and deals with the proposed modeling elements that add to the meta-models. For example, the graphical modifications to UML class diagram are integrated all at once. The next step is to take the modifications deeper, to the next level, i.e. the meta-model level. Each graphical modification cast its shadow on the meta-model. In other words, the non-conflicting extended meta-model elements are integrated. At the end of the second type of integration, the obtained graphical elements are checked for consistency. Each graphical symbol is mapped into iUML meta-model.

In order to validate the integrated model, a number of case studies are used. These case studies put the introduced iUML under test to make sure that it covers the wide range of domains effectively. The whole idea behind the validation process is to provide some kind of practical proof that the iUML is capable in an effective way to solve problems and cover domains in a way where UML is not. Each case study combines a number of problem domains and applies the integration process, in its two stages, to show how the iUML works and how this new integrated form can be applied to a number of domains.

The rest of this thesis is organized as follows: Chapter 2 gives a background on UML, UML extension types and meta-models. Chapter 3 surveys the literature for UML

extensions. Chapter 4 explains the integration process. Chapter 5 discusses the tool support for this work. Chapter 6 provides the validation to iUML using a number of case studies. Finally, Chapter 7 discusses the concluding points, threats to the validity of iUML and future work.



# CHAPTER 2

## 2. Background

This chapter gives a background on UML, UML extension types and meta-models.

### 2.1 UML

Modeling languages are artificial languages that express information in graphical or textual format. This information, whether it is graphical or textual, is driven by a set of rules. The graphical modeling languages use diagrams with modeling elements like symbols and lines, symbols represent the introduced concepts and lines represent relationships between these concepts. On the other hand, textual modeling languages use a set of well-defined keywords set by parameters to synthesize computer-interpretable expressions.

Graphical modeling languages have been available in the software industry for a long time [4]. Unlike programming languages, these languages are used due to their high level of abstraction that can aid discussions and analyses about software design. Some examples of such languages are; *EXPRESS* [5], is a standard general-purpose data modeling language that displays entity and type definitions, relationships and cardinality. *Behavior Trees* [6] is another formal, graphical modeling language that represents natural language requirements to express the stakeholders requirements needs for software-integrated system.

Unified Modeling Language (UML) [1] is a graphical modeling language used to model the analysis and the design of software systems. UML simplifies the complex process of design by providing a set of graphical notations which helps expressing the object-oriented analysis and design of software projects. In that sense, UML helps acquire an overall view of the system. UML is maintained by object management group (OMG) [3], it combines three famous modeling notations: Booch method [7], Rumbaugh's Object Modeling Technique (OMT) [8] and Jacobson's Object Oriented Software Engineering (OOSE) [9].

Fowler explained in his book [4] that there are three modes in which UML can be used: sketch, blue print and programming language. The essence of sketching is selectivity. With sketching, a team of designers can meet and write some issues in code. The ultimate aim is to use the sketches to help deliver ideas. As blueprint, UML revolves around completeness. Developed blueprints help the programmer to do the coding. In other words, the design decisions should be stated so the programmer can follow them. UML as a programming language, the developers build UML diagrams that can be compiled directly to executable code, hence, UML becomes the source code.

UML contains a variety of diagrams types. The current UML version (version 2.4.1) contains 14 diagrams [10] divided into two categories: structural and behavioral diagrams as shown in Figure 2.1 [10].

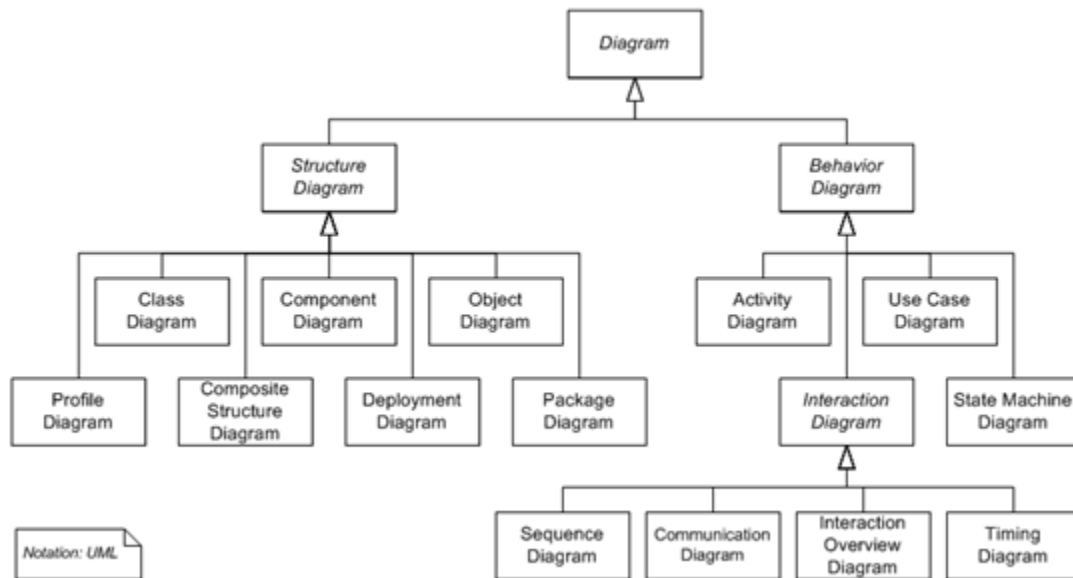


Figure 2.1: UML 2.4.1 diagrams

## The Structure diagrams

These diagrams describe the required elements in the system. They focus on the overall structure of the modeled system.

- 1- **Class diagram:** depicts the system's structure using classes. Furthermore, it shows the attributes of such classes and their relationships.
- 2- **Component diagram:** describes the division of a system into a number of components and displays the dependencies among these components.
- 3- **Composite structure diagram:** shows the internal structure of a class and the possibility of collaborations.
- 4- **Deployment diagram:** models the hardware-related artifacts of the system by showing the system's implementations and the execution environments.
- 5- **Object diagram:** shows complete or partial views of the structure of the targeted system.
- 6- **Package diagram:** depicts the split of packages by pointing out the dependencies between these packages.

- 7- Profile diagram:** works at the meta-model level to show the introduced stereotypes and profiles.

### **The Behavior diagrams**

Behavior diagrams are divided into two groups: behavior diagrams that represent the functionality of the system (activity diagram, state machine diagram and use case diagram) and interaction diagrams that focus on the flow of data and control among the parts of the system (communication diagram, interaction overview diagram, sequence diagram and timing diagrams).

- 1- Activity diagram:** shows step-by-step activities of the system. It shows the complete, overall flow of control.
- 2- State machine diagram:** describes the behavior of the system in a number of states.
- 3- Use case diagram:** shows the functionality of a system in terms of actors, use cases, and the dependencies between those use cases.
- 4- Communication diagram:** shows the interactions between objects in terms of sequential messages. These messages represent the static structure and dynamic behavior of a system.
- 5- Interaction overview diagram:** is a type of activity diagram that describes nodes and represent them as interaction diagrams.
- 6- Sequence diagram:** shows the interaction of objects through messages. It also shows the life spans of related objects.

**7- Timing diagram:** describes and focuses on the timing constraints placed over the components of a modeled system. It is a specific type of interaction diagram.

In this research, only three UML diagrams are considered; class, sequence and use case diagrams. The reason behind this consideration is because each one of these diagrams represents a different view of the modeled system. The class diagram describes the structure of the system. On the other hand, the remaining two diagrams focus on the behavior but more precisely, the sequence diagram emphasizes the interactions that happen between the objects of the system while the use case diagram focuses on the provided functionality of the modeled system. In addition to that, those three UML diagrams are the most popular ones in the literature and this research's nature of work is an integration effort, so the most common UML diagrams are to be considered. The next three sections, section 2.1.1-2.1.3, discuss the class, sequence and use case diagrams in more details.

### **2.1.1 UML class diagram**

The class diagram depicts the structure of a modeled system through a number of classes. These classes have attributes, operations and relationships with other classes. Figure 2.2 shows the main elements of the class.

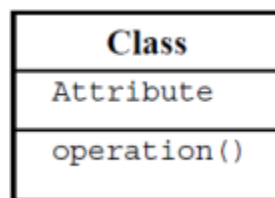


Figure 2.2: Class icon



It is a three-compartment rectangle. The first compartment contains the name of the class. The second one contains the attributes that the class has, and the last one contains the included operations.

Classes interact with each other through relationships. There are a number of relationships that happen between classes:

**2.1.1.1 Composition relationship:** denotes that one class is composed of or contains another. Figure 2.3 shows the composition relationship.



Figure 2.3: Composition relationship

The lifetime of class Point depends entirely on the lifetime of class Line.

**2.1.1.2 Aggregation relationship:** represents the whole/part relationship. Figure 2.4 denotes that the class Chair is the whole and the class Shape is the part. Figure 2.4 shows an aggregation relationship between class Chair and class Shape. In other words, class Chair has many Shape instances.



Figure 2.4: Aggregation relationship

**2.1.1.3 Association relationship:** denotes the standard relationships that happen between classes. Its indication is mostly simple. Figure 2.5 shows that Student can have zero or more Courses and a Course can have 20 students only.

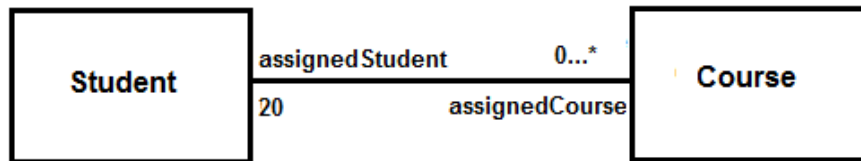


Figure 2.5: Association relationship

**2.1.1.4 Inheritance relationship:** happens between classes in the class diagram. Figure 2.6 shows class Child inherits the attributes and operations from class Father.

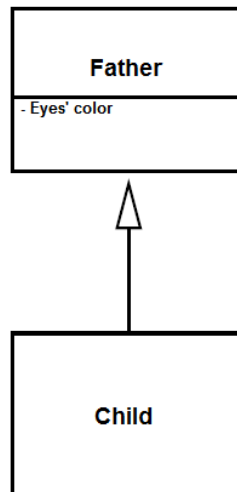


Figure 2.6: Inheritance relationship

**2.1.1.5 Dependency relationship:** states that one class depends on another class. Figure 2.7 shows that class Refrigerator depends in its operation "Operate" on class Electricity.

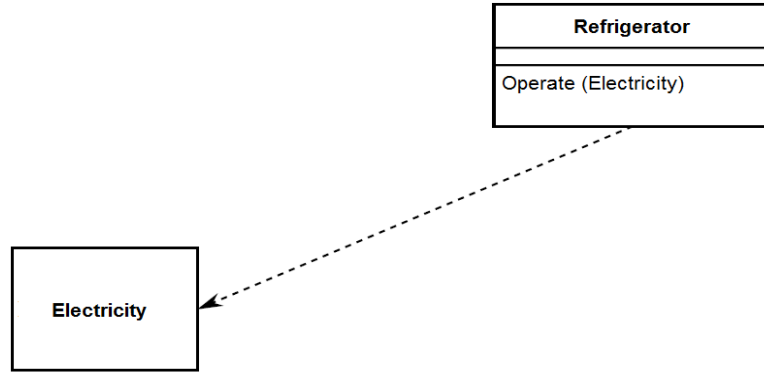


Figure 2.7: Dependency relationship

Figure 2.8 displays a UML class diagram that describes part of a course registration system and a library system.

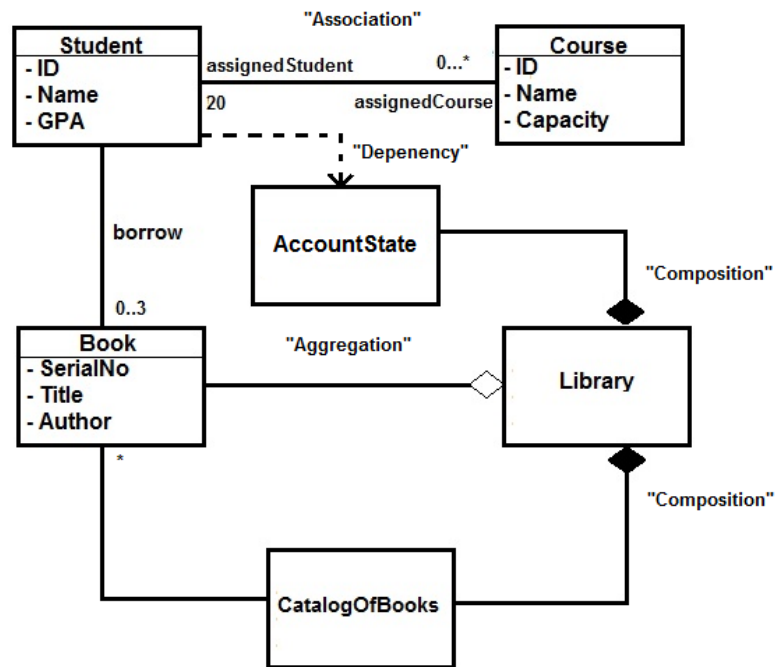


Figure 2.8: Class diagram for course registration and library systems

## 2.1.2 UML sequence diagram

The sequence diagram shows the interactions that happen between the system's objects in a sequential order. It focuses more on the order of the messages rather than the messages themselves.

One of the key elements of the sequence diagram is the lifeline. It represents the roles or object instances in the system. Figure 2.9 shows an instance "Accountant" with its lifeline descending from its containing box.

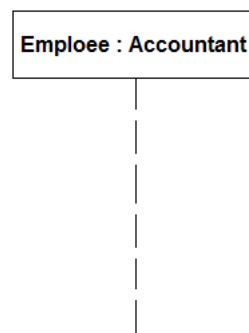


Figure 2.9: Object's lifeline

The messages that are sent and received by the objects represent means of interaction. They represent methods or operations that the sending object requests and the receiving object implements. Figure 2.10 shows an example of simple messages.

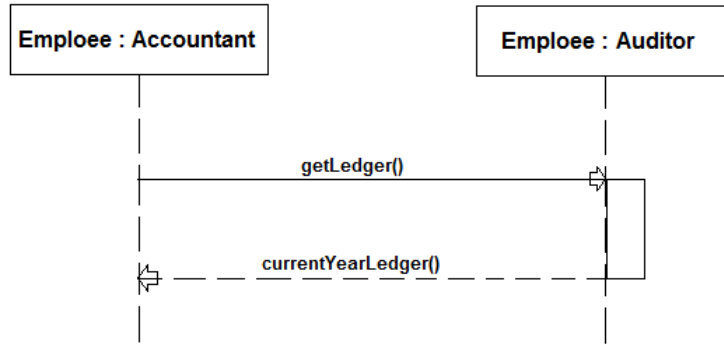


Figure 2.10: Messages between objects

Sequence diagram also allows representing modeling issues like conditions, alternatives, loops, options, etc. For example, to represent a condition that must be met for a certain message to be sent, we can use what is called Guards. To show an example of that, the previous example in Figure 2.10 is edited to include a guarding condition on the sent message.

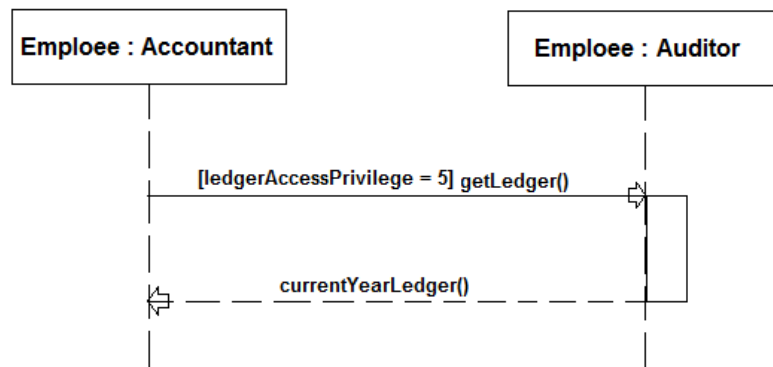


Figure 2.11: Guarded message

Figure 2.12 shows a sequence diagram of Checks handling system.



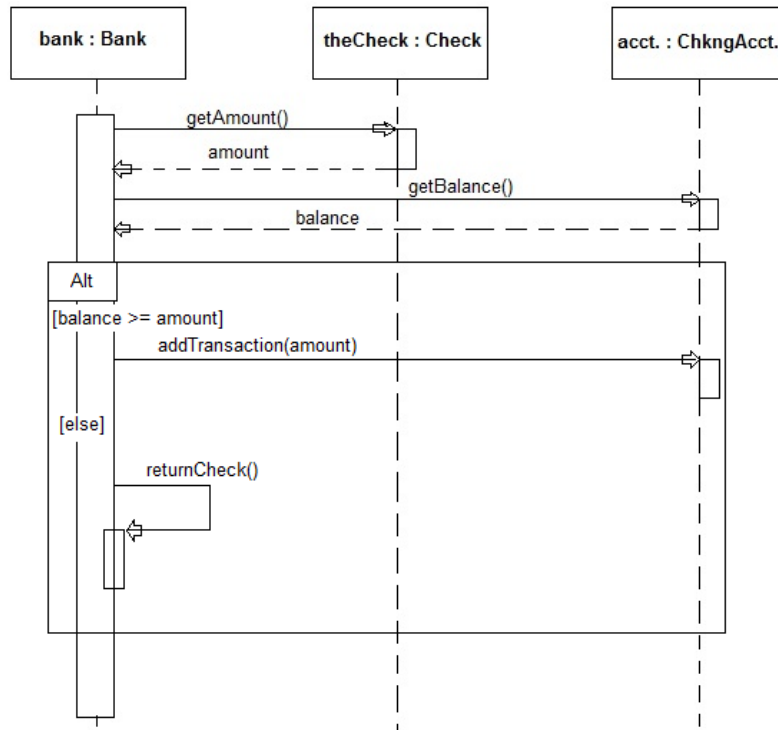


Figure 2.12: Sequence diagram for Check handling system

### 2.1.3 UML use case diagram

The purpose of using use case diagram is to depict the functionality of the modeled system through the use of actors and use cases. It is a type of behavioral diagram that shows the interactions and dependencies between use cases.

The main elements of the use case diagram are:

- 1- Use case: describes a set of actions useful to the actor. Graphically, the use case is depicted as in Figure 2.13.

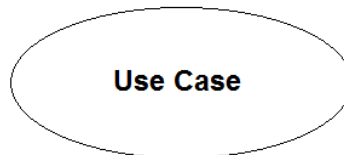


Figure 2.13: Use case icon

- 2- Actor: represents a single person, organization, working system that plays a certain role and interacts with the system. Graphically, the actor is represented as a stick person.

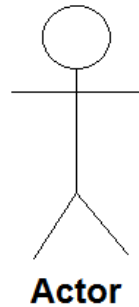


Figure 2.14: Actor icon

- 3- Associations: represent the interactions that happen between the actors and use cases of the system. They are drawn as solid lines connecting the two sides with an optional arrow heads to indicate the direction.

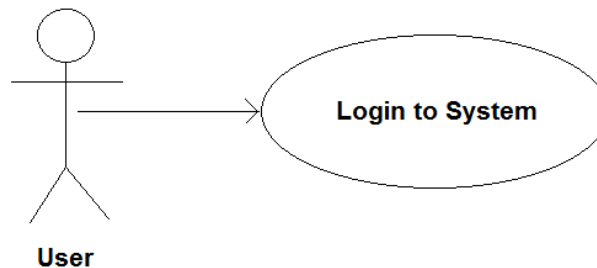


Figure 2.15: Association link between Actor and Use case

- 4- System Boundary: the use of the system boundary shows the scope of the modeled system. It is optional and it is drawn as a rectangle that surrounds the environment.

There are four types of interactions/relationships that connect use cases in the use case diagram. The first one is the Include relationship. It is a relationship that happens between two use cases which indicates that the behavior of the included use case is inserted into including one's behavior. The second type is the Extend relationship. This relationship indicates that the behavior of the extension use case is inserted into the extended one's

behavior. The third and the fourth type of relationship are the Generalization and the Specialization relationships. These relationships are used to represent common behaviors, requirements, constraints, etc. The goal is to have more generalized or specialized use cases. In that sense, behaviors, requirements and constraints can be shifted up or down to the designated use cases. Figure 2.16 depicts an e-commerce system. The figure shows the fundamental elements of use case diagram.

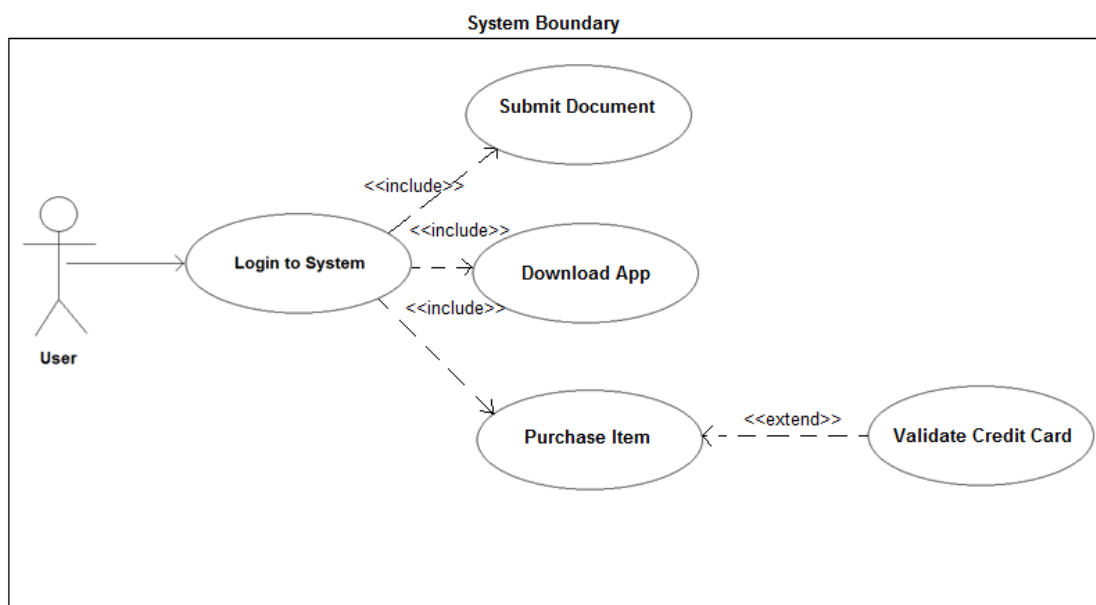


Figure 2.16: Use case diagram for e-commerce website

## 2.2 Meta-models

A meta-model specifies the model for a system [11]. It defines informative statements as a valid model using a modeling language. The meta-model concept is a major issue for software modeling because it is considered the basis for UML definition.

The OMG [11] defined a four-layered UML architecture that consists of different conceptual levels that make up a model: the instances, the model of the system, the

modeling language, and the meta-model of that language. In OMG terminology these layers are called M0, M1, M2, and M3.

### **2.2.1 Layer M0 (user model layer): instances**

The M0 layer consists of the elements that model the actual system. The concepts in this level are instances of concepts in the model layer.

### **2.2.2 Layer M1: The model of the system**

The elements of the M1 layer are instances of the elements in the meta-model layer. The elements in this layer are used to model problems and solutions.

### **2.2.3 Layer M2: The model of the model (the meta-model)**

The elements of layer M2 are the modeling languages. They include concepts from the object-oriented and component-oriented paradigms. The concepts in this layer are instances of meta-meta-model concepts.

### **2.2.4 Layer M3: The model of M2 (the meta-meta-model)**

Finally, layer M3 includes the elements that define the modeling languages.

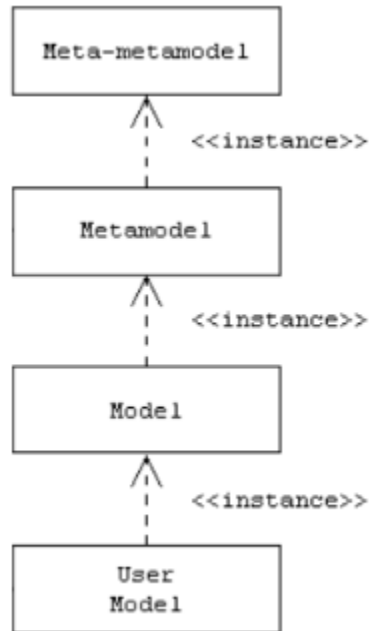


Figure 2.17: Four-layered UML architecture

This research's focal point is the third layer; the meta-model layer. The meta-model elements are the elements that constitute UML. As stated above, this layer includes concepts from the object-oriented and component-oriented paradigms. The "meta" notion is used to indicate a relationship between two sets of concepts; non-meta concepts (the model concepts) and their meta-concepts (the meta-model concepts). The "meta" notion illuminates the role that the model plays. Aspects of this relationship are shown through Abstraction and Manifestation. The Abstraction extracts common features from the non-meta concepts in order to define meta-concepts with such features. On the other hand, Manifestation instantiates meta-concepts to define non-meta concepts with common features. Other aspects of the relationship between non-meta concepts and their meta-concepts include extending (discussed in section 2.3).

Figure 2.18 shows the constituting elements of UML class diagram meta-model [10, 12, 13]. The meta-concepts define the role that the model plays. The meta-concepts are just abstracts from where the non-meta concepts can be driven, particularly from the leaf nodes. The meta-concepts themselves are internally driven from each other. The Model Element is driven from Element and Feature, NameSpace, Generalizable Element, Parameter, Constraint and Relationship are driven from the ModelElement and so on. Another example is the Classifier which classifies three meta-concepts; Class, Data Type and Interfaces. Figure 2.19 shows the original elements of UML sequence diagram meta-model [14, 15] and finally, Figure 2.20 shows the same for UML use case diagram meta-model [16, 17].

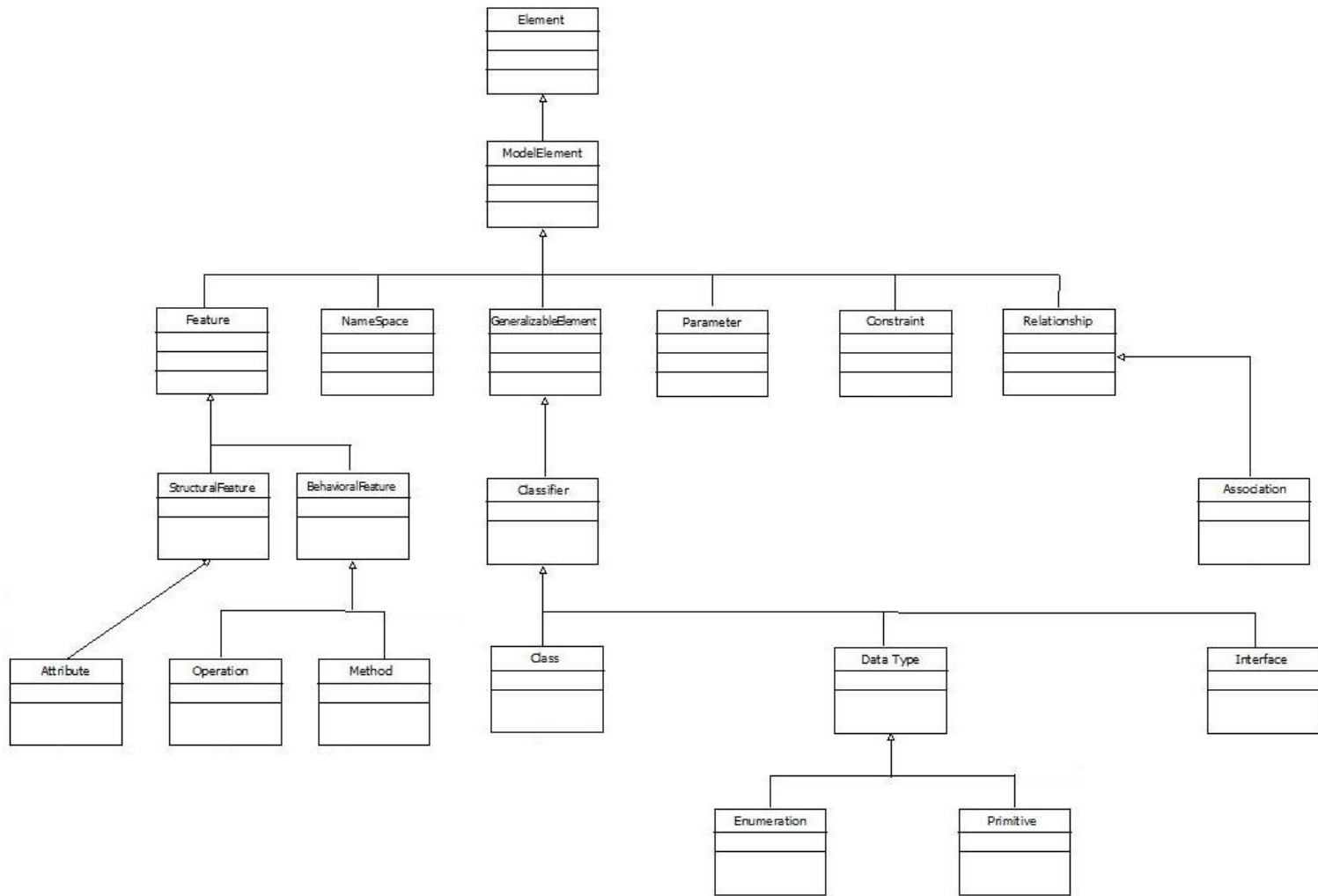


Figure 2.18: Class diagram meta-model

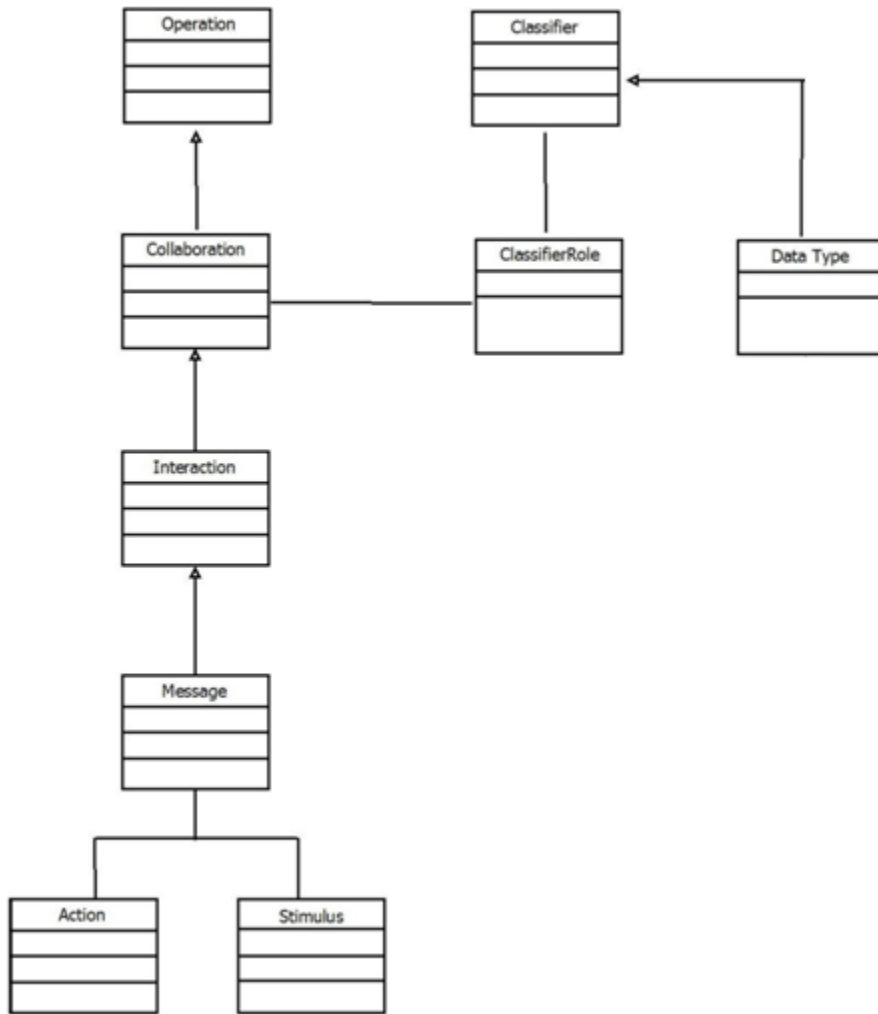


Figure 2.19: Sequence diagram meta-model



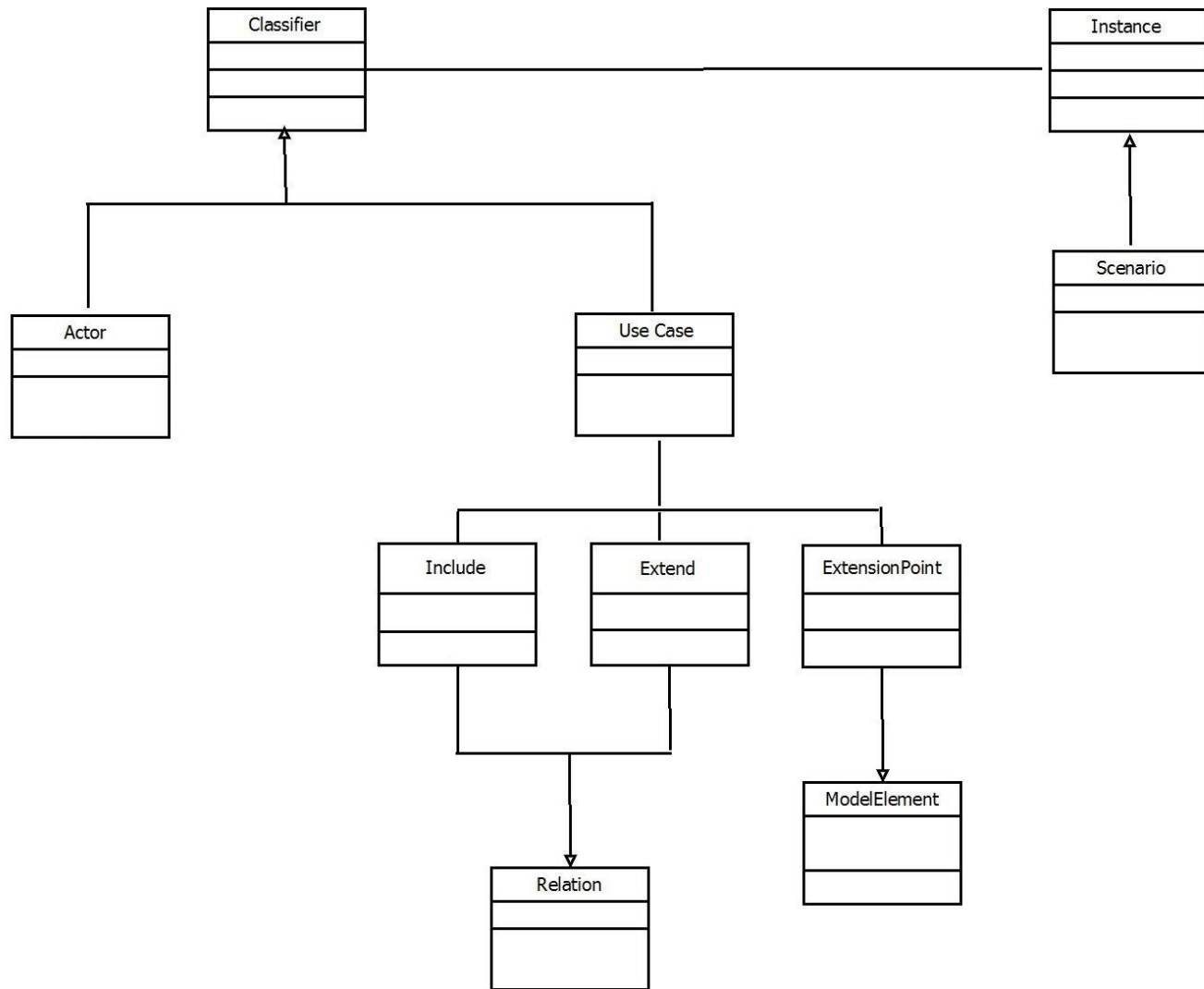


Figure 2.20: Use case diagram meta-model

## **2.3 UML Extension Types**

UML provides notations to satisfy the needs of typical software modeling projects but for certain projects, UML was unable to express certain problem domain needs. UML provides model elements with a particular set of properties. In addition to that, UML provides means to add new properties and modify the existing ones. In that sense, UML can be customized and extended to represent the non-core UML concepts in order to make it suitable to specific problem domains. There are two types of UML extension mechanisms; UML lightweight extension and heavyweight extension. UML lightweight extension defines limited extensions to the meta-model elements. It does not change UML behavior but it can add to or modify UML structure. It mainly provides graphical modifications to UML diagrams. The second type is UML heavyweight extension and it involves editing the meta-model through the reuse technique of UML package. It can customize UML behavior and operations but its development is difficult and costly [18].

### **2.3.1 UML lightweight extension**

UML profile mechanism customizes the MOF's (Meta Object Facility) meta-models by introducing a new terminology and specializing the semantics of UML. UML profile extends UML by three main constructs; stereotypes, tag definitions and constraints.

Stereotypes introduce domain specific terminology into the modeling language. They extend the meta-classes and can be applied only to instances of the extended meta-classes. The way to represent a stereotype is by placing the name of the stereotype above the name

of UML element and it needs to be between <<>> sign. Figure 2.21 illustrates a modeling stereotype.

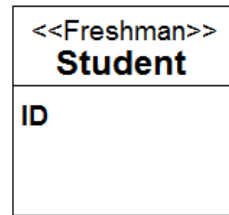


Figure 2.21: Stereotype

Tag definitions are considered properties of stereotypes, they introduce additional attributes, and they also specify values, called tagged values. Graphically, they are shown as a tag-value pair where the tag represents the newly defined property and the value represents the assigned value to that property. As stated above, tagged values can set properties for stereotypes. Figure 2.22 shows a tagged value placed in the topmost compartment of the class. It indicates that the admission year of the student is a tag with a certain value.

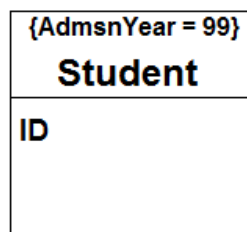


Figure 2.22: Tagged value

Finally, the modeling constraints can be written and specified by the OCL (Object Constraint Language) [19]. OCL constraints represent rules and conditions that must be held and fulfilled by the modeling elements.

The following example is merely mentioned to give more explanation on UML profiles. This example of extending UML lightly is done by Cabot et al. [20]. The goal is to create a UML profile to represent GUI components. The proposed GUI contains Forms (which can also be dialog boxes) and Buttons. In general, there are two constraints; the first one is that the Form can invoke a dialog box and the second constraint is that the Form, as well as the dialog box, can contain Buttons. The GUI profile is depicted in Figure 2.23.

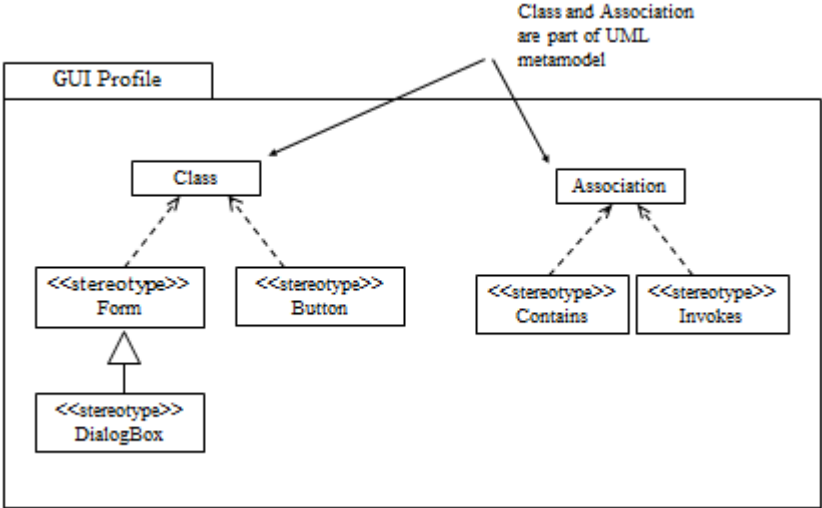


Figure 2.23: GUI Profile proposed by Cabot et al. [20]

To put this profile into action, Cabot et al. [20] made the following instance diagram.

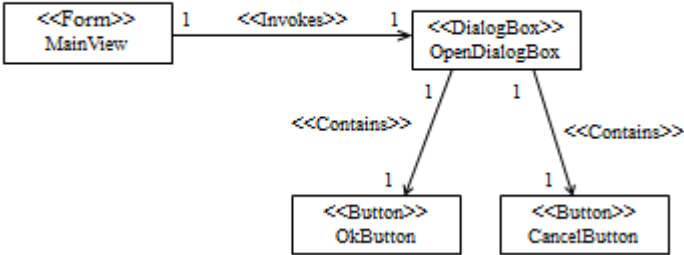


Figure 2.24: Example of using GUI profile proposed by Cabot et al. [20]

### 2.3.2 UML heavyweight extension

In comparison with the lightweight extension, UML heavyweight extension is much harder [18]. It changes UML meta-model level by adding new modeling elements or modifying the existing ones. What differentiates UML heavyweight extension from the lightweight extension is the ability to change the behavior of UML and the advantage of having more features from UML such as redefine, subset, or derivation of meta-types properties.

The way UML heavyweight extension works is by package re-using techniques such as merge and import. The procedure of extending UML heavily starts first with selecting UML modeling elements that need to be extended, these elements will be taken by importing the Kernel package, and then these elements will be merged with the other elements coming from the newly introduced package. Figure 2.25 shows the work of Przybylek [21] in extending UML heavily. Figure 2.25 also depicts the introduced package, called AoUML, in which Przybylek [21] reused elements from UML infrastructure and superstructure specifications by importing the Kernel package.

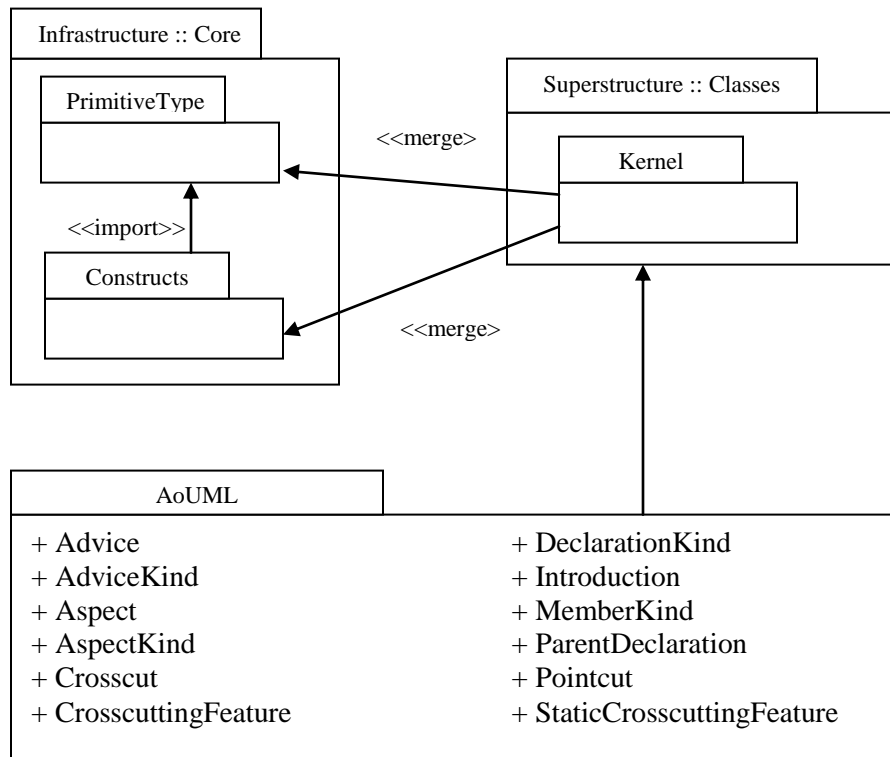


Figure 2.25: Dependencies between packages as presented by Przybylek [21]

UML heavyweight extension mechanism requires a combination of notations. First, a UML diagram, mostly UML class diagram, to show the existing constructs and the way they are built. The second notation is OCL constraints. The last one is natural language to describe the semantics of the newly introduced meta-classes.

Finally, this mechanism is extensible and scalable, but lacks package re-use modularity and its development is relatively costly. In addition to that, it is difficult to develop and maintain.

# CHAPTER 3

## 3. Literature Review

This chapter surveys the literature on the extensions of class, sequence and use case diagrams. Those three diagrams are considered the most famous representatives for three distinctive views of the modeled system. The class diagram depicts the system's structure, the sequence diagram represents the interactions between the system's objects and the use case diagram describes the provided functionality of the system. Another reason why this survey considers those diagrams only is because that most of the extensions done in the literature are applied to these three diagrams and since this research's goal is to integrate extensions from the literature, class, sequence and use case diagrams had to be picked out from the entire set of UML diagrams.

The methodology of the review is as follows; categorizing the papers into four categories; class, sequence, use case and other diagrams, and then categorizing the papers in each diagram's category into three types of UML extensions; lightweight (graphical, meta-model) and heavyweight.

## 3.1 Class diagram

### 3.1.1 UML class diagram lightweight extension (graphical)

Fontoura et al. [22] proposed a new profile called UML-F which describes how to represent framework variation points in UML diagrams to describe the structure and behavior of these variation points. Fontoura et al. [22] modeled the variation points using tagged values of Boolean type. UML diagrams are extended by the following tags; {variable} to represent variable methods and {extensible} to represent extensible classes. Also the tags {static} and {dynamic} are used to classify method and classes according to their runtime requirements. The {incomplete} tag is used to identify extensible interfaces. The tag {appl-class} place holds classes that are defined as part of the instantiated applications. {for all new methods} is used to describe the behavior of methods. In other words, {for all new methods} indicates that the OCL constraint applies to the added methods during instantiation. Finally, the {optional} tag indicates that certain interaction patterns (actions) are not mandatory. Table 3.1 summarizes the introduced tags.

Table 3.1: Summary of the new elements and their meanings proposed by Fontoura et al.[22]

Name of extension	Type of extension	Applies to notational element of UML	Description
{appl-class}	Boolean Tag	Class	“Classes that exist only in framework instances, New application classes may be defined during the framework instantiation.”
{variable}	Boolean Tag	Method	“The method that is implemented during the framework instantiation.”
{extensible}	Boolean Tag	Class	“The class interface depends on



			the framework instantiation: new methods may be defined to extend the class functionality. “
{static}	Boolean Tag	Extensible Interface, Variable Method, and Extensible Class	“The variation point does not require runtime instantiation. The missing information must be provided at compile time.”
{dynamic}	Boolean Tag	Extensible Interface, Variable Method, and Extensible Class	“The variation point requires runtime instantiation. The missing information may be provided only during runtime.”
{incomplete}	Boolean Tag	Generalization and Realization	“New subclasses may be added in this generalization or realization relationship.”
{for all new methods}	Boolean Tag	OCL Constraint	“Used to indicate that the OCL constraint must be met by the introduced methods.”
{optional}	Boolean Tag	Events	“Used to indicate optional event.”

In Figure 3.1, Fontoura et al. [22] used a couple of the proposed Boolean tags to indicate certain issues. For example, they applied the tag {appl-class} to the class Librarian to show that this class exists only in framework instances.

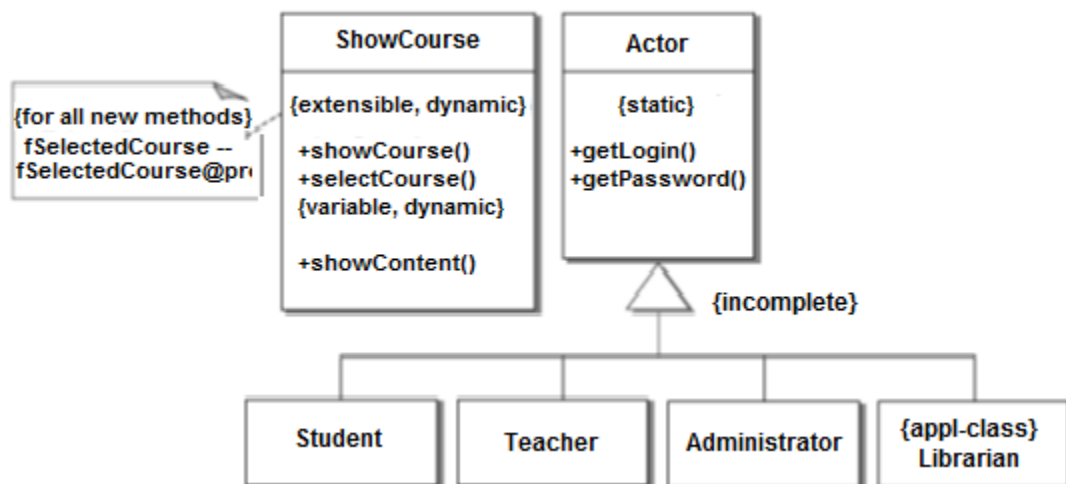


Figure 3.1: UML-F extended class diagram proposed by Fontoura et al. [22]

Byeon et al. [23] extended UML to model GNSS (Global Navigation Satellite System). GNSS is an environment that requires accurate measurements and calculation of real-world geographical entities with the aid of GPS (Global Position System) in two specific areas; temporal and spatial.

Byeon et al. [23] used a diagrammatic tool called "Stereotype Creator" to create iconic stereotypes to model GNSS application. The main elements of geo-referenced classes are: a graphical representation with a symbolistic icon, an iconic notation to indicate the geographic type, class name, attributes and operations.

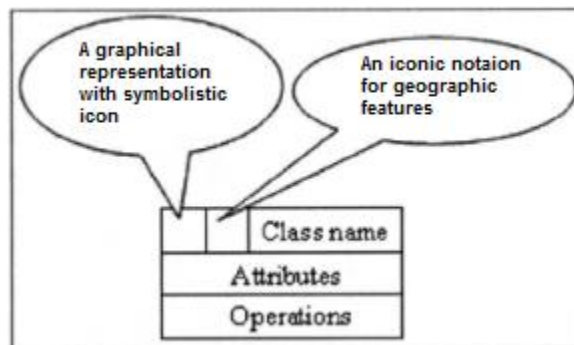


Figure 3.2: Graphical representation of class meta-model element proposed by Byeon et al.[23]

The authors have put the following example;

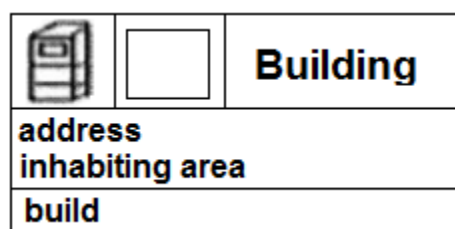


Figure 3.3: Example of geo-referenced class presented by Byeon et al.[23]

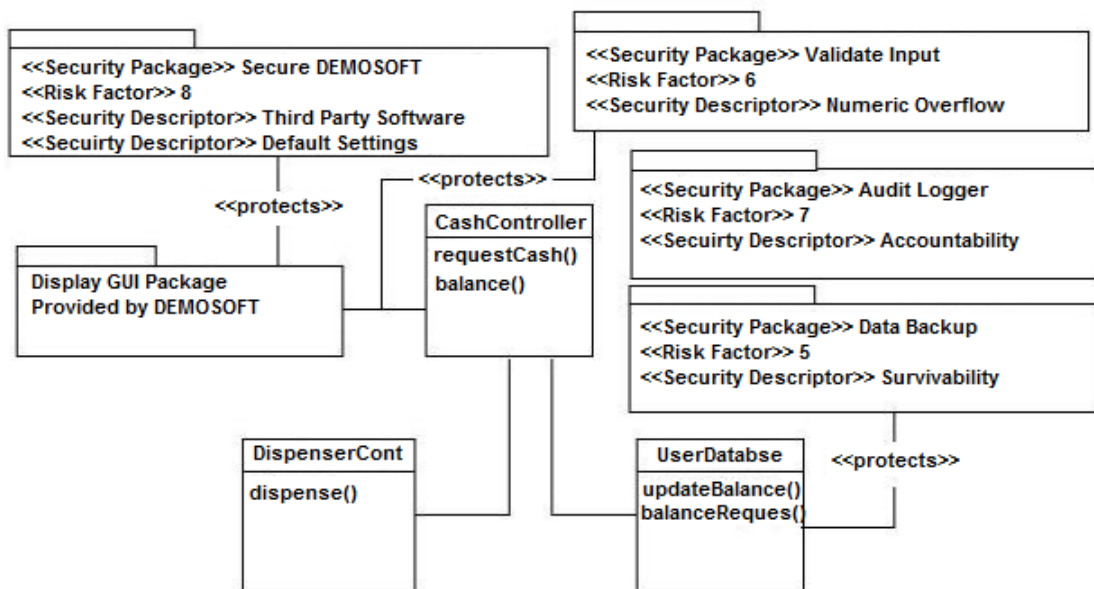


Figure 3.4: Class diagram integrated with UMLpac for security features proposed by Peterson et al. [24]

Dong [25] presented notations to represent individual and composed design patterns. The author believes that identifying design patterns is extremely difficult, especially when they are composed, because some pattern-related information may get truncated or even lost when using the traditional UML diagrams.

Dong [25] showed a number of annotations for design patterns. To name a few; Venn Diagram-Style Pattern Annotation, Dotted Bounding Pattern Annotation, UML Collaboration Notation, Pattern: Role Annotations, Stereotype Annotations, Tagged Pattern Annotation.

Dong [25] noticed that the Venn Diagram-Style Pattern Annotation and UML Collaboration cause confusion when the class participates in a huge number of patterns. As for the Dotted Bounding Pattern Annotation and Pattern: Role Annotations, the author found them difficult to identify precisely the roles of modeling elements and as for the Stereotype Annotations, the author found the notations expensive to design , use and maintain, plus they are not scalable.

The Tagged Pattern Annotation is the notation that the author suggested. Its core idea is that, for each class, new tagged values are created to hold pattern and participant names associated with the class and the same goes for the class's operations and attributes. If the tagged values cause any confusion, the participants' names will only be shown.

Sanada and Adams [26] defined a new UML profile to model design patterns and frameworks in design class diagrams. This work distinguishes between design class diagrams, detailed design class diagrams and design pattern class diagrams.

The authors provided new stereotypes for design patterns: <<InstanceClass>>, <<forAllNewMethods>>, <<Template>> and <<Hook>> as shown in Table 3.2.

Table 3.2: Stereotypes for Design Patterns proposed by Sanada and Adams [26]

Stereotype	Base Class	Parent	Tags	Constraints
InstanceClass <<InstanceClass>>	Class	N/A	Extensible, instantiation, final	None
ForAllNewMethods <<ForAllNewMethods>>	Constraint	N/A	None	None
Hook <<Hook>>	Method	N/A	None	None
Template <<Template>>	Method	N/A	None	None

<<InstanceClass>> is used to model the varying concept encapsulated by the pattern. <<ForAllNewMethods>> is used to indicate that the constraint will be held for all the new methods, and as for <<Template>> and <<Hook>>, they are used to indicate the roles of methods in the pattern.

Sanada and Adams also provided new tags: extensible, instantiation and final as shown in Table 3.3.

Table 3.3: Tags in UML Profile for Design Patterns proposed by Sanada and Adams [26]

Tag	Stereotype	Type	Multiplicity
Extensible	N/A	UML::Datatypes::Boolean	1
Instantiation	InstanceClass	UML::Enumeration:{replace, extend}	1
Final	N/A	UML::Datatypes::Boolean	1

The tag (extensible) is used to add new attributes and methods for the new class. The (instantiation) tag is used to indicate the instantiation of classes and the tag (final) are used to indicate that the final class has no decedent classes (leaf).

On the other hand, Sanada and Adams [26] have also added stereotypes and tags to model frameworks as shown in Table 3.4.

Table 3.4: Stereotypes for Frameworks proposed by Sanada and Adams [26]

Stereotype	Base Class	Parent	Tags	Constraints
InstanceClass <<InstanceClass>>	Class	N/A	Extensible, instantiation, final	None
ForAllNewMethods <<ForAllNewMethods>>	Constraint	N/A	None	None

Hook <<Hook>>	Method	N/A	None	None
Template <<Template>>	Method	N/A	None	None

The stereotypes in Table 3.5 are the same ones for Design Patterns except for <<ApplicationClass>> which indicates classes that exist only in the framework instance. As for the tags, three of them are especially made for frameworks.

Table 3.5: Tags in UML Profile for Frameworks proposed by Sanada and Adams [26]

Tag	Stereotype	Type	Multiplicity
Variation	N/A	UML::Datatypes::Boolean	1
Extensible	N/A	UML::Datatypes::Boolean	1
Binding	N/A	UML::Enumeration:{static, dynamic}	1
Instantiation	ApplicationClass	UML::Enumeration:{replace, extend}	1
Final	N/A	UML::Datatypes::Boolean	1
PatternName-Role	N/A	UML::Datatypes::String	1

The tag (variation) means that the method's implementation is the same as the varying concept that the pattern encapsulates. The tag (building) indicates whether the variation points require runtime instantiation. Finally, the tag (PatternName-Role) is used to specify the participants' roles in the patterns.

### 3.1.2 UML class diagram lightweight extension (meta-model)




Jantan et al. [27] proposed a hypermedia design method called ComHDM which is a UML profile. The authors proposed modeling elements to model the conceptual, navigational and user interface artifacts of web hypermedia applications.




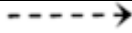

Jantan et al. [27] claimed that the effort of developing web applications has risen a number of design issues, such as; modeling complex business processes, navigation access structures, activities and transactional workflows, user dependent processes, and so on.

The proposed method separates design stages: conceptual, navigational and user interface. It also uses UML stereotypes to model the application domain. The navigational stereotypes define navigation classes and their associated access mechanisms. Finally, the method defines user interface modeling elements to provide interaction mechanisms between the users and the application.

Jantan et al. [27] proposed the stereotypes shown in Table 3.6 to model complex processes in web applications.

Table 3.6: Proposed Stereotypes in Conceptual Process Design by Jantan et al. [27]

Stereotype/ Graphical Notation	Descriptions
 <<process class>> Process_Class	<ul style="list-style-type: none"> <li>• “Defined as the similar way as action taken by user to perform an activity. This can be done easily by referring to the use case definition in functional requirement analysis.”</li> <li>• “Instance or object would be used by users during the execution of a sequence of pre-defined processes.”</li> </ul>
 <<atomic class>> Atomic_Class	<ul style="list-style-type: none"> <li>• “Inherits the definition of process class. Determined by the action taken in use case definition.”</li> <li>• “Can be performed in sequential order (they might have dependencies from each other).”</li> </ul>
 <<non-atomic class>> NonAtomic_Class	<ul style="list-style-type: none"> <li>• “Inherits the definition of process class. Determined by the action taken in use case definition.”</li> <li>• “The execution of non-atomic or pre-defined processes must be performed in sequential order (they might have</li> </ul>










	dependencies from each other).”
 <<database class>> Database_Class	<ul style="list-style-type: none"> <li>• “Models the experience of database in design (to provide a logical view of database operations between process class and database class).”</li> <li>• “Database class must owned by at least one process class.”</li> </ul>
 <<process container>> Process_Container	<ul style="list-style-type: none"> <li>• “Group and partition process class and all of its objects/ instances in order to indicate their relationships or dependencies. “</li> <li>• “Determine which partition an instance of processes belongs to.”</li> </ul>
 <<Process_Link>> (Stealth Arrow)	<ul style="list-style-type: none"> <li>• “Association between two separated classes; conceptual class to process class and vice versa.”</li> <li>• “Also known as external link.”</li> </ul>
 <<Action_Link>> (Dashed-Stealth Arrow)	<ul style="list-style-type: none"> <li>• “Association between operations taken by users in the same process class (process class to process class).”</li> <li>• “To force dependencies of processes and information flow in particular process class.”</li> </ul>
 <<Database_Link>> (Bold Arrow)	<ul style="list-style-type: none"> <li>• “Association between conceptual class or process class to database class.”</li> <li>• “Represent the information and data operations such as query, lookup, entry, etc., that involved with database.”</li> </ul>




After defining the conceptual process design stereotypes, Jantan et al. [27] also defined a set of navigation classes connected through hyperlinks. The purpose of doing that is to present navigation classes, interaction classes and hyperlinks.

The following UML stereotypes in Table 3.7 are proposed to model the navigational access:



Table 3.7: Summary of Navigational Access Stereotypes in ComHDM proposed by by Jantan et al. [27]

Stereotype/ Graphical Notation	Descriptions
 <<navigational class>> Navigational Class	<ul style="list-style-type: none"> <li>• “Derived from the Conceptual Class Model (CCM) – Has similar name as Conceptual Class name. “</li> <li>• “Instance or object would be used by users during the navigation access. “</li> </ul>
 <<interaction class>> Interaction Class	<ul style="list-style-type: none"> <li>• “Derived from the Conceptual Process Model (CPM) “</li> <li>• “Presents the existence of Used to represent complex interaction between users and web application. “</li> </ul>
 <<navigation link>> Hyperlink	<ul style="list-style-type: none"> <li>• “Presents the association / hyperlinks between navigation classes (from source code to the target code). “</li> <li>• “Equipped by "role name" and "multiplicity".“</li> </ul>
 <<index>> Index	<ul style="list-style-type: none"> <li>• “An access element that contains a number of listed items/ target name with a link to the target navigation class.”</li> </ul>
 <<guided tours>> Guided Tours	<ul style="list-style-type: none"> <li>• “To provide an ordered sequential access to instances/ objects of a navigation class. It can be controlled by either web users (interactive) or system (temporal/ time-based). “</li> </ul>
 <<textQuery>> Text Query	<ul style="list-style-type: none"> <li>• “An interactive access element that provide an input field (string or character) mainly for search mechanism. “</li> </ul>
 <<selectableList>> Selectable List	<ul style="list-style-type: none"> <li>• “An interactive access element that provide frozen listed items (selectable items). “</li> <li>• “An alternative access element for non-text (input) query. “</li> </ul>
 <<tree>> Tree	<ul style="list-style-type: none"> <li>• “Uses for classifying instances- for orientation purpose, it helps users for browsing a kind of hierarchical structure of information (listed items can be expanded or collapsed). “</li> <li>• “An alternative access element for nested index. “</li> </ul>
 <<page>> Page	<ul style="list-style-type: none"> <li>• “Provides direct access to group of instances in a navigation class. Each</li> </ul>

	<p>page is numbered or named and has its own link to target instance location. “</p> <ul style="list-style-type: none"> <li>• “An alternative access element for guided tours. “</li> </ul>
 <<menu>> Menu	<ul style="list-style-type: none"> <li>• “A group of homogenous items that provide access links to target navigation classes or access structure elements. “</li> <li>• “Each item has its own link to a target location and they are all frozen items. “</li> </ul>
 <<trail menu>> Trail Menu	<ul style="list-style-type: none"> <li>• “Inherits the definition of menu. An alternative access element of menu if they consist of menu sub-items. “</li> <li>• “Sub-items can be expanded or collapsed. “</li> </ul>
 <<tab menu>> Tab Menu	<ul style="list-style-type: none"> <li>• “Provides variety options of views in menu. The menu items are partitioned (separated) into different number of tabs (normally in horizontal view). “</li> </ul>

Finally, the Jantan et al. [27] provided user interface elements for every single web page. They presented user interface mapping rules to ensure correct mapping between navigation stereotypes and user interface stereotypes as shown in Table 3.8.

Table 3.8: Mapping Rules between Navigation Design and User Interface Design in ComHDM proposed by Jantan et al. [27]

Navigation Stereotype	Map to – User Interface Stereotypes
<<navigation class>>	“<<UIPage>>→<<framePage>>→<<UIElement>>“
<<interaction class>>	“<<UIInteraction>>→<<framePage>>    <<UILogin>>→<<framePage>>    <<UISession>>    <<UIElement>>“
<<navigation link>>	“<<accessElement>>    <<hyperlink>>    <<formElement>>“
<<access structure>>	“<<UIElement>>→<<accessElement>>    <<standardElement>>    <<formElement>>“

Fernandez-medina et al. [12] addressed the confidentiality problems for Data Warehouses by specifying security constraints in the conceptual Multidimensional Database modeling

to design secured Data Warehouses. The reason why the authors emphasized Data Warehouses' security is because Data Warehouses and other applications like Multidimensional Databases and On-Line Analytical Processing applications are considered very powerful mechanisms for discovering important business information; hence, security for such applications is considered a major issue.

The proposed UML extension reused a number of previously defined stereotypes and defined new ones of their own. Fernandez-medina et al. [12] have also added a number of tagged values and constraints to model the Multidimensional Databases properly. The new elements helped in specifying security measures, such as; security levels and user roles on the main elements like; facts, dimensions and classification hierarchies. In addition to that, Fernandez-medina et al. [12] used OCL constraints on the new defined elements in order to avoid misuse.

Simons and Wirtz [13] presented Context Modeling Profile (CMP), a UML profile for modeling mobile distributed systems. They defined stereotypes and well-formedness rules.

Mahmood and Lai [28] presented an extension to UML called RE-UML, to support the phases of Requirements Analysis and Assessment Process (RAAP). RE-UML extends UML class diagram with two specialized classes, Rclass to specify stakeholder requirements and Cclass to specify component features.

RClass, shown in Figure 3.5, is a special class divided into four sections:

- 1<sup>st</sup> section: stereotyped requirement text + name of the class + abstraction level to differentiate the requirement level.
- 2<sup>nd</sup> section: the objective of the RClass.
- 3<sup>rd</sup> section: scenario which is the set of interactions necessary to achieve the objective.
- 4<sup>th</sup> section: rank of the RClass.

<<requirements >> - Abstraction Level
Goal
Scenario
Rank

Figure 3.5: RClass proposed by Mahmood and Lai [28]

CClass is another special class divided into three sections:

- 1<sup>st</sup> section: stereotyped component text+ name of the class.
- 2<sup>nd</sup> section: the functionality provided by the component.
- 3<sup>rd</sup> section: the dependency on elements and their relationships.

<<component >> Name
Features
Context Dependency

Figure 3.6: CClass proposed by Mahmood and Lai [28]

As for Associations, there are two types of associations were introduced:

- 1- Interaction relationship (association): between two RClasses.
- 2- Mapping relationship (association): between RClass and CClass (RSatisfy).

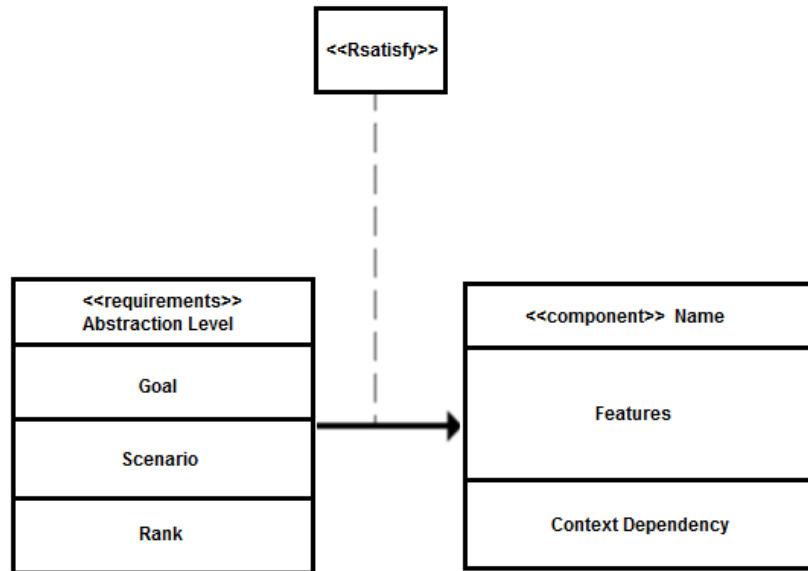


Figure 3.7: Satisfy mapping relationship proposed by Mahmood and Lai [28]

Similarly, UML sequence diagram is extended with the frame <<Rsatisfaction>> to model the satisfaction process that happens between stakeholder requirements and component features.

Sharafi et al. [29] presented an UML extension to capture crosscutting concerns in aspect-oriented modeling. The novelty of their work is in their model, which was created to be language-independent, plus, it was abstracted away from any platform specific details. The reason why the authors have done that is because they wanted their model to make the transformations that happen during maintenance.

The defined model included the following elements:

- A set of core concepts.

- A set of sound relationships between the core concepts.
- A set of constraints.
- A concrete syntax or graphical representation of the domain model.
- Semantics of the domain model.

The next step in their work was a mapping process. Sharafi et al. [29] mapped the domain model to UML meta-model. For example, they mapped the Aspect to UML meta-model. The following step was providing a graphical representation for modeling crosscutting concerns using UML tools. Finally, the authors claimed that to be able to deploy the defined profile in CASE tools, it is necessary to provide a robust interchange format. The authors selected XMI [30] (XML Meta-data Interchange) for three reasons, first; it has a wide market and tool support and secondly, it is compatible with UML and finally it uses XML syntax.

### **3.1.3 UML class diagram heavyweight extension**

Przybylek [21] extended UML meta-model to support aspect-oriented modeling. Przybylek's work [18] is an integration of previous works, existing AO extensions. It also defines a MOF meta-model based on UML but with means to model AOM. The specification of this extension uses a combination of notations; UML class diagram, OCL constraints and natural language.

El-Kady et al. [18] developed a MAS-UML (Multi-Agent System UML) by extending UML meta-model heavily. The goal of their work was to represent the MAS conceptual model.

The added meta-classes have the following relationships:

- 1- “AgentType represents the meta-class for the agent instances that have the same features specification. The agentType internal structure contains beliefs, goals and agentStates features.”
- 2- “Belief meta-class represents the belief component as part of an agent.”
- 3- “Goal represents the goal that should be achieved by the owner.”
- 4- “AgentTypePermission meta-class represents the permission that an agentType instances can achieve for a specific resource.”
- 5- “Environment meta-class represents the environment where agents and resources can exist.”
- 6- “Behavior meta-class is an abstract meta-class representing the root of the MAS actions pattern.”

## **3.2 Sequence diagram**

### **3.2.1 UML sequence diagram lightweight extension (graphical)**

Zhou et al. [14] presented three things; first they proposed UML extension profile for aspect-oriented modeling. Secondly, they built a framework with UML and finally, they presented a way to model the dynamic behaviors that happen in aspect-oriented software. Their main objective was to propose architecture for aspect-oriented modeling, and address the separation of concerns properly.

Zhou et al. [14] extended UML sequence diagram from two angles: the first one is by presenting joint points in sequence diagram and the other is by adding new crosscutting bar that is used to send crosscutting message. Figure 3.8 shows the addition of crosscutting bar to UML sequence diagram.

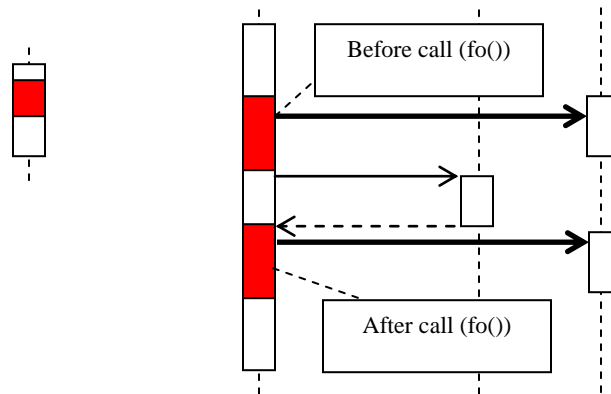


Figure 3.8: Crosscutting Bar and Invocation with Crosscutting Bar proposed by Zhou et al. [14]

Hausmann et al. [15] specified the operational semantics of UML behavioral diagrams. Since Fontoura et al. [21] believe that UML has no agreed specification of its semantics, Hausmann et al. [15] presented an approach that specifies the semantics of modeling languages.

OMMMA-L (Object-oriented Modeling of Multi-Media Applications - Language) [31] has been proposed as an extension of UML to specify interactive multimedia presentations. OMMMA model basically consisted of:

- 1- “A class diagram that forms the application aspect. It contains application classes related to media classes. (Application)“



2- “A state chart diagrams that represent state machines to specify the media aspects.  
(Dynamic and event-driven system behavior)“

“An (Extended) sequence diagrams that model sequences of presentation behavior.“

### **3.2.2 UML sequence diagram lightweight extension (meta-model)**

Cortellessa and Pompei [32] focused their work on integrating UML with non-functional attributes (aspects). Their goal was representing issues related to the reliability modeling of component-based systems. Issues like Quality of Service and Fault Tolerance.

Cortellessa and Pompei [32] defined a domain model, and then mapped its concepts to UML viewpoint. The elements of the defined model are; REservice, which is a set of actions and interactions that happen among a set of REcomponents that interact through REconnectors. The goal of REservice is to serve REuser that requests the service and finally, a REhost that performs the hosting of a set of components.

After defining the core concepts, a set of relations were defined.

- 1- “One REuser requires many REservices and one REservice can be required from many REusers.”
- 2- “A REservice can be triggered either by a REuser or a REcomponent.”
- 3- “Each REcomponent can have a hierarchical structure.”
- 4- “A set of REcomponents is hosted by a REhost.”

- 5- “Each REconnector can be a logical link between two REcomponents. It also can be a physical link between two REhosts.”

The next step was a mapping step. The newly defined elements were mapped to UML viewpoint as follows:

Table 3.9: <<REcomponent>> as defined by Cortellessa and Pompei [32]

Stereotype	Base Class	Tags
<<REcomponent>>	Classifier ClassifierRole Component Instance	REcompfailprob REbp

Table 3.10: <<REconnector>>as defined by Cortellessa and Pompei [32]

Stereotype	Base Class	Tags
<<REconnector>>	Message Stimulus AssociationRole	REconnfailprob REnummsg

Table 3.11: <<REuser>>as defined by Cortellessa and Pompei [32]

Stereotype	Base Class	Tags
<<REuser>>	Classifier ClassifierRole Interactor Instance	REaccessprob REserviceprob

Table 3.12: <<REservice>>as defined by Cortellessa and Pompei [32]

Stereotype	Base Class	Tags
<<REservice>>	Classifier	REprob

Table 3.13: <<REhost>>as defined by Cortellessa and Pompei [32]

Stereotype	Base Class	Tags
<<REhost>>	Node Classifier ClassifierRole	REindexHost

## **3.3 Use case diagram**

### **3.3.1 UML use case diagram lightweight extension (graphical)**

Dong et al. [33] believe that UML lacks support for the distributed system. So they proposed an extension to UML to address this problem. Their UML extension [33] changes the use case diagram to be active and multi-level for requirements engineering of distributed system.

The proposed changes to the use case diagram were the following:

- 1- “Change Use Case Diagram to be multi-level: It divides the use case diagram into three levels; user-system level (Level 1), sub-network and sub-network level (level 2) and node and sub-network level (level 3).“
- 2- “Introduce the concept of Abstract Actor: The goal is to specify the actors who have uncertain types but their roles are the same. “
- 3- “Introduce the concept of Abstract Connection: The goal is to specify the relationship between Abstract Actors and Use Cases.“

### **3.3.2 UML use case diagram lightweight extension (meta-model)**

Fei and Yan [16] analyzed a real application called SPAERIS using an UML extension called Agent UML. SPAERIS (Shipping pollution accident emergence reflecting information system) is an application used to monitor and control the ships' security. Fei and Yan [16] used Agent UML to design a distributed management information system.

In the analysis stage, they used symbols like  $\langle \text{agent} \rangle$  as an extension to UML made by the Agent UML to express that the entity is seen as an agent instead of a class.

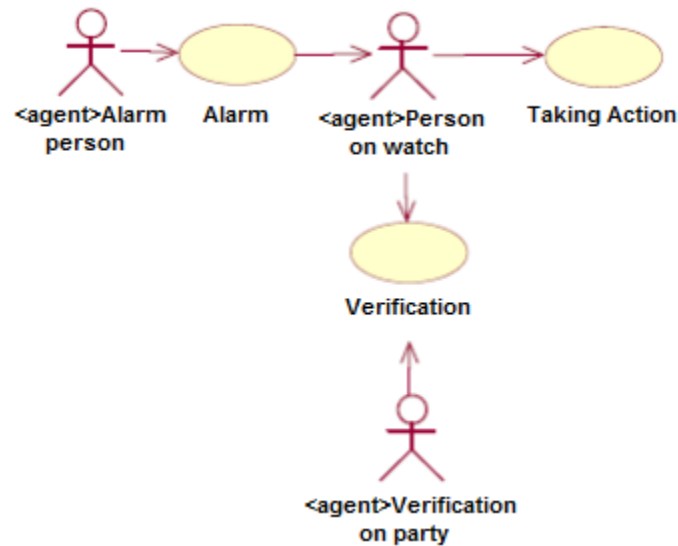


Figure 3.9: Alarm use case proposed by Fei and Yan [16]

Agents are specified by three classifiers; agent classifier, agent physical classifier and agent role classifier. Agent classifiers are used to classify agents. An agent role classifier is an agent classifier that is used to classify agents according to their given roles. Finally, agent physical classifier is used to define common features that exist in all agents.

Djemaa et al. [17] presented WA-UML (Web Adaptive - UML) which is a UML profile to model adaptive web applications. This profile added labels and notations to UML diagrams in order to express UML more effectively.

Djemaa et al. [17] chose Use Case diagram to express the added labels and notations. In terms of actors, three categories of actors were proposed. These actors are classified as follows:

- 1- “Physical actor: represents the human user who visits the Web application.”
- 2- “Logical actor: represents the role played by a human user (physical actor) to maintain the Web application.”
- 3- “System Actor: represents the hardware aspect of the system, whether it is a computer system, device hardware or web service. “

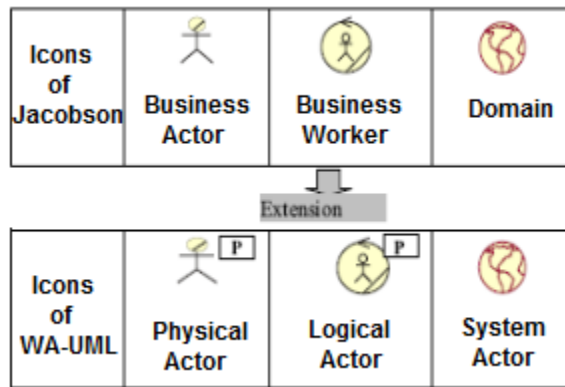


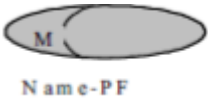


Figure 3.10: Actors of WA-UML proposed by Djemaa et al. [17]

And in terms of functionalities, Three types of functionalities were pointed at; Static Informational Functionality (SIF), Dynamic Informational Functionality (DIF) and Professional Functionality (PF).

Table 3.14: WA-UML notations for use cases proposed by Djemaa et al. [17]

Notation	Description
 Name-SIF	“SIF: Static Informational Functionality used to represent a static Web page.”
 Name Name-DIF	“DIF: Dynamic Informational Functionality used to represent a dynamic Web page. “

	<p>“PF: Profession Functionality used to represent a dynamic Web page using update request.”</p>
---	--

### 3.4 Other diagrams

#### 3.4.1 Class, component, activity, state chart and interaction diagram

Romero et al. [34] focused their work on open distributed processing (ODP) computational viewpoint which describes the functionality and the environment of a system. UML Profile for the ODP computational viewpoints consists of three parts. First, it defines the ODP computational viewpoint meta-model. Second, it maps ODP concepts to UML elements. Finally, it defines a set of OCL constraints.

#### 3.4.2 State chart diagram

Andre et al. [35] used SysML (System Modeling Language) which extends UML to model real-time systems. SysML is a modeling language for systems engineering applications. Their time model, which adds meta-classes to represent time and duration, uses: value property and constraint block. The value property specifies values and the constraint block embeds equations to define the value constraints.

### 3.4.3 Class and activity diagram

Majzik et al. [36] introduced a UML extension to integrate platform-specific development environment of time-triggered systems and a visual design tool based on UML. In their UML extension, Majzik et al. [36] extended two main elements from UML meta-model; classes and association classes. Classes were used to model concepts within the system while association classes were used when associations have class properties. The authors used a number of stereotypes to define the new modeling elements, a number of tagged values to attach properties to the elements and a set of constraints to specify conditions held onto the elements. Using this profile, designers are able to create time-triggered architecture cluster in the form of class diagrams and specify task behavior using activity diagram.

### 3.4.4 Activity diagram

Pllana and Fahringer [37] claimed that the semantics of specific diagrams are not always clear in order to decide how to model specific aspects of parallel applications. The presented UML extension solved this problem by adding new stereotypes, tagged values and some OCL constraints. The new defined modeling elements were used to represent the important concepts of sequential and shared memory basic constructs which allows modeling enormous applications.

The following tables summarize all of the discussed extensions.

Table 3.15: UML extensions sorted by domain

Ref.	Domain	Purpose of Extension	Type of Extension	Diagram
Agents (2)				

Fei and Yan 2008 [16]	Agent UML	Enhance the analysis and design of an agent system.	Lightweight	Use Case Diagram
El-kady et al. 2008 [18]	Multi-agent systems	Represent the MAS conceptual model.	Heavyweight	Class Diagram
Aspect-Oriented (3)				
Zhou et al. 2008 [14]	Aspect-oriented modeling (AOM)	Model the functional crosscutting concerns and integrate the AOM architecture.	Lightweight	Sequence Diagram
Przybylek 2008 [21]	Aspect-oriented modeling (AOM)	Support aspect-oriented modeling by adding its concepts to the design phase.	Heavyweight	Class Diagram
Sharafi et al. 2010 [29]	Aspect-oriented modeling (AOM)	Capture crosscutting concerns.	Lightweight	Class Diagram
Component-based (2)				
Mahmood and Lai 2009 [28]	Component-based software system	Specify satisfaction and risk assessment to evaluate customer demands against component features.	Lightweight	Class Diagram
Cortellessa and Pompei 2004 [32]	Component-based systems	Integrate UML profiles for Quality of Service and Fault Tolerance.	Lightweight	Use Case & Sequence Diagram
Design Pattern (2)				
Dong 2002 [25]	Design patterns	Represent design patterns in the applications and compositions of design patterns and maintain pattern-related information.	Lightweight	Class Diagram
Sanada and Adams 2002 [26]	Design patterns	Model design patterns and frameworks in design class diagrams (DCDs).	Lightweight	Class Diagram
Others (14)				
Jantan et al. 2008 [27]	Web hypermedia applications	Model complicated design issues.	Lightweight	Class & Activity Diagram
Romero et al. 2007	Open distributed processing (ODP)	Provide notations to be used in the individual	Lightweight	Class, Component,



[34]	computational viewpoint.	viewpoints.		Activity, State Chart, Interaction Diagram
Fontoura et al. 2000 [22]	Object-oriented frameworks	Model variation points in UML diagrams.	Lightweight	Class & Sequence Diagram
Byeon et al. 2004 [23]	Global navigation satellite system	Provide notational help to accurate calculations of real-world geographical entities.	Lightweight	Class Diagram
Peterson et al. 2006 [24]	Security	Incorporate security techniques into software class design.	Lightweight	Class Diagram
Hausmann et al. 2001 [15]	UML semantics specification	Integrate extensions' specific semantic with UML semantics.	Lightweight	Sequence Diagram
Li and Lilius 1999 [38]	Time analysis	Give a solution for timing analysis of sequence diagrams.	Heavyweight	Sequence Diagram
Djemaa et al. 2006 [17]	Adaptive Web Application	Model AWA	Lightweight	Use Case Diagram
Dong et al. 2002 [33]	Distributed systems	Change Use Case Diagram to multi-level for requirement engineering of distributed system.	Lightweight	Use Case Diagram
Andre et al. 2007 [35]	Real-time embedded applications	Model time-dependent events and behaviors.	Lightweight	State Chart Diagram
Simons and Wirtz 2007 [13]	Mobile distributed systems	Model context for mobile distributed systems.	Lightweight	Class Diagram
Fernandez-medina et al. 2007 [12]	Data warehouses	Address confidentiality problems and set security constraints in the conceptual modeling of data warehouses.	Lightweight	Class Diagram
Majzik et al. 2004 [36]	Time triggered systems	Integrate time-triggered (TT) systems' environment with visual design tools.	Lightweight	Class & Activity Diagram
Pllana and Fahringer	Parallel applications	Model structural and behavioral patterns of	Lightweight	Activity Diagram

2002 [37]		parallel programming paradigms.		
-----------	--	---------------------------------	--	--

Table 3.16: UML extensions sorted by type of extension

Ref.	Domain	Purpose of Extension	Type of Extension	Diagram
Lightweight (20)				
Jantan et al. 2008 [27]	Web hypermedia applications	Model complicated design issues.	Lightweight	Class & Activity Diagram
Fei and Yan 2008 [16]	Agent UML	Enhance the analysis and design of an agent system.	Lightweight	Use Case Diagram
Romero et al. 2007 [34]	Open distributed processing (ODP) computational viewpoint.	Provide notations to be used in the individual viewpoints.	Lightweight	Class, Component, Activity, State Chart, Interaction Diagram
Zhou et al. 2008 [14]	Aspect-oriented modeling (AOM)	Model the functional crosscutting concerns and integrate the AOM architecture.	Lightweight	Sequence Diagram
Fontoura et al. 2000 [22]	Object-oriented frameworks	Model variation points in UML diagrams.	Lightweight	Class & Sequence Diagram
Byeon et al. 2004 [23]	Global navigation satellite system	Provide notational help to accurate calculations of real-world geographical entities.	Lightweight	Class Diagram
Peterson et al. 2006 [24]	Security	Incorporate security techniques into software class design.	Lightweight	Class Diagram
Hausmann et al. 2001 [15]	UML semantics specification	Integrate extensions' specific semantic with UML semantics.	Lightweight	Sequence Diagram
Djemaa et al. 2006 [17]	Adaptive Web Application	Model AWA	Lightweight	Use Case Diagram
Dong et al. 2002 [33]	Distributed systems	Change Use Case Diagram to multi-level for requirement engineering of distributed system.	Lightweight	Use Case Diagram

Andre et al. 2007 [35]	Real-time embedded applications	Model time-dependent events and behaviors.	Lightweight	State Chart Diagram
Simons and Wirtz 2007 [13]	Mobile distributed systems	Model context for mobile distributed systems.	Lightweight	Class Diagram
Mahmood and Lai 2009 [28]	Component-based software system	Specify satisfaction and risk assessment to evaluate customer demands against component features.	Lightweight	Class Diagram
Sharafi et al. 2010 [29]	Aspect-oriented modeling (AOM)	Capture crosscutting concerns.	Lightweight	Class Diagram
Fernandez-medina et al. 2007 [12]	Data warehouses	Address confidentiality problems and set security constraints in the conceptual modeling of data warehouses.	Lightweight	Class Diagram
Dong 2002 [25]	Design patterns compositions	Represent design patterns in the applications and compositions of design patterns and maintain pattern-related information.	Lightweight	Class Diagram
Sanada and Adams 2002 [26]	Design patterns	Model design patterns and frameworks in design class diagrams (DCDs).	Lightweight	Class Diagram
Majzik et al. 2004 [36]	Time triggered systems	Integrate time-triggered (TT) systems' environment with visual design tools.	Lightweight	Class & Activity Diagram
Cortellessa and Pompei 2004 [32]	Component-based systems	Integrate UML profiles for Quality of Service and Fault Tolerance.	Lightweight	Use Case & Sequence Diagram
Pllana and Fahringer 2002 [37]	Parallel applications	Model structural and behavioral patterns of parallel programming paradigms.	Lightweight	Activity Diagram
Heavyweight (3)				

Przybylek 2008 [21]	Aspect-oriented modeling (AOM)	Support aspect-oriented modeling by adding its concepts to the design phase.	Heavyweight	Class Diagram
El-kady et al. 2008 [18]	Multi-agent systems	Represent the MAS conceptual model.	Heavyweight	Class Diagram
Li and Lilius 1999 [38]	Time analysis	Give a solution for timing analysis of sequence diagrams.	Heavyweight	Sequence Diagram

Table 3.17: UML extensions sorted by diagram

Ref.	Domain	Purpose of Extension	Type of Extension	Diagram
Class (9)				
Przybylek 2008 [21]	Aspect-oriented modeling (AOM)	Support aspect-oriented modeling by adding its concepts to the design phase.	Heavyweight	Class Diagram
El-kady et al. 2008 [18]	Multi-agent systems	Represent the MAS conceptual model.	Heavyweight	Class Diagram
Byeon et al. 2004 [23]	Global navigation satellite system	Provide notational help to accurate calculations of real-world geographical entities.	Lightweight	Class Diagram
Peterson et al. 2006 [24]	Security	Incorporate security techniques into software class design.	Lightweight	Class Diagram
Simons and Wirtz 2007 [13]	Mobile distributed systems	Model context for mobile distributed systems.	Lightweight	Class Diagram
Sharafi et al. 2010 [29]	Aspect-oriented modeling (AOM)	Capture crosscutting concerns.	Lightweight	Class Diagram
Fernandez-medina et al. 2007 [12]	Data warehouses	Address confidentiality problems and set security constraints in the conceptual modeling of data warehouses.	Lightweight	Class Diagram
Dong 2002 [25]	Design patterns	Represent design patterns in the applications and	Lightweight	Class Diagram

		compositions of design patterns and maintain pattern-related information.		
Sanada and Adams 2002 [26]	Design patterns	Model design patterns and frameworks in design class diagrams (DCDs).	Lightweight	Class Diagram
Sequence (3)				
Zhou et al. 2008 [14]	Aspect-oriented modeling (AOM)	Model the functional crosscutting concerns and integrate the AOM architecture.	Lightweight	Sequence Diagram
Hausmann et al. 2001 [15]	UML semantics specification	Integrate extensions' specific semantic with UML semantics.	Lightweight	Sequence Diagram
Li and Lilius 1999 [38]	Time analysis	Give a solution for timing analysis of sequence diagrams.	Heavyweight	Sequence Diagram
Use Case (3)				
Fei and Yan 2008 [16]	Agent UML	Enhance the analysis and design of an agent system.	Lightweight	Use Case Diagram
Djema et al. 2006 [17]	Adaptive Web Application	Model AWA	Lightweight	Use Case Diagram
Dong et al. 2002 [33]	Distributed systems	Change Use Case Diagram to multi-level for requirement engineering of distributed system.	Lightweight	Use Case Diagram
Others (8)				
Jantan et al. 2008 [27]	Web hypermedia applications	Model complicated design issues.	Lightweight	Class & Activity Diagram
Romero et al. 2007 [34]	Open distributed processing (ODP) computational viewpoint.	Provide notations to be used in the individual viewpoints.	Lightweight	Class, Component, Activity, State Chart, Interaction Diagram
Fontoura et al. 2000 [22]	Object-oriented frameworks	Model variation points in UML diagrams.	Lightweight	Class & Sequence Diagram
Andre et al. 2007	Real-time embedded applications	Model time-dependent events and	Lightweight	State Chart Diagram

[35]		behaviors.		
Mahmood and Lai 2009 [28]	Component-based software system	Specify satisfaction and risk assessment to evaluate customer demands against component features.	Lightweight	Class & Sequence Diagram
Majzik et al. 2004 [36]	Time triggered systems	Integrate time-triggered (TT) systems' environment with visual design tools.	Lightweight	Class & Activity Diagram
Cortellessa and Pompei 2004 [32]	Component-based systems	Integrate UML profiles for Quality of Service and Fault Tolerance.	Lightweight	Use Case & Sequence Diagram
Pllana and Fahringer 2002 [37]	Parallel applications	Model structural and behavioral patterns of parallel programming paradigms.	Lightweight	Activity Diagram

# CHAPTER 4

## 4. Extension Integration

In this chapter, the integration process of the previously mentioned UML extensions is provided. First, the process of integration is explained and then the process is applied to the UML extensions for the three selected models: class, sequence, use case diagrams. The results section shows the integrated diagram elements and the meta-model for each of the selected models.

### 4.1 The Integration Process

The integration process is applied to two different types of extensions; the first type addresses the UML extensions that provide graphical symbols only, and the second type goes beyond the graphical representations in the UML diagrams and deals with the proposed modeling elements that add to the meta-models. At the end of the second type of integration, the obtained graphical elements are checked for consistency. Each graphical symbol is mapped to iUML meta-model. As for the constraints, they are still valid as they accompany the modeling elements during the integration process.

The integration process of the first type, the graphical symbols type of extensions, requires a creation of a graphical library that contains the proposed graphical symbols

themselves and their descriptions. After having this collection of symbols, one can look for symbols that can be soundly integrated. To check for soundness, the symbols must not cause any graphical conflict in a way that keeps the original intent of the symbols clear. In other words, the final symbol must deliver the idea behind it without any confusion. The following process explains the integration of graphical symbols:

- 1- Creation of Library:** Create a library for the graphical symbols. The library shall contain the graphical symbols themselves and their descriptions.
- 2- Case A: Combination:** For each type of UML diagram, combine possible graphical symbols that cause no graphical conflicts but make sure that the final symbol is still displaying its intended goal.
- 3- Case B: Conflict:** In case of a graphical conflict, insert each graphical symbol on its own into the library.

The integration of the second type, the meta-model type of extensions, takes the proposed stereotypes and tag definitions and inserts them properly into the original meta-models of each model. The proper placement of modeling elements in the meta-model is crucially important. Therefore, one must correctly place the modeling elements (instances in the meta-model) under their classifiers. Categorizing these elements is also important. There are two categories of modeling elements in the meta-model; <<Stereotype>> and <<TaggedValue>>. In addition, each UML extension's elements must be clearly shown using distinguished colors. To integrate two or more extensions in the meta-model, each extension must be also clearly identified. The following process explains the integration of meta-model elements:



- 1- Adding the Elements:** Add the newly introduced modeling elements under the appropriate classifier in the meta-model. The introduced modeling element will be an instance of that classifier. The classifier describes the behavioral and the structural features and the instance describes the operations and the state of iUML meta-model elements. Adding the elements is a fundamental step. It has to be applied correctly because it affects the soundness of the resulting meta-model. Every modeling element has to be carefully and correctly placed under the appropriate classifier in the meta-model.
- 2- Categorizing the Elements:** Categorize each introduced element as <<Stereotype>> or <<TaggedValue>>. These two categories are the main categories of the extended modeling elements. This step is crucially important. Failing to correctly categorize the modeling elements will result in an invalid system model. The two categorizes are significantly different; Stereotypes represent new terminologies while Tagged Values represent properties or values to those terminologies.
- 3- Defining Meta-classes or other classifiers:** State the introduced meta-classes with the symbol [class] or other classifiers below their names and categories. The importance of this step revolves around the introduction of meta-classes and/or other classifiers, for example: Boolean, String, etc. They are essentials because they define classes or other data types in the diagrams. However, the introduction of meta-classes will only give results in the integration of UML class diagram extensions since UML sequence and use case diagrams do not include any classes.

- 4- **Case A: Combination:** Combine modeling elements of the same domain as one instance. Each modeling element must be clearly distinguished in that instance. The goal is to show the integrated modeling elements as they share the same domain. The integration process enhances the organization of the meta-model. It provides the end-user with one comprehensive domain-specific set of modeling elements.
- 5- **Case B: Conflict:** In case two extensions have a conflict, gather only the most common modeling elements from both extensions and place them in the integrated meta-model. Conflicts between extensions can be caused for example by the removal of essential UML infrastructure and superstructure elements. The goal of having this step is to resolve the conflicts that might happen between two or more extensions. The results of this step depend on the process and the results of step # 4, Case A: Integration.

## 4.2 Applying the Integration Process

In this section, the Integration Process, mentioned above, is applied to the three UML diagrams; class, sequence and use case diagram. In each sub-section, a step-by-step explanation of the Integration Process is shown.

We defined the below inclusion/exclusion criteria; only extensions that meet our inclusion criteria were included in iUML while others are excluded. The inclusion criteria:

- 1- UML lightweight and heavyweight extensions.
- 2- UML class, sequence and use case diagrams extension only.

- 3- UML domain-specific extensions that can be combined with the other same domain-specific extensions, preferably working on different areas of the extension but at the same level of extension.
- 4- UML domain-specific extensions that can be combined with the other different domain-specific extensions, preferably general extensions.
- 5- When two UML extensions focus on one particular area and on one type of UML diagram, combine them together or choose the more general one.

And the exclusion criteria are:

- 1- UML activity, component, state chart, interaction diagrams.
- 2- UML heavyweight extensions that manipulate the UML meta-model whether by editing or deleting UML packages.
- 3- Theoretical and algorithmic UML extensions.

### **4.2.1 Integration of graphical symbols**

This sub-section addresses the application of the Integration Process on the UML class, sequence and use case diagrams graphical extensions. This process has three steps; Creation of Library, Integration and Conflict. Each UML diagram will be subjected to these steps and the results will be shown as the process is applied.

#### **4.2.1.1 Class diagram**






In the literature, 9 out of the 23 reviewed extensions were applied to UML class diagram.



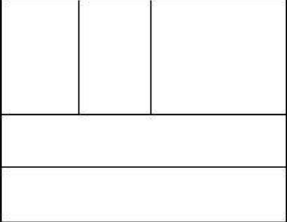
## Step 1: Creation of Library

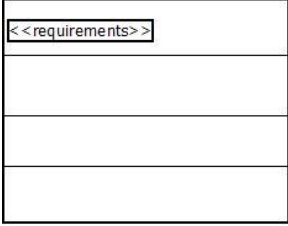
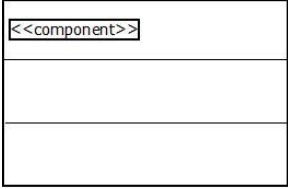

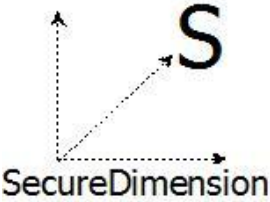
In this process of graphical integration, a library is created to include the proposed graphical extensions. All of the graphical symbols are inserted along with their descriptions. The idea behind having such library is to have a graphical database for iUML. Such database lists all the symbols and their descriptions, plus, their original source. The Description column informs the user of the intended objective of the symbol.


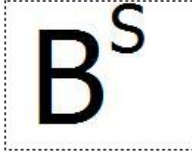
Table 4.1 shows the created library for UML class diagram graphical extensions.

Table 4.1: Library of proposed graphical symbols (class diagram)

Modeling element	Source	Use of the Symbol
 Atomic_Class	Jantan et al. 2008 [27]	Used to represent single process.
 Database_Class	Jantan et al. 2008 [27]	Used to represent database in the class diagram design.
 Database Link	Jantan et al. 2008 [27]	Used to represent the information and data operations such as query, lookup, entry, etc., that involved with database.
 Interaction Class	Jantan et al. 2008 [27]	Used to represent complex interaction between users and web application.
 Navigational Class	Jantan et al. 2008 [27]	Used to represent hyperlinks in the class diagram design.

 NonAtomic_Class	Jantan et al. 2008 [27]	Used to represent pre-defined and complex processes.
 Process_Class	Jantan et al. 2008 [27]	Used to represent the user's action to perform activities.
{variable}	Fontoura et al. 2000 [22]	Used to represent the implemented methods during the framework instantiation.
{appl-class}	Fontoura et al. 2000 [22]	Used to represent classes that are defined as framework instances.
{extensible}	Fontoura et al. 2000 [22]	Used to represent the extensibility of class functionality.
{static}	Fontoura et al. 2000 [22]	Used to represent variation points of non-runtime instantiation.
{dynamic}	Fontoura et al. 2000 [22]	Used to represent variation points of runtime instantiation.
{incomplete}	Fontoura et al. 2000 [22]	Used to represent the possibility of adding new subclasses.
{forAllNewMethods}	Fontoura et al. 2000 [22]	Used to indicate that the OCL constraint must be met by the introduced methods.
{optional}	Fontoura et al. 2000 [22]	Used to indicate optional event.
{final}	Sanada and Adams 2002 [25, 26]	Used to indicate that the final class has no decedent classes (leaves).
	Byeon et al. 2004 [23]	The geo-referenced class is used to represent the class icon with the aid of graphical notations. The main elements of geo-referenced classes are: a graphical representation with a symbolistic icon, an

		<p>iconic notation to indicate the geographic type, class name, attributes and operations.</p>
	<p>Mahmood and Lai 2009 [28]</p>	<p>RClass is used to represent to stakeholder requirements, and it is divided into four sections: First, stereotyped requirement text, name of the class and abstraction level to differentiate the requirement level. Secondly, the objective to of the RClass. Thirdly, scenario which is the set of interactions necessary to achieve the objective. The last one is rank of the RClass.</p>
	<p>Mahmood and Lai 2009 [28]</p>	<p>CClass is used to represent component features, and it is divided into three sections: First, stereotyped component text and name of the class. Secondly, the functionality provided by the component. The last section is the dependency on elements and their relationships.</p>
	<p>Fernandez-medina et al.2007 [12]</p>	<p>Used to represent security information and constraints.</p>
	<p>Fernandez-medina et al.2007 [12]</p>	<p>Used to represent dimensions within a multidimensional model.</p>

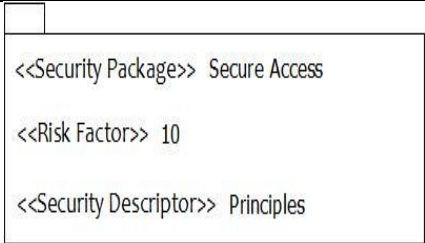
 <p>SecureFact</p>	<p>Fernandez-medina et al.2007 [12]</p>	<p>Used to represent facts within a multidimensional model.</p>
 <p>SecureBase</p>	<p>Fernandez-medina et al.2007 [12]</p>	<p>Used to represent dimension hierarchy levels within a multidimensional model.</p>

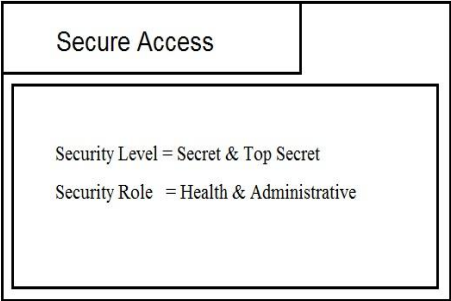
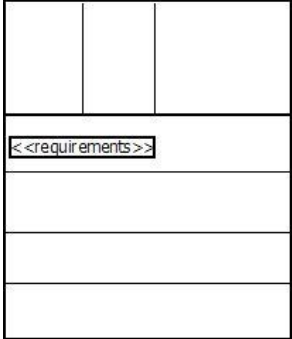
### Step 2: Case A: Combination

If some of the already existing symbols in the library can be combined together with other existed symbols, combine them both into one symbol and add that symbol to the library.

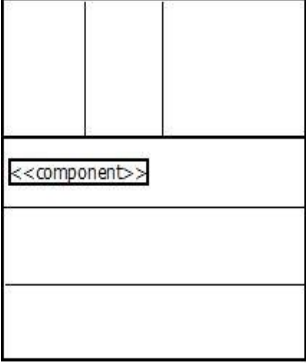
Table 4.2 shows the integrated graphical symbols.

Table 4.2: Integrated graphical extensions

Modeling element	Source	Use of the Symbol	Method of Combination
	<p>Peterson et al. [24] and Fernandez-Medina et al. [12]</p>	<p>The security package will be inserted into the class diagram and will be attached to the classes that need to be protected from security attacks. Each security package has three attributes: Risk Factor; which calculates the probability the security attack, Security Tile; protects the main parts of a system</p>	<p>The design of the security package was adopted from the work of Peterson et al. [24]. While the security information were suggested by Fernandez-Medina et al. in [12].</p>

		and finally, Security Descriptor: protects specific parts of the system.	
	Peterson et al. [24] and Fernandz-Medina et al. [12]	A Security Tile which protects the main parts of the system. It mostly contains tagged values specified by security analysts and it can be attached to security packages to cover more security concerns.	The design of the security package was adopted from the work of Peterson et al. [24]. While the security information was suggested by Fernandz-Medina et al. in [12].
	Byeon et al. 2004 [23]& Mahmood and Lai 2009 [28]	The new main elements of the class are three vertical compartments to indicate symbolic icons, iconic notations and class name, and <<requirements>> to specify stakeholder requirements. It will be used to represent requirements with the aid of graphical notations.	The three vertical compartments that will contain some graphical and textual information was suggested by Byeon et al. [23]. The requirements stereotype and the other requirements-related information were proposed by Mahmood and Lai in [28].



	<p>Byeon et al. 2004 [23]&amp; Mahmood and Lai 2009 [28]</p>	<p>The new main elements of the class are three vertical compartments to indicate symbolic icons, iconic notations and class name, and &lt;&lt;component&gt;&gt; to specify component features. It will be used to represent components with the aid of graphical notations.</p>	<p>The three vertical compartments that will contain some graphical and textual information was suggested by Byeon et al. [23]. The component stereotype and the other component-related information were proposed by Mahmood and Lai in[28].</p>
---	--	--	---

### Step 3: Case B: Conflict

If a graphical conflict happens between two or more extensions, these extensions will be inserted individually in the library. This case happens when the final integrated symbol becomes unclear due to the process of integration. In the process of integrating UML class diagram no graphical extensions found to have conflict.

#### 4.2.1.2 Sequence diagram


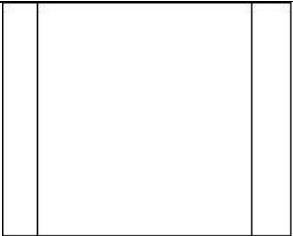
In the literature, 3 out of the 23 reviewed extensions were applied to UML sequence diagram.

## Step 1: Creation of Library

The following table, Table 4.3, shows the created library for UML sequence diagram graphical extensions.

Table 4.3: Library of proposed graphical symbols (sequence diagram)

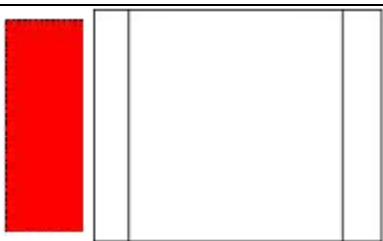
Modeling element	Source	Use of the Symbol
{variable}	Fontoura et al. 2000 [22]	Used to represent the methods that must be implemented during the framework instantiation.
{appl-class}	Fontoura et al. 2000 [22]	Used to represent classes that are defined and used as framework instances.
{extensible}	Fontoura et al. 2000 [22]	Used to represent the extensibility of class functionality.
{static}	Fontoura et al. 2000 [22]	Used to represent variation points of non-runtime instantiation.
{dynamic}	Fontoura et al. 2000 [22]	Used to represent variation points of runtime instantiation.
{incomplete}	Fontoura et al. 2000 [22]	Used to represent the possibility of adding new subclasses.
{forAllNewMethods}	Fontoura et al. 2000 [22]	Used to indicate that the OCL constraint is meant to hold for all newly introduced methods.
{optional}	Fontoura et al. 2000 [22]	Used to indicate that a given event is optional.
{final}	Sanada and Adams 2002 [25, 26]	Used to indicate that the final class has no decedent classes (leaves).

	Zhou et al. 2008 [14]	Crosscutting bar to indicate join points between two events
	Hausmann et al. 2001 [15]	Synchronization bold bars to be placed between activations. They mean that the activities must start and end at the same time.

**Step 2: Case A: Combination**

The result of this step is one integrated symbol. Table 4.4 shows that symbol.

Table 4.4: Integrated graphical extension

Modeling element	Source	Use of the Symbol	Method of Integration
	Zhou et al. 2008 [14] and Hausmann et al. 2001 [15]	The red crosscutting bar indicates join points that must start and end at the same time.	The red crosscutting bar was suggested by Zhou et al. [14] to show the join points between two events. Hausmann et al. [15] proposed the other graphical symbol to enforce synchronization between two activities. Both symbols focus on the start time of the activity, hence, the final integrated

			symbol indicates synchronizing join points.
--	--	--	---

### Step 3: Case B: Conflict

The only process of integration that was attempted was the one in Table 4.4 and it did not cause any conflict.

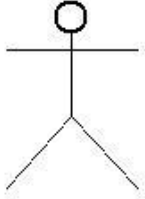
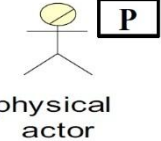
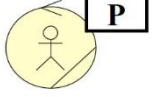
#### 4.2.1.3 Use case diagram





In the literature, 3 out of the 23 reviewed extensions were applied to UML use case diagram.

#### Step 1: Creation of Library

The following table, Table 4.5, shows the created library for UML use case diagram graphical extensions.

Table 4.5: Library of proposed graphical symbols (use case diagram)

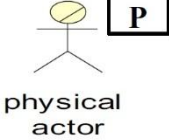
Modeling element	Source.	Use of the Symbol
 <Agent>	Fei and Yan 2008 [16]	Used to represent agents.
 physical actor	Djema et al. 2006 [17]	Used to represent the human user who visits the web application.
 Logical actor	Djema et al. 2006 [17]	Used to represent the role played by a human user (physical actor) to maintain the web application.

 <p>System actor</p>	Djemaa et al. 2006 [17]	Used to represent the hardware aspect of the system, whether it is a computer system, device hardware or web service.
 <p>Name Name-DIF</p>	Djemaa et al. 2006 [17]	SIF: Static Informational Functionality used to represent a static Web page.
 <p>Name-SIF</p>	Djemaa et al. 2006 [17]	DIF: Dynamic Informational Functionality used to represent a dynamic Web page.
 <p>Name-PF</p>	Djemaa et al. 2006 [17]	PF: Profession Functionality used to represent a dynamic Web page using update request

## Step 2: Case A: Combination

The result of this step is one integrated symbol. Table 4.6 shows that symbol.



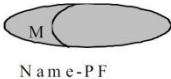
Table 4.6: Integrated graphical extension

Modeling element	Source	Use of the Symbol	Method of Integration
 <p>physical actor</p>	Fei and Yan 2008 [16] and Djemaa et al. 2006 [17]	Used to represent the human user who visits the web application. It could also represent agents in agent-oriented systems.	The human user symbol suggested by Djemaa et al. in [17] is more general, hence, can represent agents in agent-oriented systems.

### Step 3: Case B: Conflict

One conflict occurred during the attempt of integrating three graphical extensions. Table 4.7 Table 4.7 shows the three symbols that could not be integrated.

Table 4.7: The three extended functionalities proposed by Djemaa et al. 2006 [16]

Modeling element	Source	Use of the Symbol
 Name-DIF	Djemaa et al. 2006 [17]	SIF: Static Informational Functionality used to represent a static Web page.
 Name-SIF	Djemaa et al. 2006 [17]	DIF: Dynamic Informational Functionality used to represent a dynamic Web page.
 Name-PF	Djemaa et al. 2006 [17]	PF: Profession Functionality used to represent a dynamic Web page using update request

The goal behind integrating these functionalities was to have one abstract use case. But during the creation of the diagram, the abstract use case would make the diagram confusing because every time there will be a need for a specific functionality; one has to refer to the abstract use case. In conclusion, it is better to have three independent functionalities where each one presents a different type of information.

### 4.2.2 Integration of the meta-model extensions

The goal in this type of integration is to integrate the proposed modeling elements (stereotypes and tag definitions) into the original UML class, sequence and use case diagram meta-models.

#### 4.2.2.1 Class diagram

The reviewed literature contains 23 extensions; 9 of which are applied to UML class diagram meta-model. Table 4.8 shows the modeling elements accompanied with some constraints from the class diagram extensions. The main objective of this table is to show constituting elements of the integrated class diagram along with their specified constraints.

Table 4.8: The modeling elements of UML class diagram extensions

Modeling Element	Extended from (Meta Class)	Use of the Modeling Element	Associated Constraints
ContextItem [13]	Class	Models the types of context items.	Must have a basic type (Integer or String), a composite type, an enumeration type or another context item type.
ContextItemEnum [13]	Enumeration	Models the types of context items enumeration.	Must have a type with a stereotype CompositeType.
ContextAssociation [13]	Association	Models the characteristics of context items.	Must have one stereotype to represent the access prevention and one stereotype to represent the source.
Aspect [29]	Class	Models static and dynamic features.	Has a behavioral feature (Advice) and an operation (Pointcut).
Advice [29]	-- Behavioral feature (Operation)	Encapsulates behavior during the execution.	Defined by (after, before, around) and attached to (Pointcut).
Pointcut [29]	-- Behavioral feature (Operation)	Defines a place during the execution where the aspect interacts with the core functionally.	
Level [12]	Enumeration	Orders enumeration of the security levels.	Must have a correct value of tagged

			values.
Levels [12]	Primitive	Represents an interval of upper and lower levels.	Must have a correct value of tagged values.
Role [12]	Primitive	Represents the hierarchy of user roles.	
Compartment [12]	Enumeration	Enumerates the user compartments.	
Privilege [12]	Enumeration	Orders enumeration of the privileges.	
AccessAttempt [12]	Enumeration	Orders enumeration of the access attempts.	
InstanceClass [25, 26]	Class	Models the varying concepts encapsulated by the pattern.	
ApplicationClass [25, 26]	Class	Models the framework classes (instances).	
ForAllNewMethods [25, 26]	Constraint	Models that the constraint must be held for all the new methods.	
Hook [25, 26]	Method	Models the role of methods in the pattern. (Supply the concrete implementation)	
Template [25, 26]	Method	Models the role of methods in the pattern. (Define the generic instantiation in interaction between classes)	

### Step 1: Adding the Elements

The goal of this step is to add the newly introduced modeling element under the appropriate classifier in the meta-model. The introduced modeling element will be an instance of that classifier. The classifier describes the behavioral and structural features and the instance describes the operations and the state. Due to its large size, the meta-model will be divided into three parts to show the addition of extensions' modeling elements. The first part of the meta-model is shown in Figure 4.1. The white boxes in



Figure 4.1 represent the original elements of UML and the colored ones represent the extensions. Figure 4.2 shows the second part of the meta-model and Figure 4.3 shows the last part.

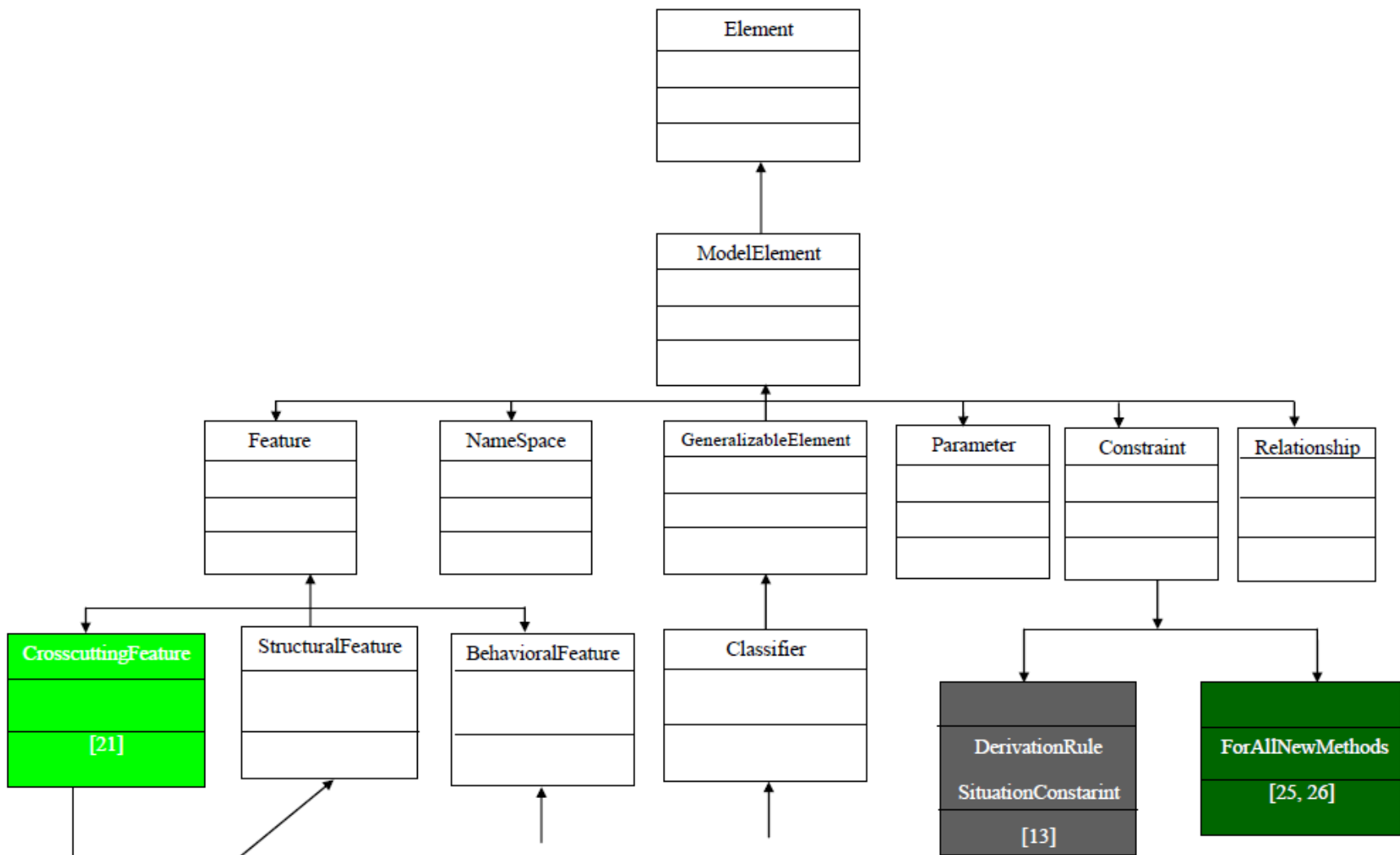


Figure 4.1: First part of original UML class diagram meta-model elements and integrated elements



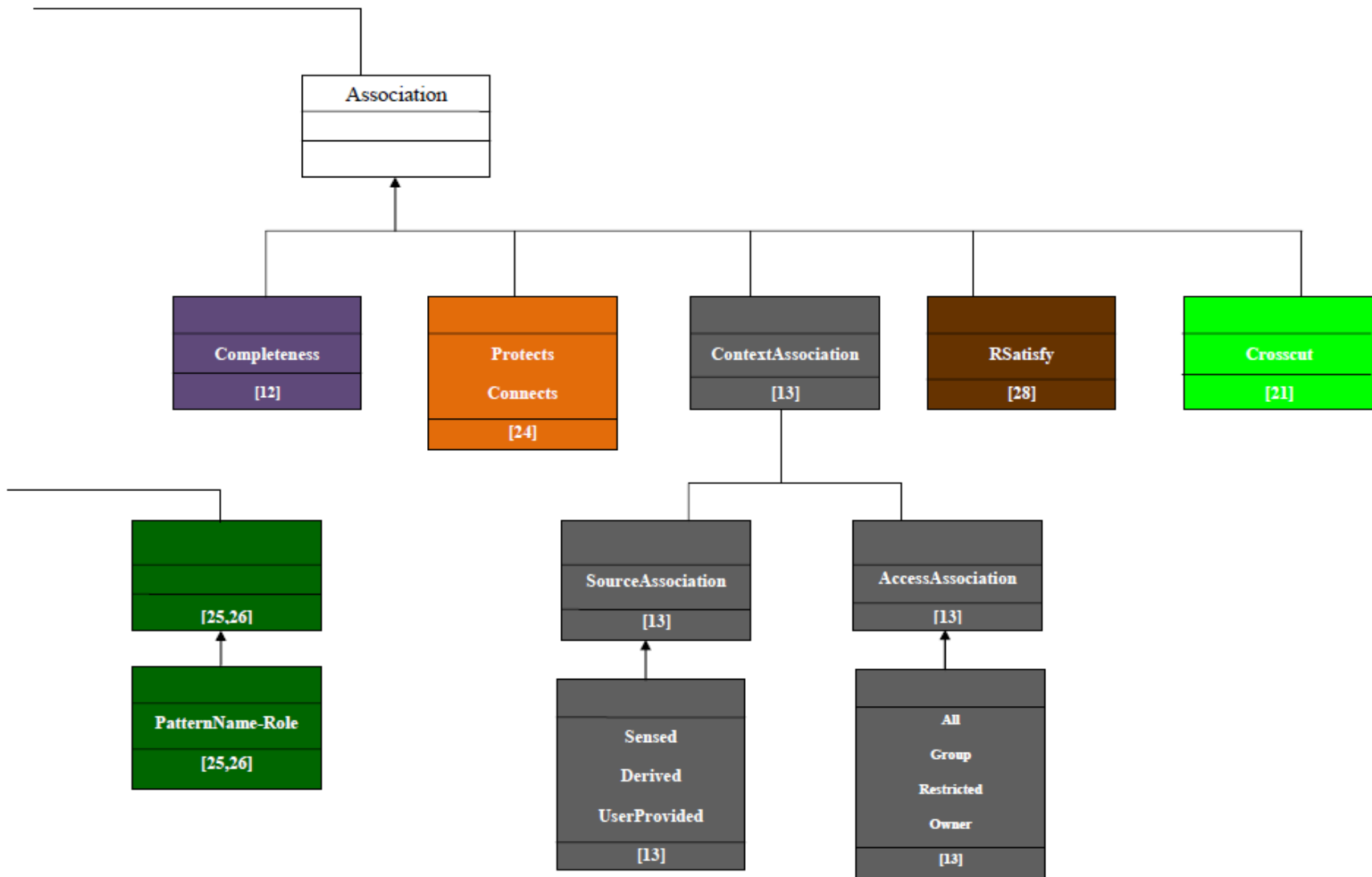


Figure 4.3: Third part of original UML class diagram meta-model elements and integrated elements

## **Step 2: Categorizing the Elements**

Categorize each introduced element as <<Stereotype>> or <<TaggedValue>>. Due to the large size of the meta-model, categorizing the extensions' modeling elements will be done in three parts. The first part of the meta-model is shown in Figure 4.4. The second part is shown in Figure 4.5 and the third part is shown in Figure 4.6.

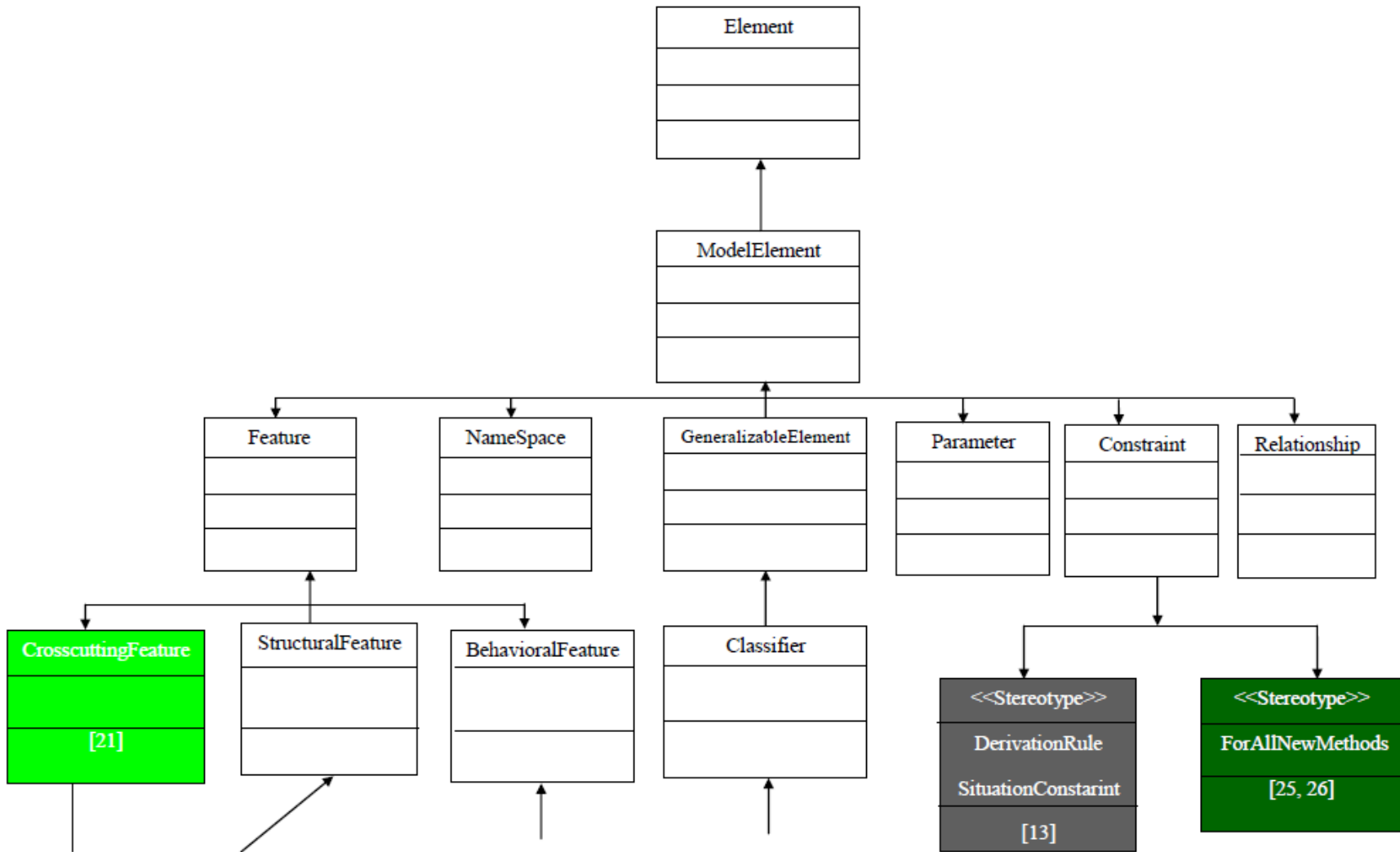


Figure 4.4: Stereotype and Tagged Value categories applied to the first part of modeling elements

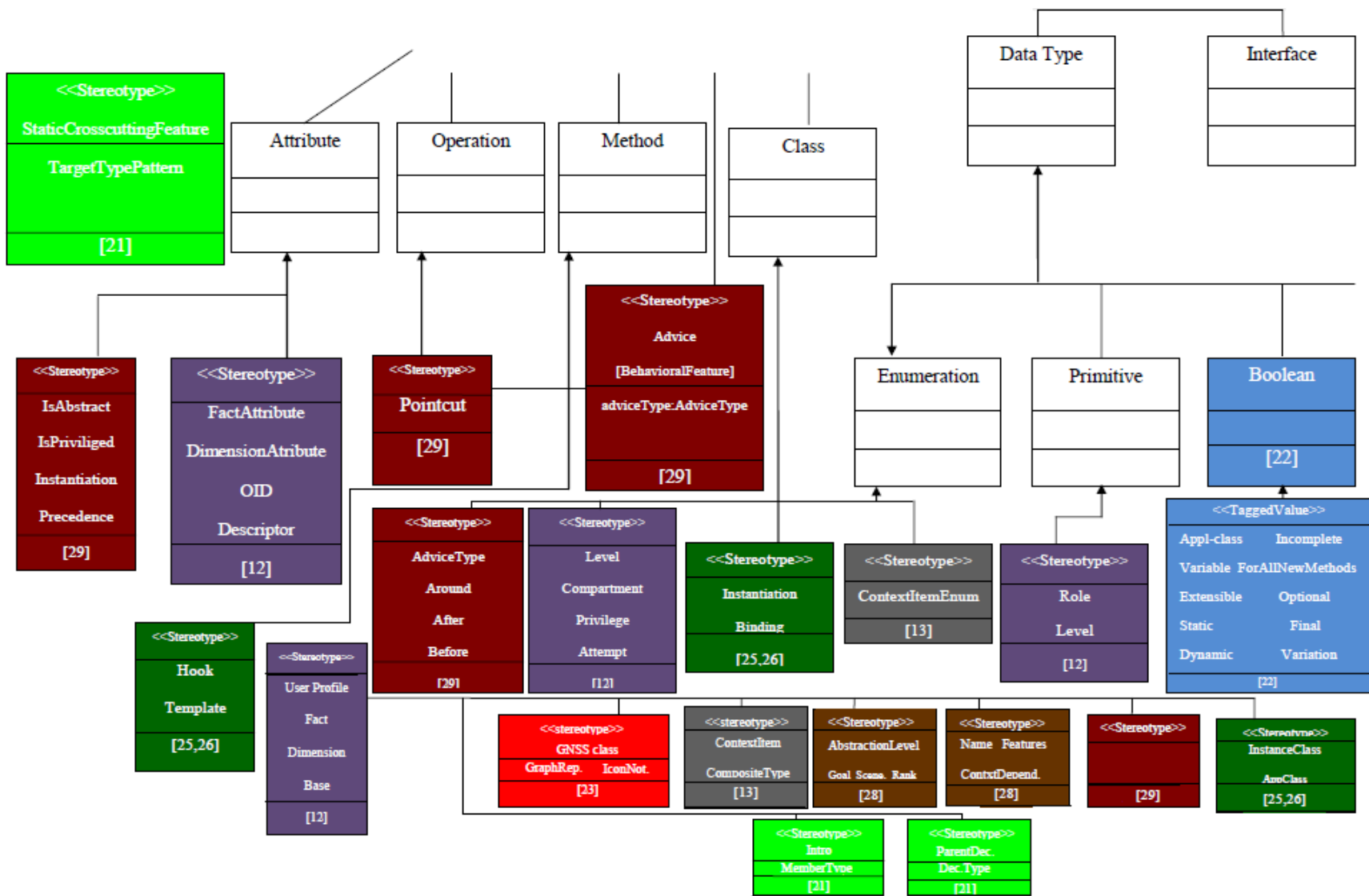


Figure 4.5: Stereotype and Tagged Value categories applied to the second part of modeling elements

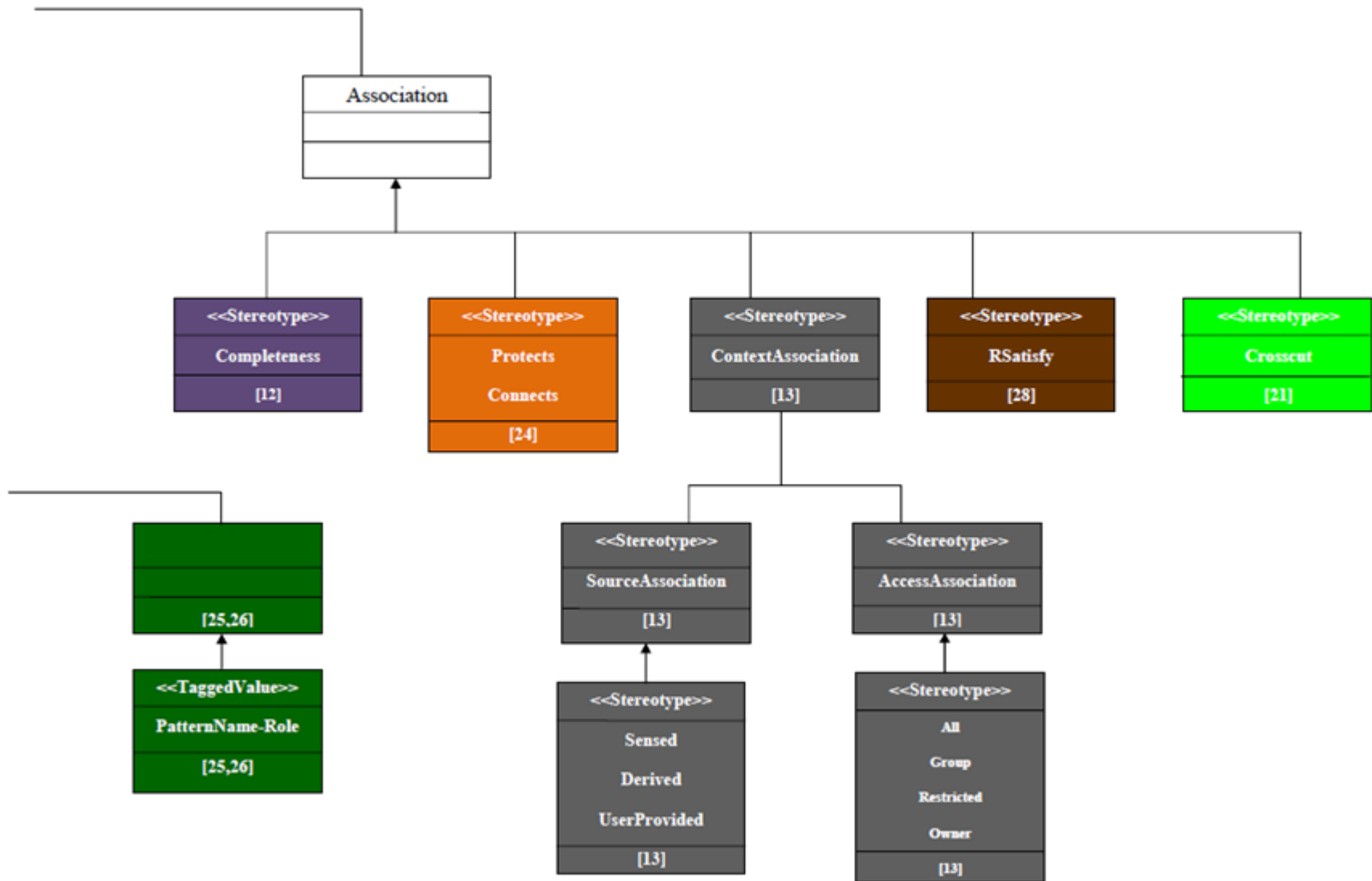


Figure 4.6: Stereotype and Tagged Value categories applied to the third part of modeling elements



### **Step 3: Defining Meta-classes**

State the meta-classes with the symbol [class] below its name and category. Due to the large of the meta-model, defining meta-classes will be done in three parts. The first part of the meta-model is shown in Figure 4.7. The second part is shown in Figure 4.8 and the third part is shown in Figure 4.9.

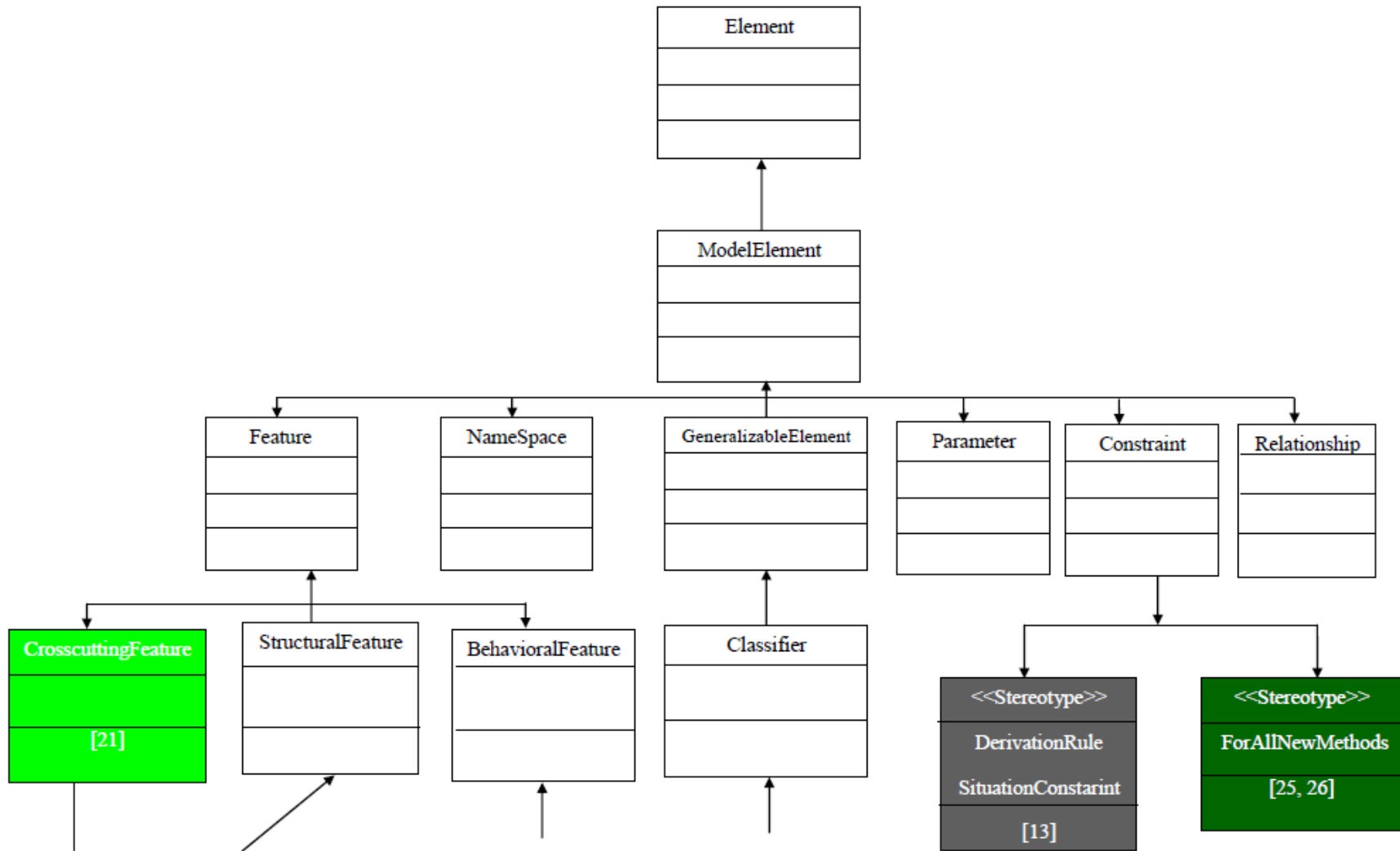


Figure 4.7: Meta-classes defined in the first part of modeling elements

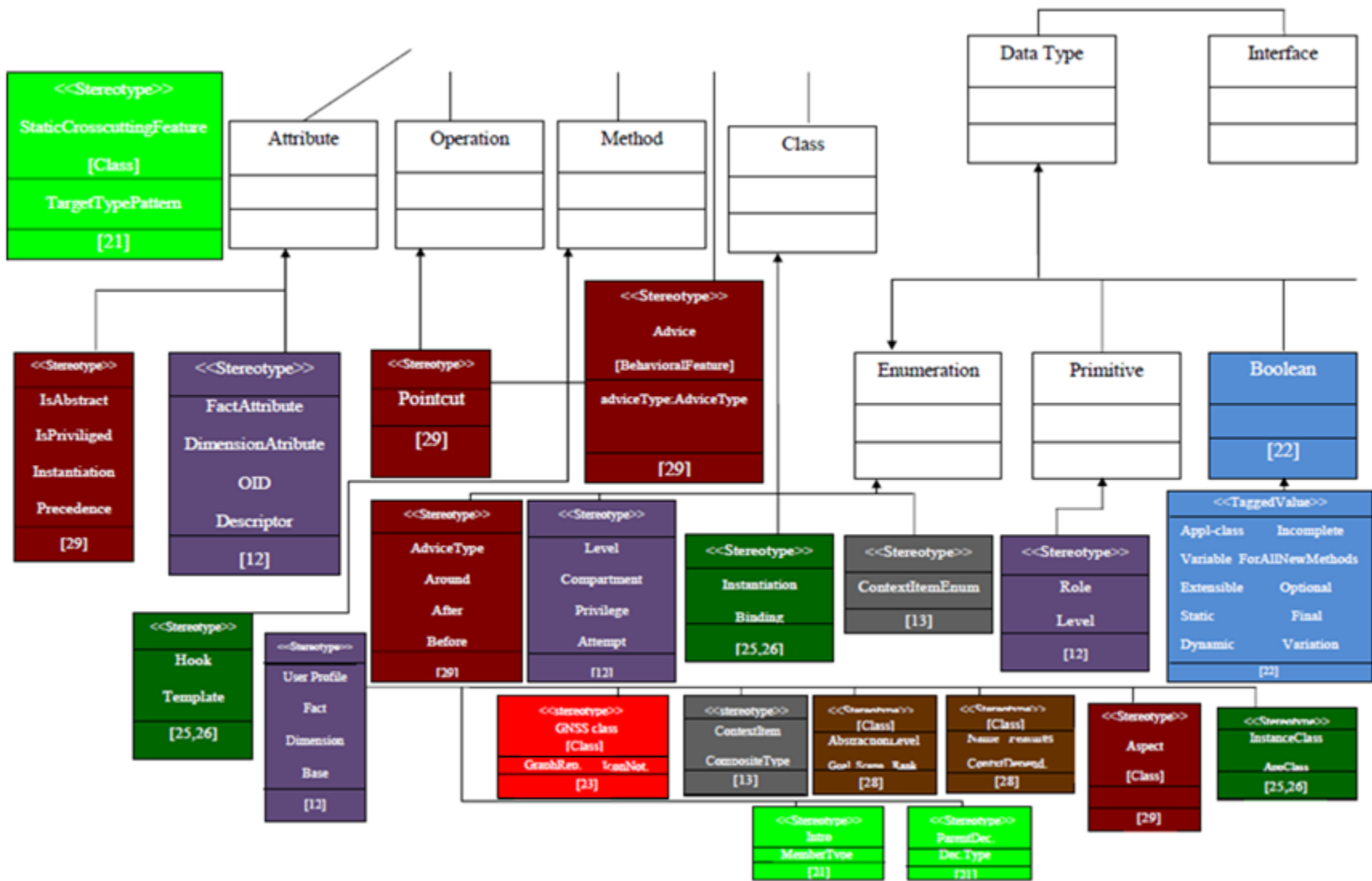


Figure 4.8: Meta-classes defined in the second part of modeling elements

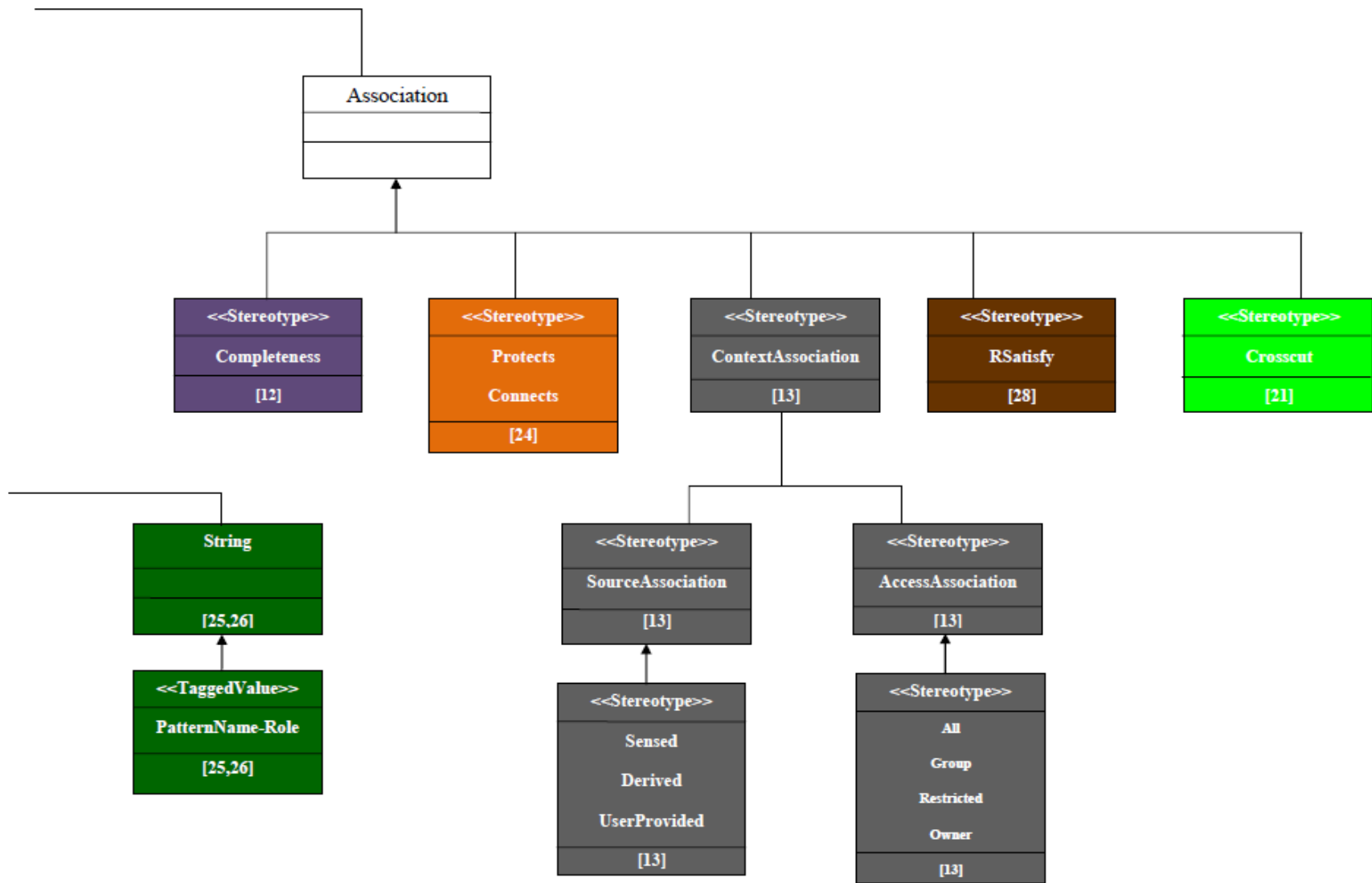


Figure 4.9: Meta-classes defined in the third part of modeling elements

#### **Step 4: Case A: Combination**

Combine modeling elements of the same domain as one instance. Each modeling element must be clearly distinguished in that instance. Due to the large size of the meta-model, combining extensions' modeling elements will be done in three parts. The first part of the meta-model is shown in Figure 4.10. The second part is shown in Figure 4.11 and the third part is shown in Figure 4.12.

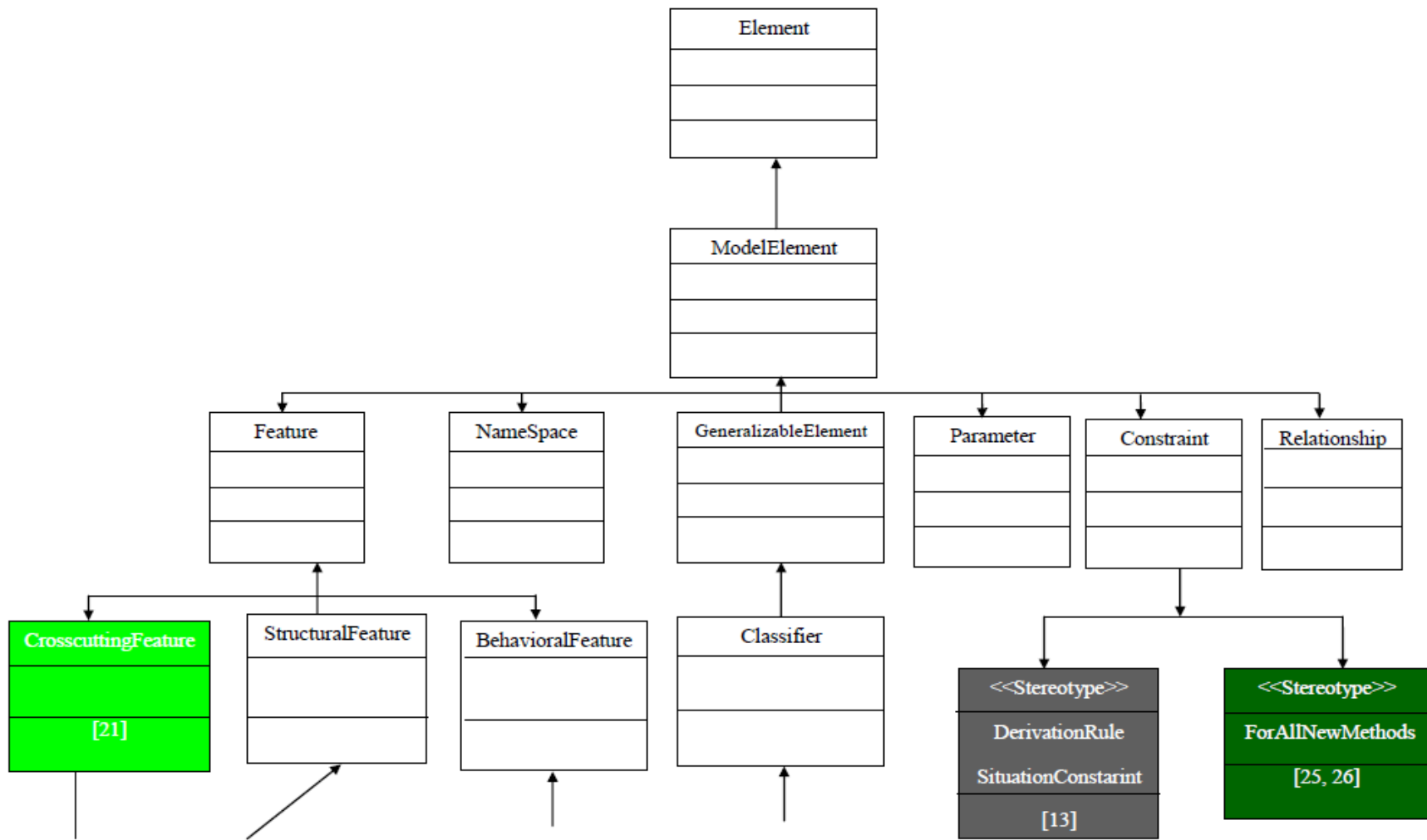


Figure 4.10: The first part of integrated domain model elements

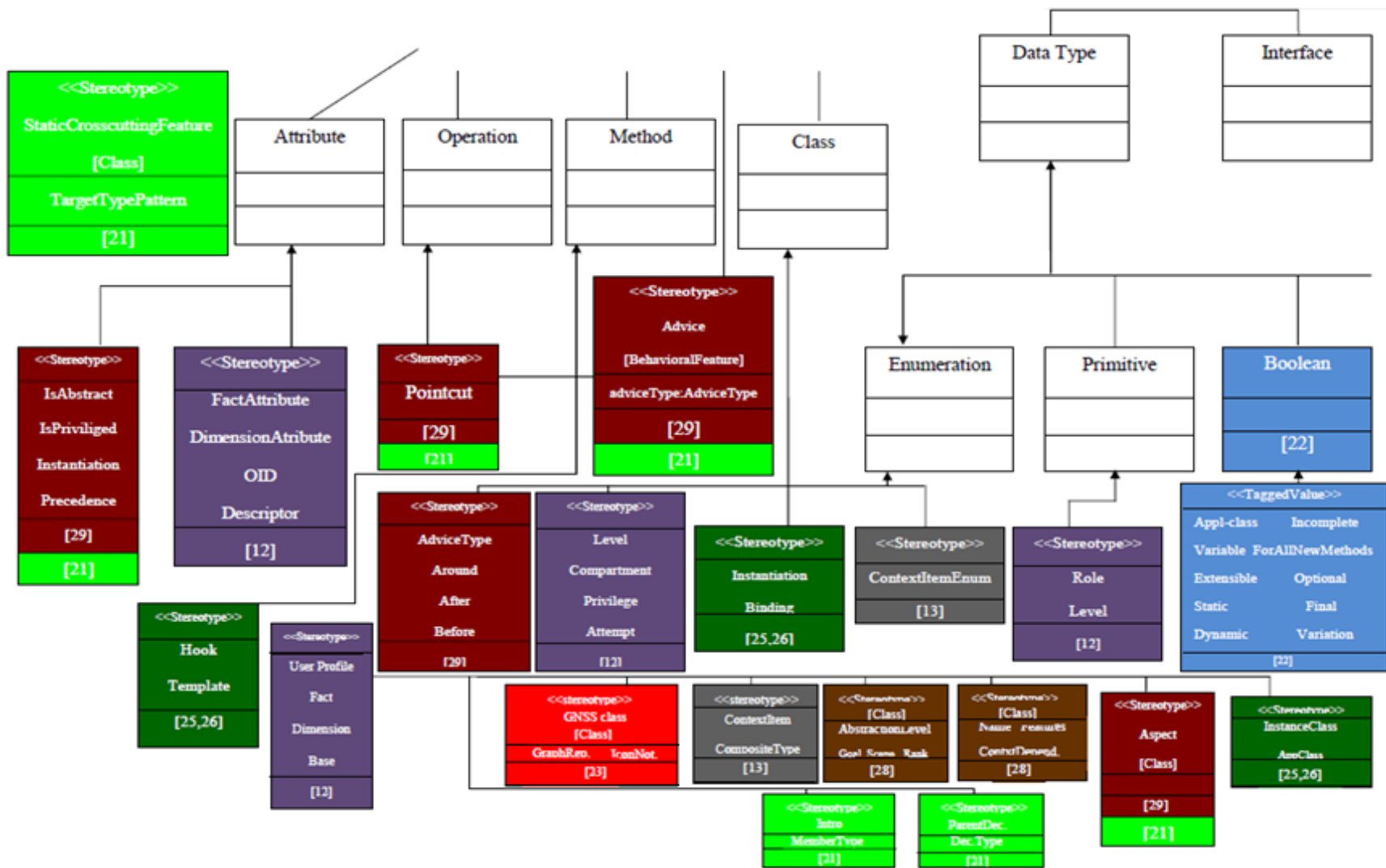


Figure 4.11: The second part of integrated domain model elements

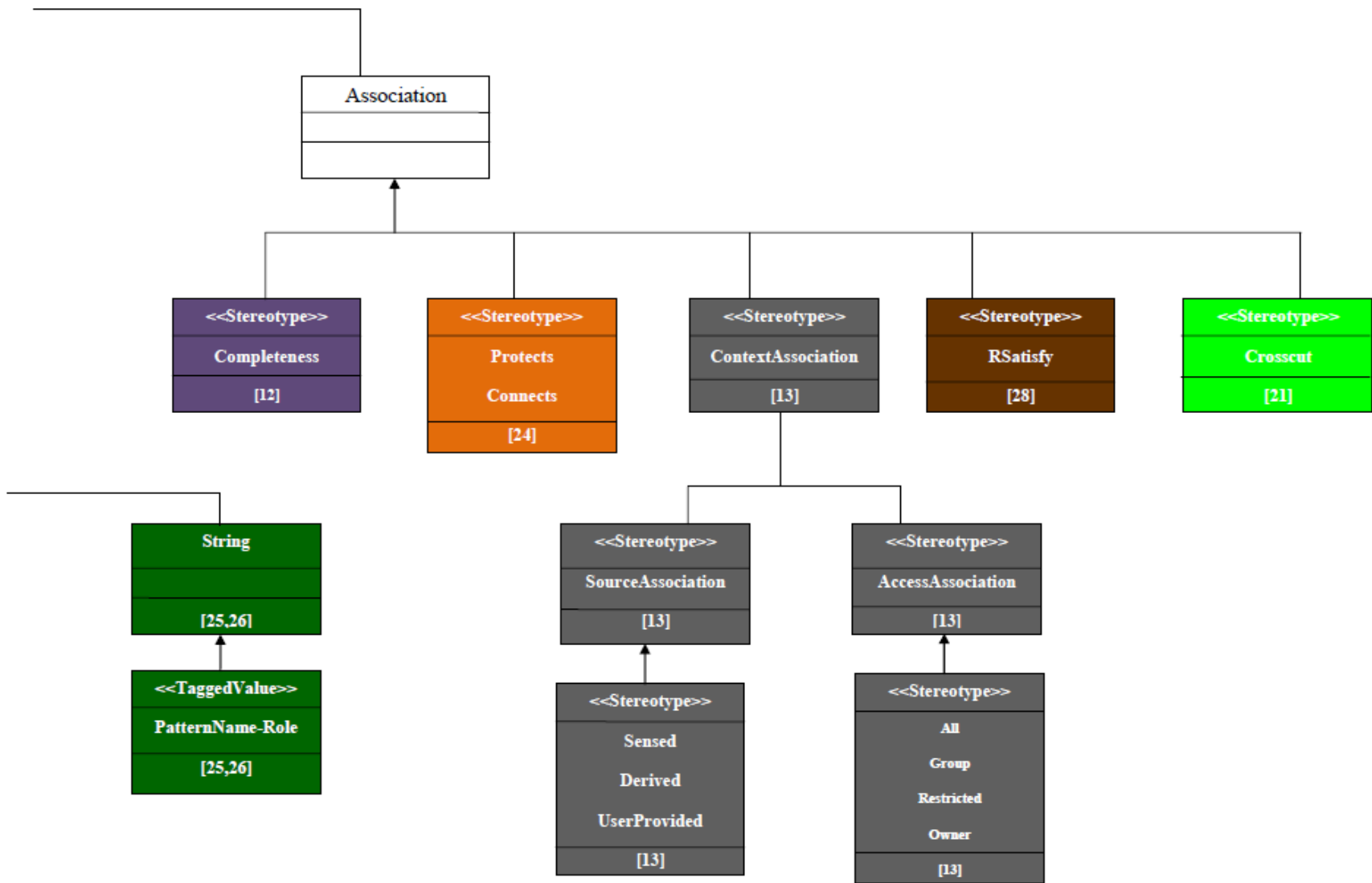


Figure 4.12: The third part of integrated domain model elements



Some extensions not only add instances to the meta-model but also classifiers. Such classifiers are considered instances of the original UML meta-model classifiers. For example, in Figure 4.13, Przybylek [21] defined Crosscutting Feature as a meta-class derived from the element Feature. From that meta-class a stereotype named Static Crosscutting Feature was presented.

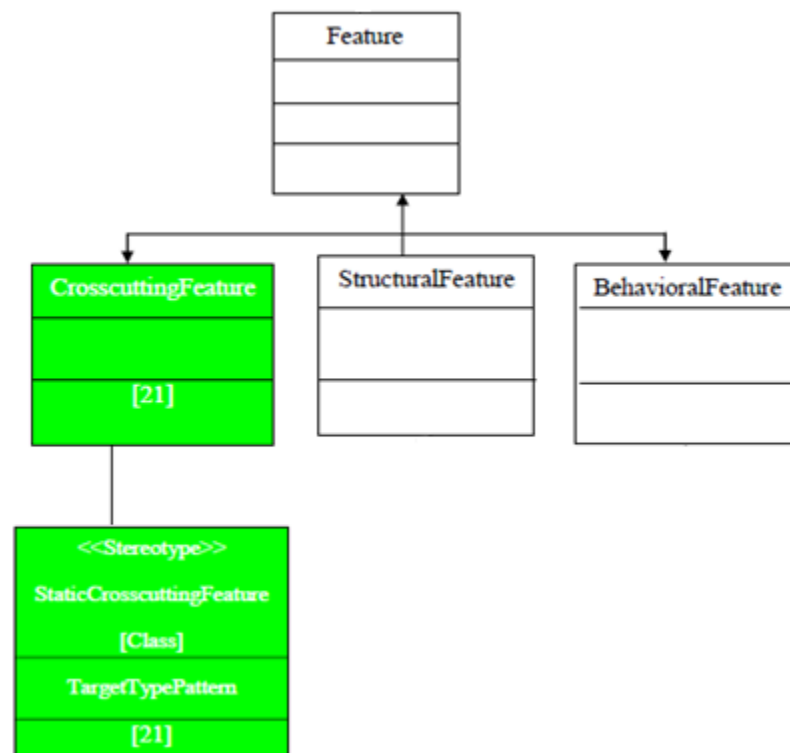


Figure 4.13: Crosscutting Feature derivation

### Step 5: Case A: Conflict

In case if a conflict occurred between two or more extensions, the two extensions should be reviewed thoroughly and only the common modeling elements from both extensions should be added to the integrated meta-model.

In the Integration Process of UML class diagram extensions, only one conflict was found. Przybylek [21] and Sharafi et al. [29] both worked on aspect-oriented modeling but

Przybylek extended the UML heavily while Sharafi et al. extended it lightly. Przybylek defined a whole new meta-model that uses UML to reuse elements from its infrastructure and superstructure. Przybylek also defined a whole new package that contains modeling elements for aspect-oriented concepts. In other words, the behavior of UML was altered. This alteration of behavior came from specifying the attributes and semantics of the defined modeling elements.

Figure 4.14 shows the proposed package by Przybylek, called Aspect-oriented UML which imports the Kernel package.

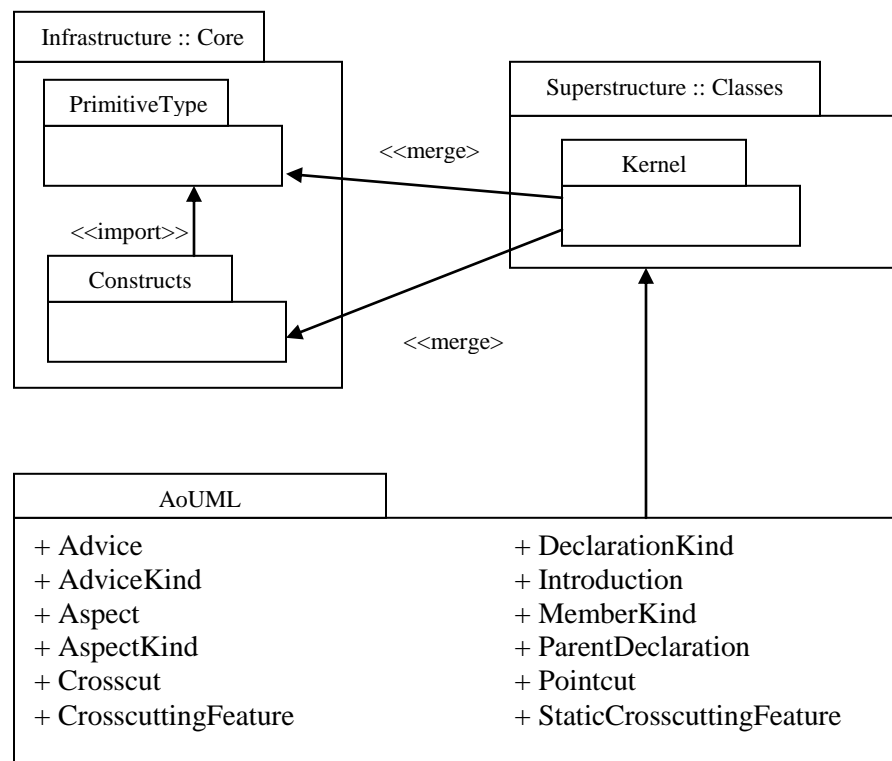


Figure 4.14: Przybylek’s UML heavyweight extension mechanism in [21]

Sharafi et al., on the other hand, simply mapped their domain model to the UML meta-model. Sharafi et al. used UML meta-classes such as Class and Behavioral Feature to

represent their domain model elements, such as; Aspect, Advice, etc. Figure 4.15 shows the proposed methodology by Sharafi et al. where their second main step was a simple mapping procedure of Aspect-oriented constructs into UML profile components.

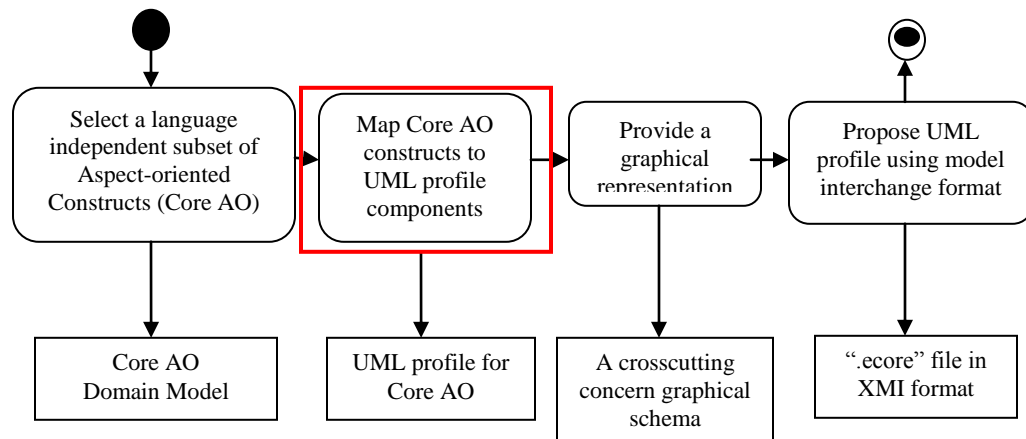


Figure 4.15: UML lightweight extension mechanism by Sharafi et al. in [28]

Both authors worked on the same domain elements but Przybylek specified more attributes and restricted the behavior of the elements on a meta-model that became less similar to the UML meta-model. If Przybylek’s extended the UML lightly, the integration with the work of Sharafi et al. would have been straightforward, since both extensions would have been just a plain procedure of mapping and the differences would have been unmentionables.

Nevertheless, Przybylek’s modeling elements were gathered and only the common ones were acquired to be fitted in the integrated meta-model. The other modeling elements were excluded because they contradict with the intended goal and purpose of the common modeling elements. For example: the modeling element Crosscut was excluded because it alters the behavior of the element: Aspect. The common modeling elements were; Aspect, Advice, Introduction, Point-cut, Abstract, Privileged, Instantiation, Precedence and Parent

Declaration. Figure 4.16 is an excerpt of iUML class diagram meta-model where the red boxes represent the acquired modeling elements proposed by Przybylek. The green tails below the red boxes indicate that there is another extension, Sharafi et al. extension, which shares the same modeling elements proposed by Przybylek.

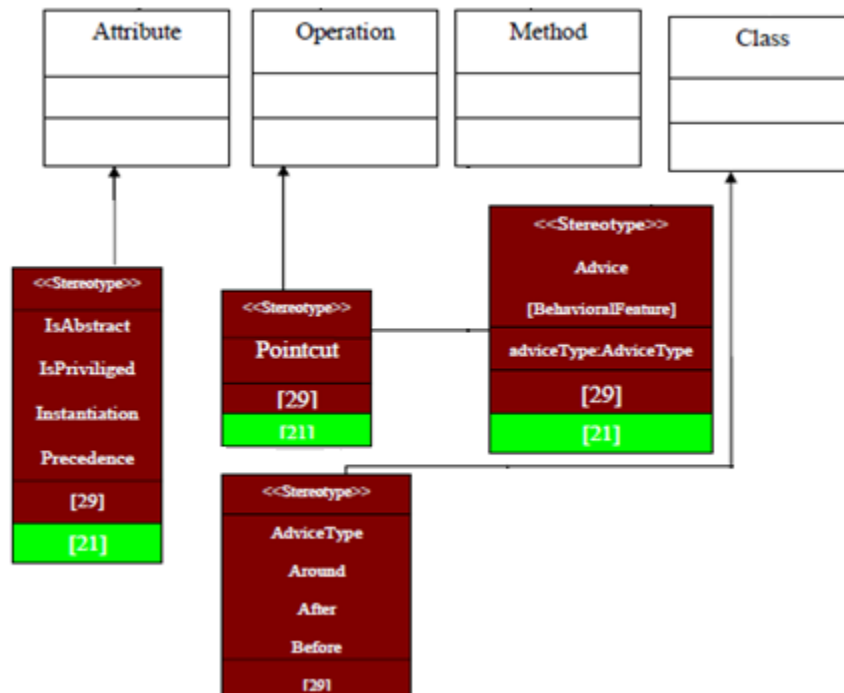



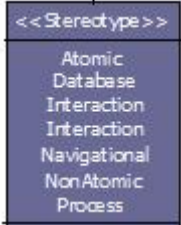





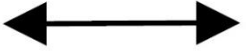



Figure 4.16: Excerpt of iUML class diagram meta-model

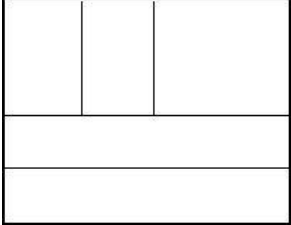
### Consistency between the Class Diagram Graphical Symbols and the Class Diagram Meta-Model



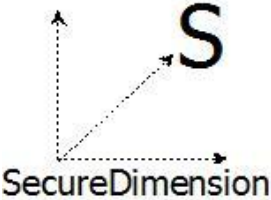
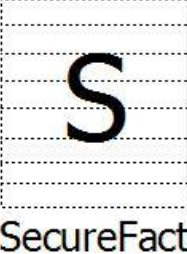
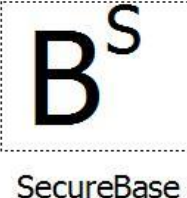
The class diagram graphical symbols found in iUML library are checked for consistency with the iUML meta-model. The goal of the checking process is to make sure that each graphical element reflects an existing meta-model element.

Table 4.9 lists every class diagram graphical symbol found in iUML library and its location in iUML meta-model.

Table 4.9: Mapping iUML class diagram graphical symbols into the meta-model

Modeling element	Source	Location in iUML meta-model
 Atomic_Class	Jantan et al. 2008 [27]	
 Database_Class	Jantan et al. 2008 [27]	
 Interaction Class	Jantan et al. 2008 [27]	
 Navigational Class	Jantan et al. 2008 [27]	
 NonAtomic_Class	Jantan et al. 2008 [27]	
 Process_Class	Jantan et al. 2008 [27]	
 Database Link	Jantan et al. 2008 [27]	
	Fontoura et al. 2000 [22]	
	Fontoura et al. 2000 [22]	
	Fontoura et al. 2000 [22]	

<div style="border: 1px dashed black; padding: 5px; display: inline-block;">{static}</div>	Fontoura et al. 2000 [22]	<div style="border: 1px solid black; padding: 5px; background-color: #4a5568; color: white;">       &lt;&lt; TaggedValue &gt;&gt;        Appl-Class        Variable        Extensible        Static        Dynamic        Incomplete        ForAllNewMethods        Optional        Final        Variation     </div>
<div style="border: 1px dashed black; padding: 5px; display: inline-block;">{dynamic}</div>	Fontoura et al. 2000 [22]	
<div style="border: 1px dashed black; padding: 5px; display: inline-block;">{incomplete}</div>	Fontoura et al. 2000 [22]	
<div style="border: 1px dashed black; padding: 5px; display: inline-block;">{forAllNewMethods}</div>	Fontoura et al. 2000 [22]	
<div style="border: 1px dashed black; padding: 5px; display: inline-block;">{optional}</div>	Fontoura et al. 2000 [22]	
<div style="border: 1px dashed black; padding: 5px; display: inline-block;">{final}</div>	Sanada and Adams 2002 [25, 26]	
	Byeon et al. 2004 [23]	<div style="border: 1px solid black; padding: 5px; background-color: #4a5568; color: white;">       &lt;&lt; Stereotype &gt;&gt;        SymbolRep        IconNotation     </div>
<div style="border: 1px solid black; padding: 5px;">       &lt;&lt;requirements&gt;&gt;  <hr/> <hr/> <hr/> </div>	Mahmood and Lai 2009 [28]	<div style="border: 1px solid black; padding: 5px; background-color: #4a5568; color: white;">       &lt;&lt; Stereotype &gt;&gt;        RClass        [Class]        AbstractionLevel        Goal        Scenario        Rank     </div>
<div style="border: 1px solid black; padding: 5px;">       &lt;&lt;component&gt;&gt;  <hr/> <hr/> </div>	Mahmood and Lai 2009 [28]	<div style="border: 1px solid black; padding: 5px; background-color: #4a5568; color: white;">       &lt;&lt; Stereotype &gt;&gt;        CClass        [Class]        Name        Features        ContextDependency     </div>

	Fernandez-medina et al.2007 [12]	
	Fernandez-medina et al.2007 [12]	
	Fernandez-medina et al.2007 [12]	
	Fernandez-medina et al.2007 [12]	

### Results of integrating meta-model concepts

The final integrated meta-model is shown in Figure 4.17 to Figure 4.19. The white boxes are the original elements of the class diagram meta-model [10, 12, 13] and the colored boxes are the UML extensions.

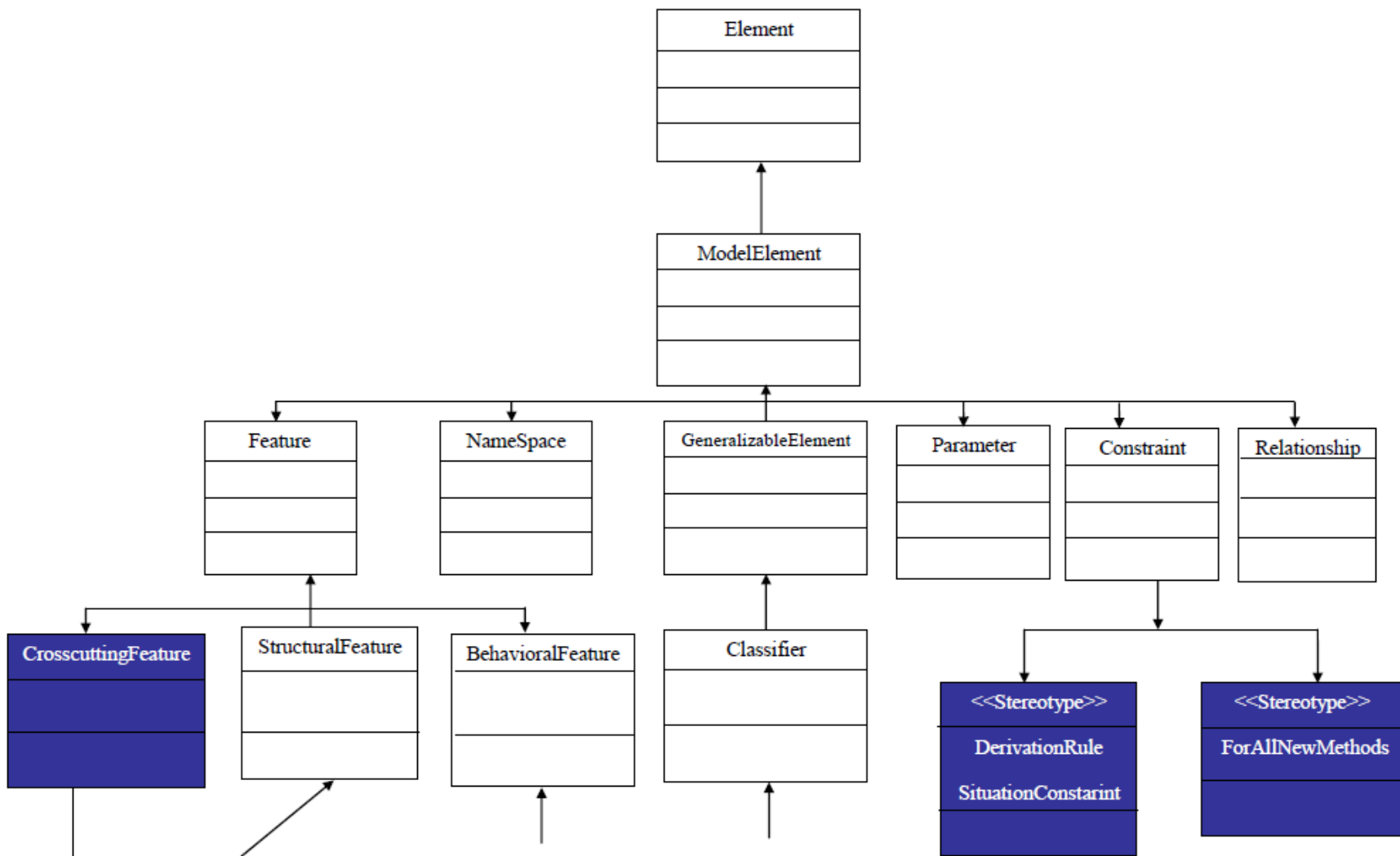


Figure 4.17: First part of iUML class diagram meta-model



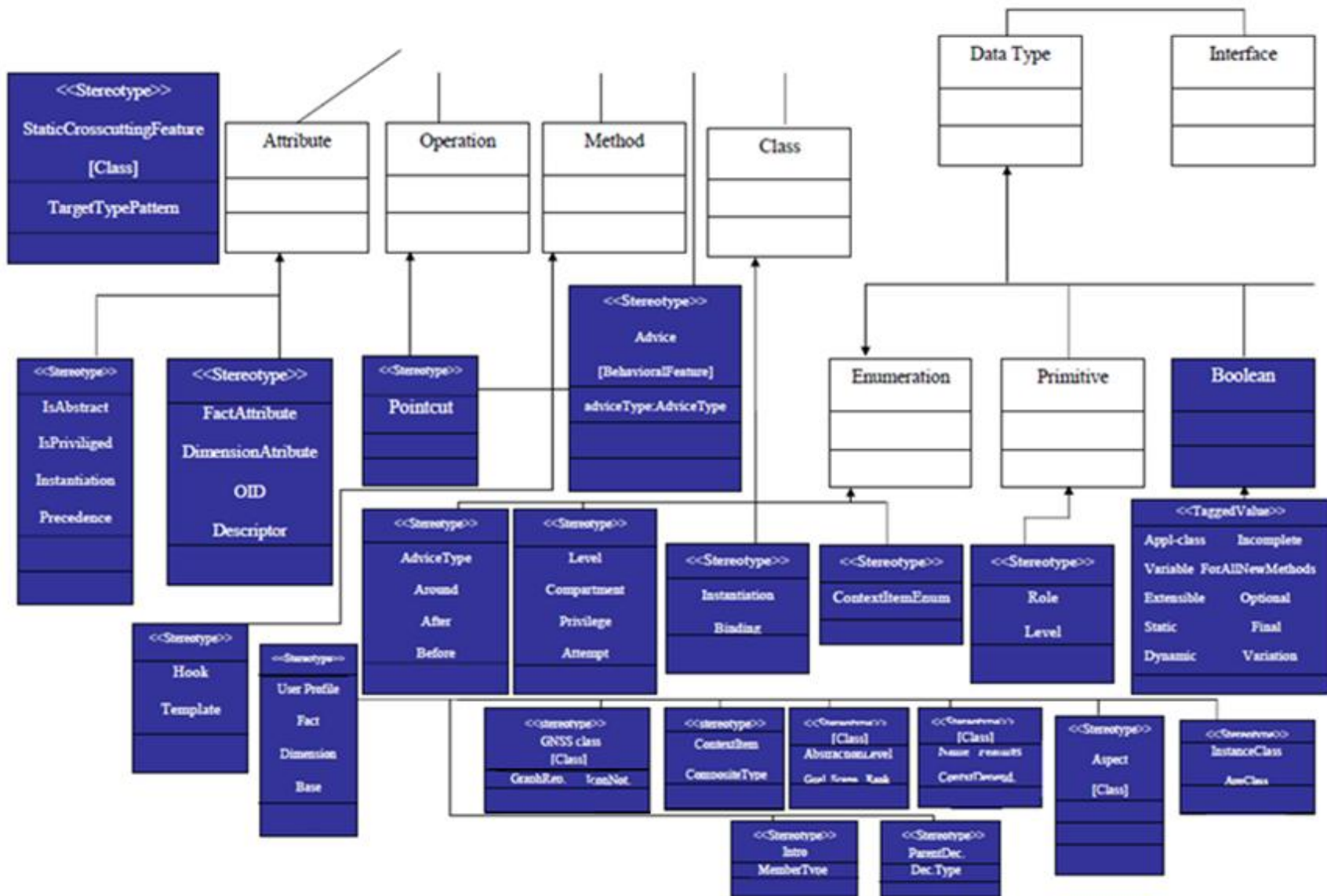


Figure 4.18: Second part of iUML class diagram meta-model

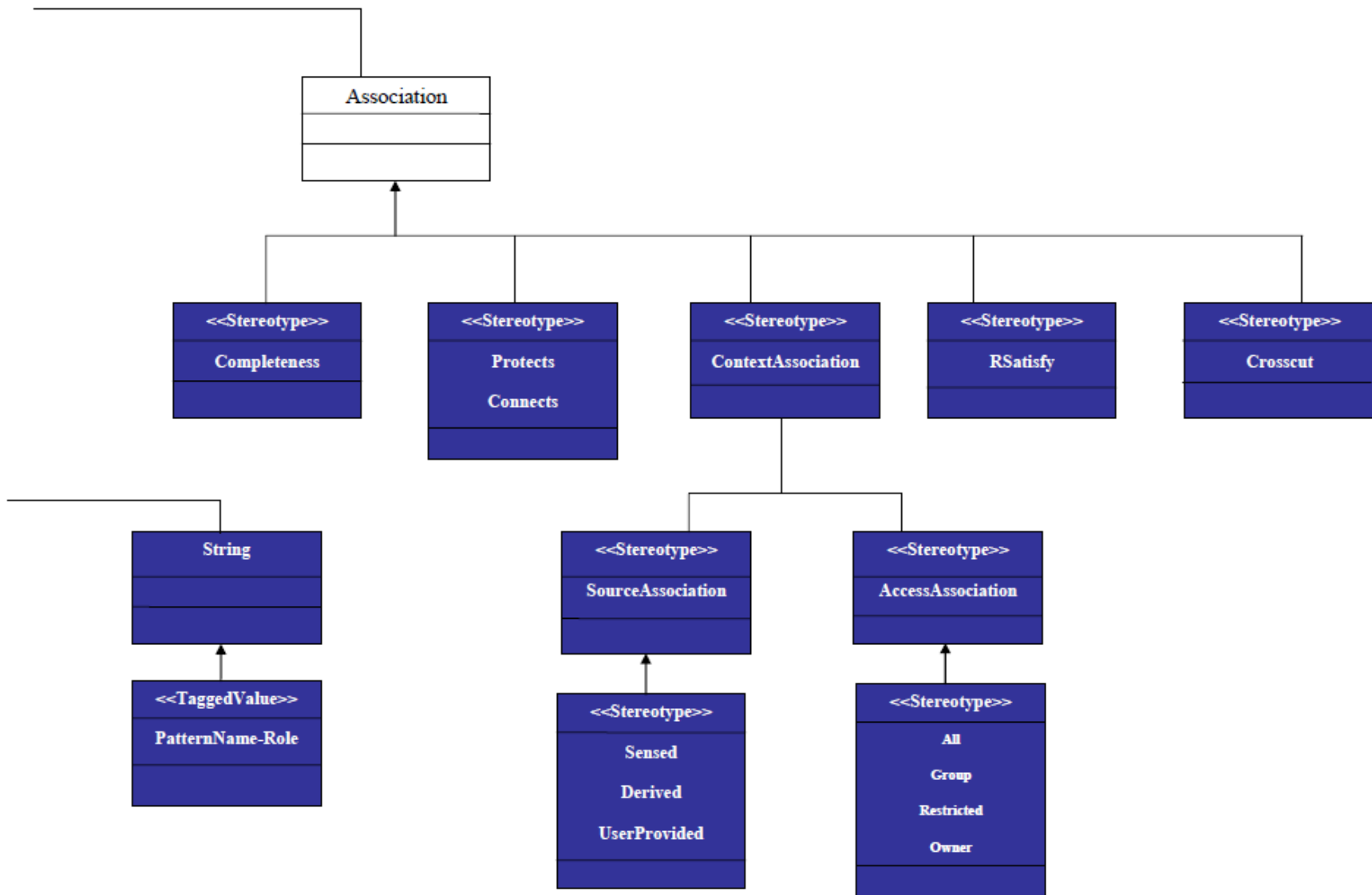


Figure 4.19: Third part of iUML class diagram meta-model

#### 4.2.2.2 Sequence diagram

The reviewed literature contains 23 extensions; 3 of which are applied to UML sequence diagram. Table 4.10 shows the modeling elements accompanied with some constraints from the sequence diagram extensions. The main objective of this table is to show constituting elements of the integrated sequence diagram along with their specified constraints.

Table 4.10: The modeling elements of UML sequence diagram extensions

Modeling Element	Extended from (Meta Class)	Use of the Modeling Element	Associated Constraints
REservice [32]	Classifier	Represents a sequence of actions and interactions.	A REservice can be requested by a REuser or from a REcomponent.
REcomponents [32]	Classifier	Represents the main interacting objects.	REhost hosts a set of REcomponents. Each REcomponent has a structure of possibly other REcomponents.
REconnectors [32]	AssociationRole	Represents the means in which the REcomponents interact through.	Each REconnector links REcomponents.
REuser [32]	Classifier	Represents the party that triggers the actions.	One REuser requires many REservices and one REservice is requested by many REusers.
REhost [32]	Classifier	Represents the hosting party of REcomponents.	REhost hosts a set of REcomponents. Each REconnector links REcomponents.

#### Step 1: Adding the Elements

The first step is the addition of newly introduced modeling elements under the appropriate classifier in the meta-model. Figure 4.20 represent the original elements of UML and the

colored ones represent extensions. Due to its large size, the meta-model will be divided into three parts to show the addition of extensions' modeling elements. The first part of the meta-model is shown in Figure 4.20. The second part is shown in Figure 4.21.

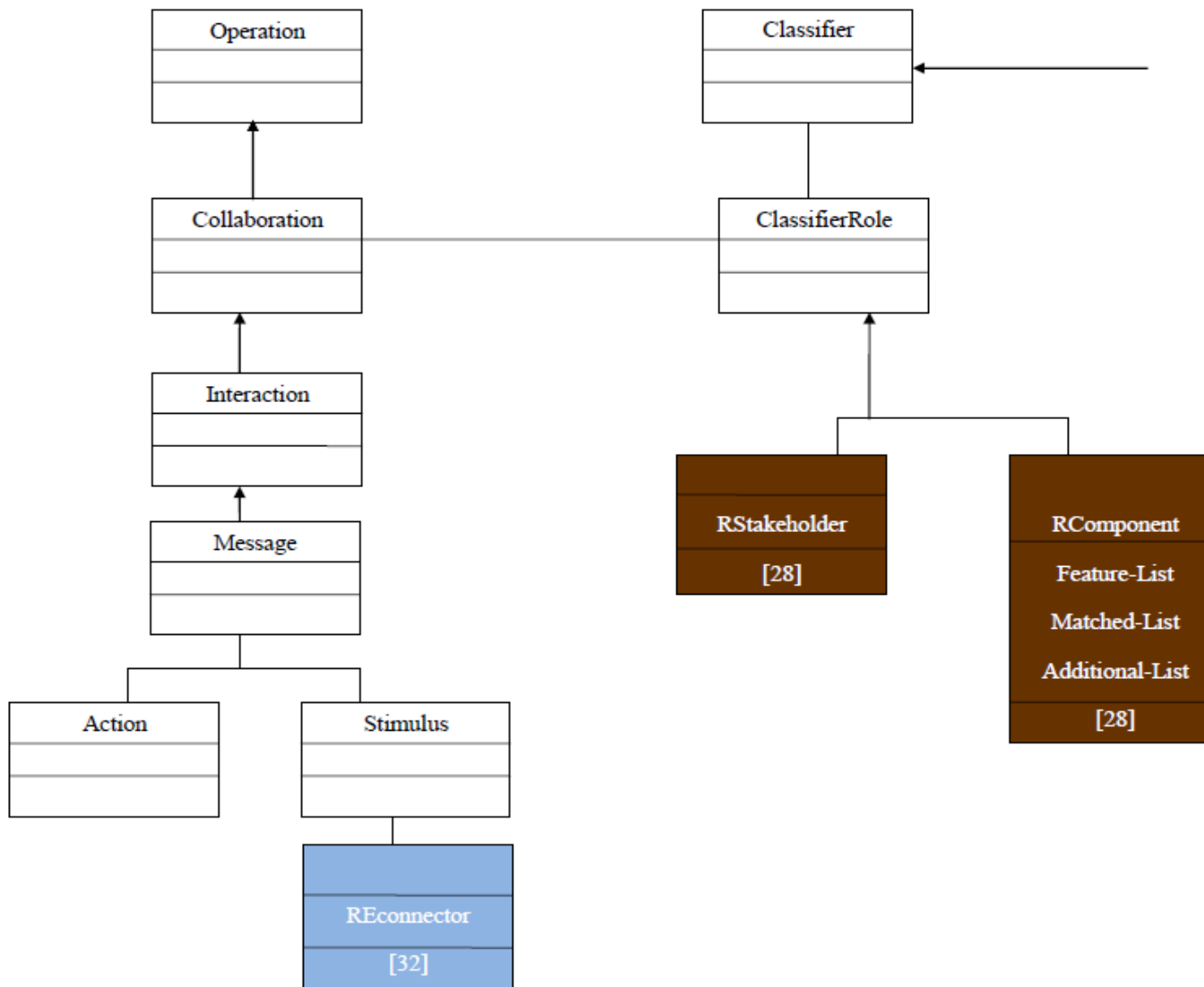


Figure 4.20: First part of original UML sequence diagram meta-model elements and integrated elements

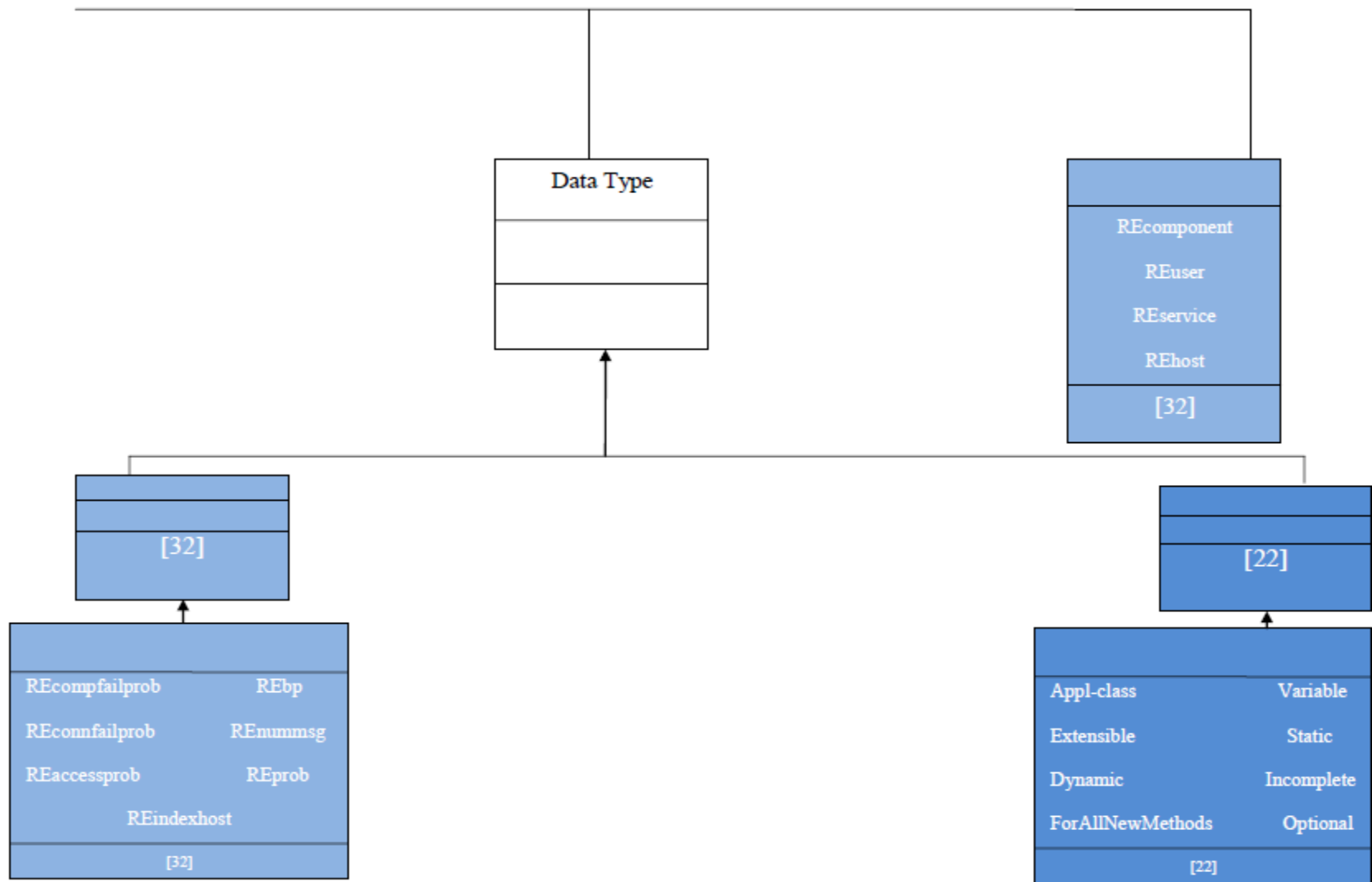


Figure 4.21: Second part of original UML sequence diagram meta-model elements and integrated elements

## **Step 2: Categorizing the Elements**

Next is categorizing elements as <<Stereotype>> or <<TaggedValue>>. Figure 4.22 shows the two types of meta-model concepts; stereotype and tagged value. Due to the large size of the meta-model, categorizing the extensions' modeling elements will be done in two parts. The first part of the meta-model is shown in Figure 4.22. The second part is shown in Figure 4.23.

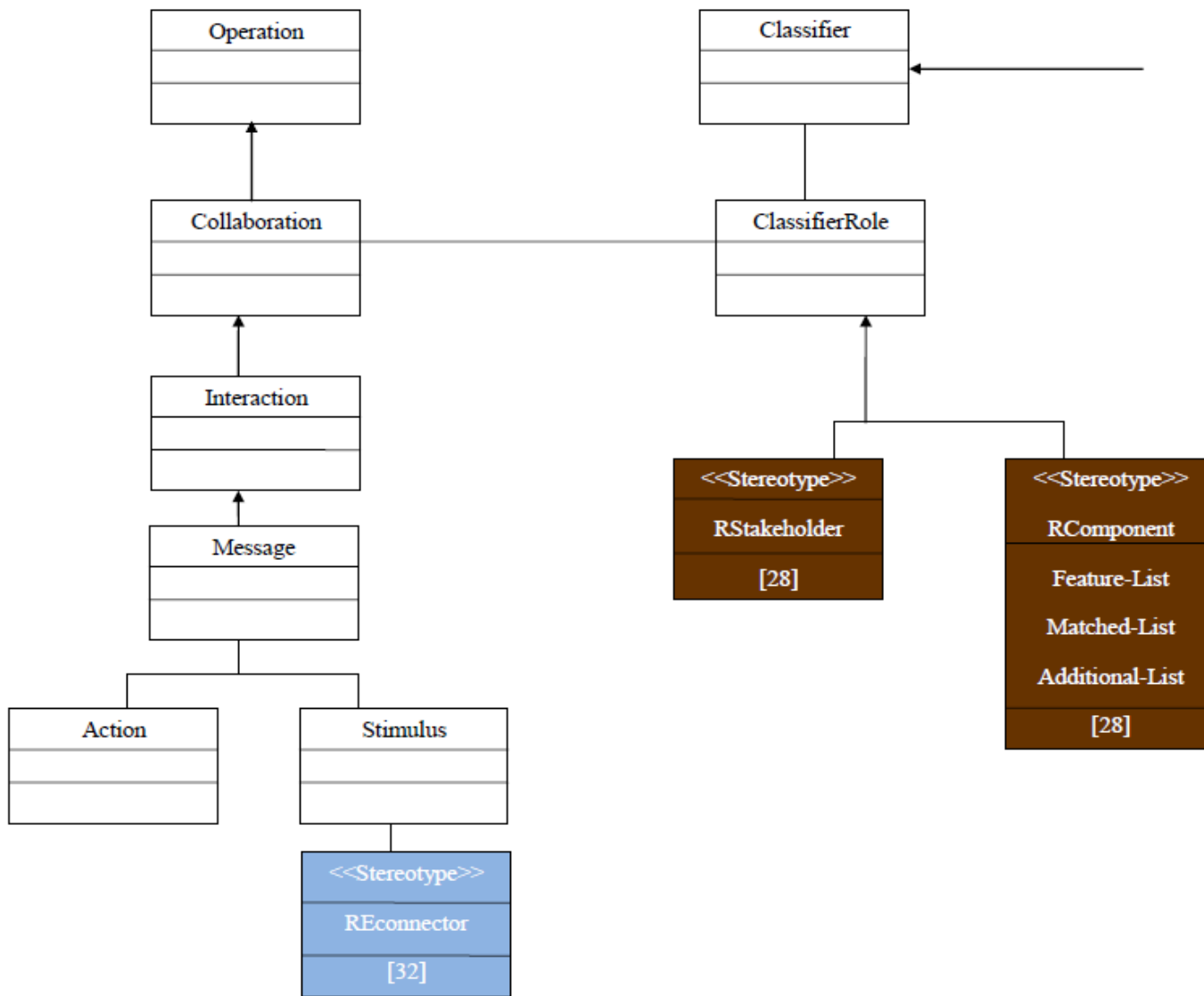


Figure 4.22: Categorizing first part of elements as Stereotypes and Tagged Values



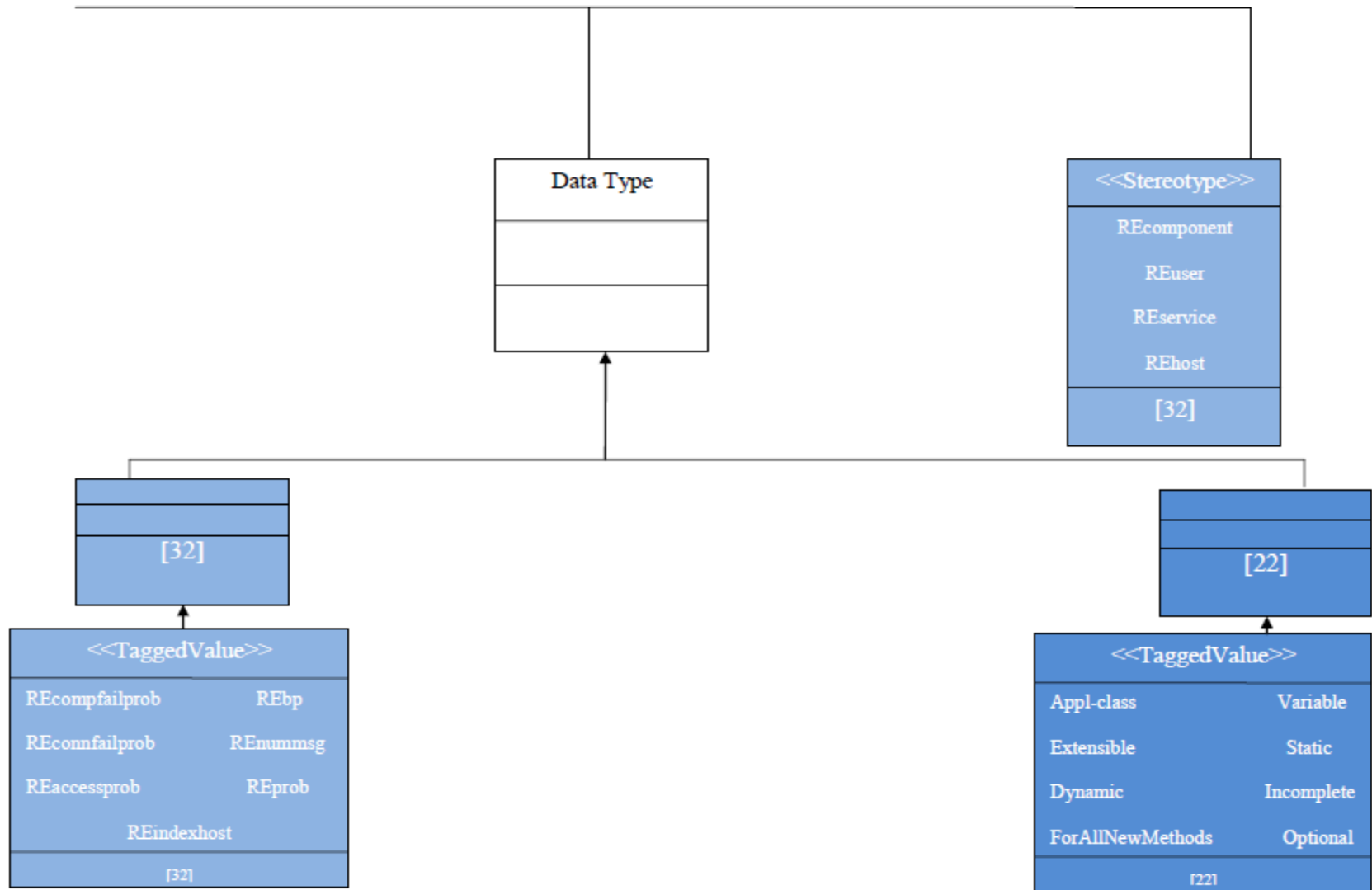


Figure 4.23: Categorizing second part of elements as Stereotypes and Tagged Values

### Step 3: Defining Meta-classes or other classifiers

As for meta-classes, no meta-classes were found in the reviewed literature to be integrated into the meta-model. In Figure 4.24, there is an example that shows an introduction of a classifier by Fontoura et al. [22]. This classifier is Boolean which derives the tag definitions beneath it. The Boolean classifier is considered an original element in the UML meta-model but Fontoura et al. emphasized it and showed it to be able to use the proposed tag definitions in their model.

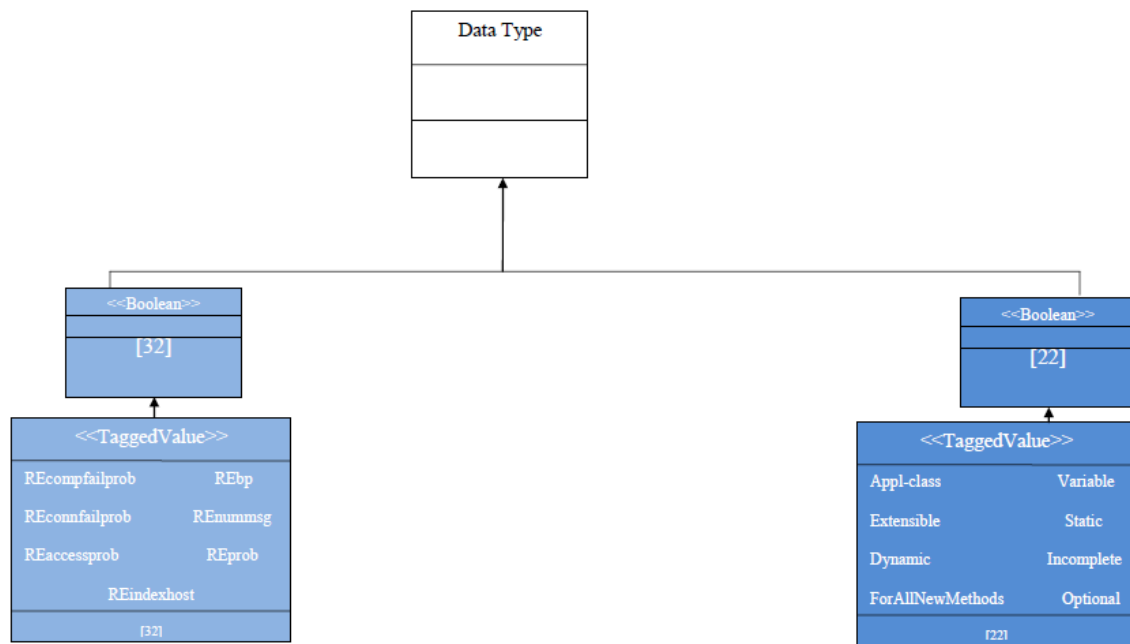


Figure 4.24: Boolean type classifier

### Step 4: Case A: Combination

In the process of integrating UML sequence diagram extensions, no extensions shared the exact same domain; therefore, the extensions were separately integrated in the meta-model.


### Step 5: Case A: Conflict

No conflicts were found as no attempts of combining extensions of the same domain happened.

### Consistency between the Sequence Diagram Graphical Symbols and the Sequence Diagram Meta-Model

The sequence diagram graphical symbols found in iUML library are checked for consistency in iUML meta-model. Table 4.11 lists every sequence diagram graphical symbol found in iUML library and its location in iUML meta-model.

Table 4.11: Mapping iUML sequence diagram graphical symbols into the meta-model

Modeling element	Source	Location in iUML meta-model
{variable}	Fontoura et al. 2000 [22]	
{appl-class}	Fontoura et al. 2000 [22]	
{extensible}	Fontoura et al. 2000 [22]	
{static}	Fontoura et al. 2000 [22]	
{dynamic}	Fontoura et al. 2000 [22]	
{incomplete}	Fontoura et al. 2000 [22]	
{forAllNewMethods}	Fontoura et al. 2000 [22]	
{optional}	Fontoura et al. 2000 [22]	
{final}	Sanada and Adams 2002 [25, 26]	

## **Results of integrating meta-model concepts**

Figure 4.25 and Figure 4.26 show the meta-model for the UML sequence diagram extensions. The white boxes are the original elements of the sequence diagram meta-model [14, 15] and the colored boxes are the UML extensions.

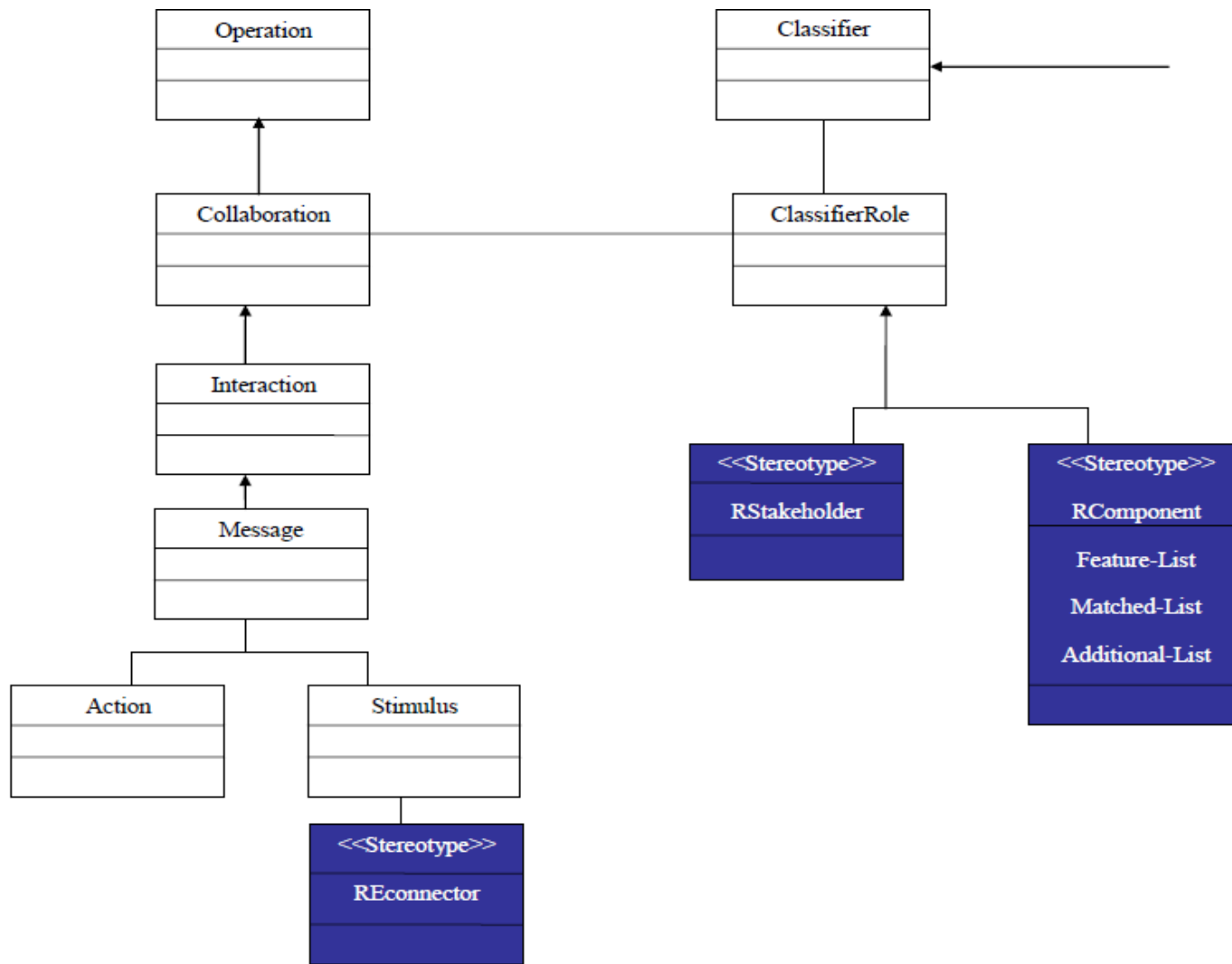


Figure 4.25: First part of iUML sequence diagram meta-model

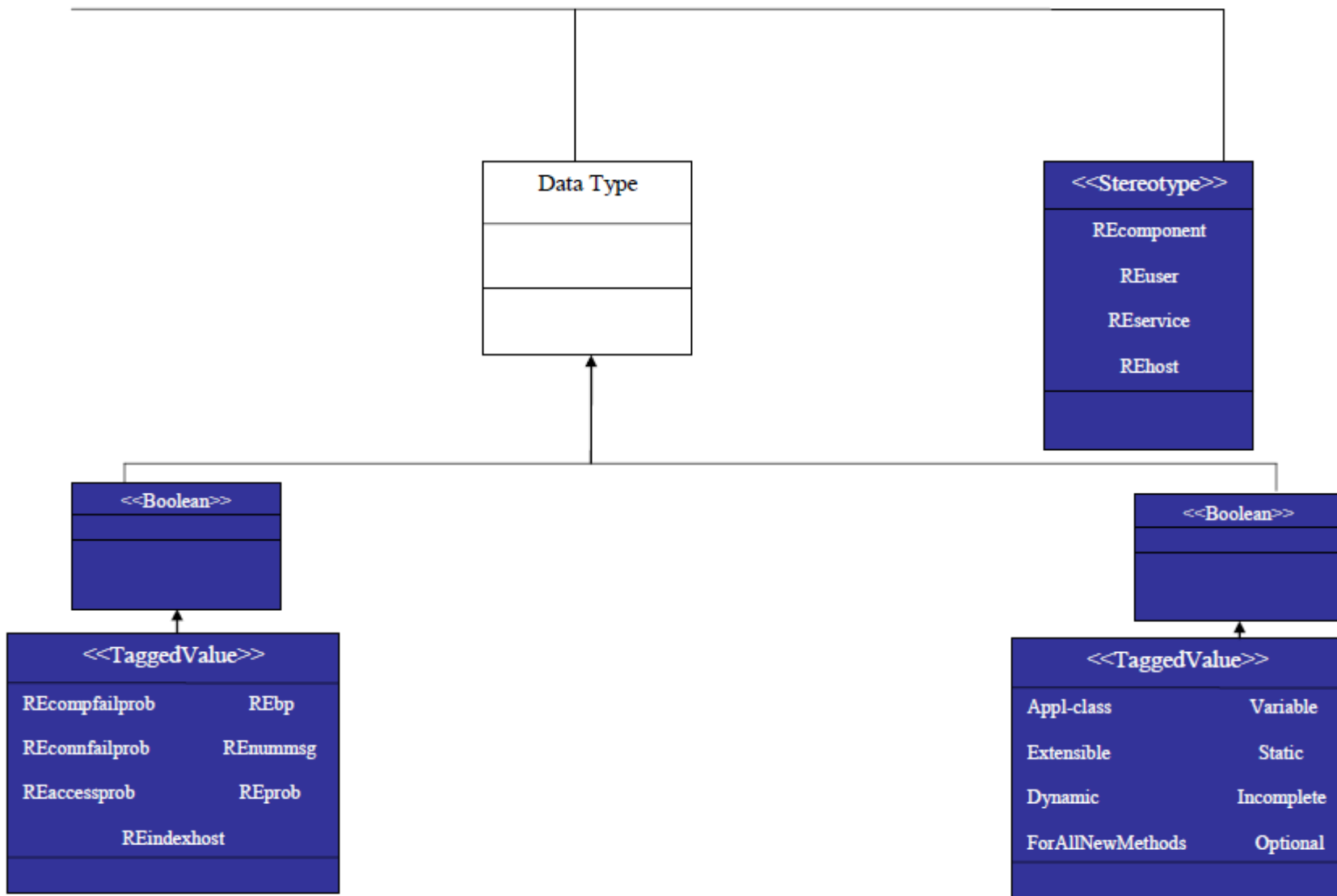


Figure 4.26: Second part of iUML sequence diagram meta-model

### 4.2.2.3 Use case diagram

The reviewed literature contains 23 extensions; 3 of which are applied to UML use case diagram. Table 4.12 shows the modeling elements accompanied with some constraints from the use case diagram extensions. The main objective of this table is to show constituting elements of the integrated use case diagram along with their specified constraints.

Table 4.12: The modeling elements of UML use case diagram extensions

Modeling Element	Extended from (Meta Class)	Use of the Modeling Element	Associated Constraints
Agent [16]	AgentClassifier	Provides a way to classify agents.	Responsible for classifying agents.
	AgentPhysicalClassifier	Provides features for the agents.	Responsible for providing agents with features.
	AgentRoleClassifier	Provides roles for the agents.	
PhysicalActor [17]	ActorClassifier	Represents a human user who visits the Web application.	Visits the Web Application.
LogicalActor [17]		Represents a role played by a human user to assure the maintenance of Web application.	Maintains the Web Application.
SystemActor [17]		Represents a computer system, device hardware or Web service, etc.	
UseCaseSIF [17]	Meta-scenario SIF	Displays static Web Page.	

UseCaseDIF [17]	Meta-scenario DIF	Displays dynamic Web Page.	
UseCasePF [17]	Meta-scenario PF	Displays dynamic Web Page using UPDATE request.	
REservice [32]	Classifier	Represents a sequence of actions and interactions.	A REservice can be requested by a REuser or from a REcomponent.
REcomponents [32]	Classifier	Represents the main interacting objects.	REhost hosts a set of REcomponents. Each REcomponent has a structure of possibly other REcomponents.
REconnectors [32]	AssociationRole	Represents the means in which the REcomponents interact through.	Each REconnector links REcomponents.
REuser [32]	Classifier	Represents the party that triggers the actions.	One REuser requires many REServices and one REService is requested by many REusers.
REhost [32]	Classifier	Represents the hosting party of REcomponents.	REhost hosts a set of REcomponents. Each REconnector links REcomponents.

### Step 1: Adding the Elements

The first step is to insert the newly introduced modeling elements under the appropriate classifier in the meta-model. Figure 4.27 represents the original elements of UML and the colored ones represent extensions. Due to its large size, the meta-model will be divided into three parts to show the addition of extensions' modeling elements. The first part of the meta-model is shown in Figure 4.27. The second part is shown in Figure 4.28 and the third part is shown in Figure 4.29 .



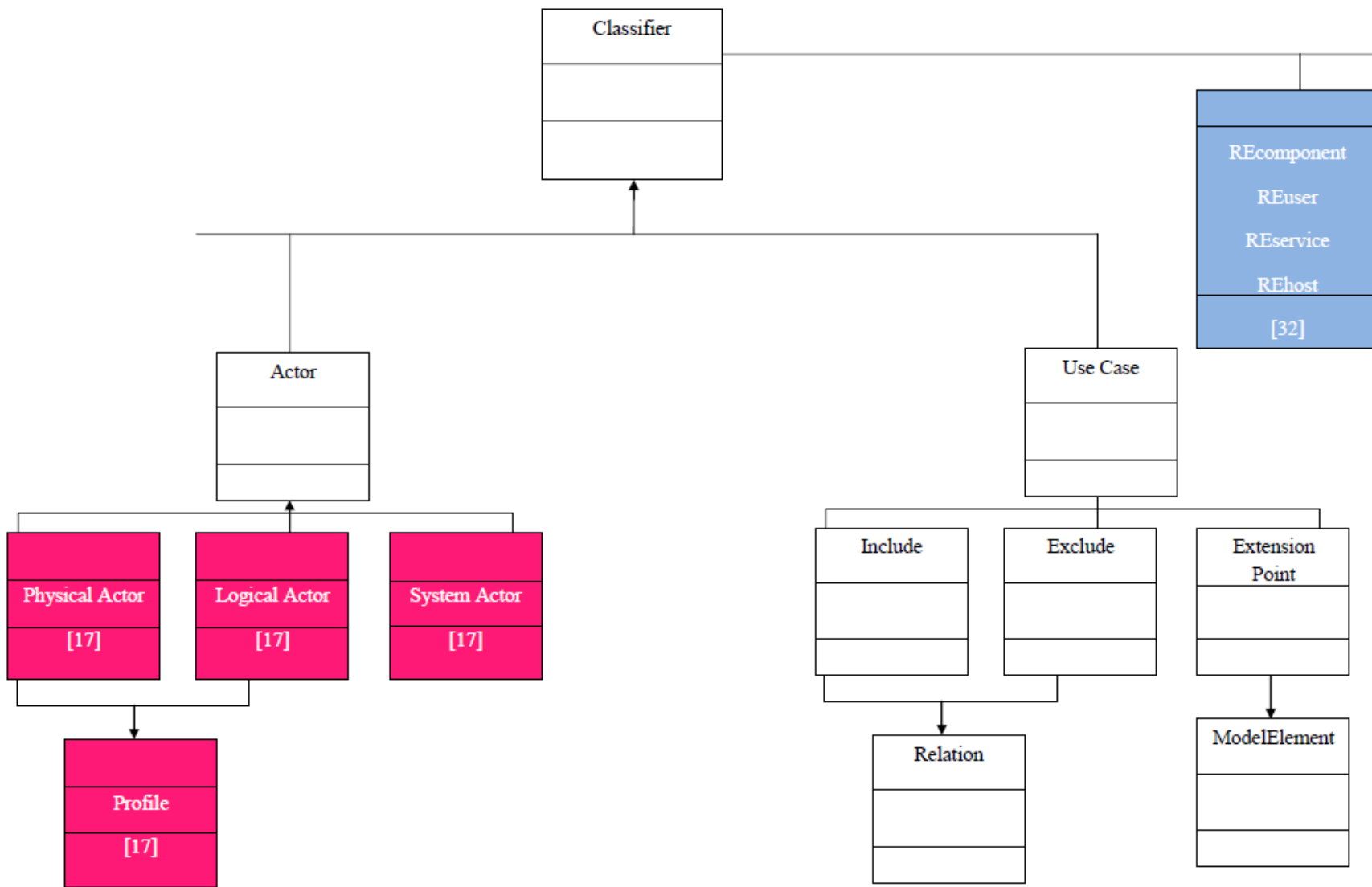


Figure 4.27: First part of original UML use case diagram meta-model elements and integrated elements

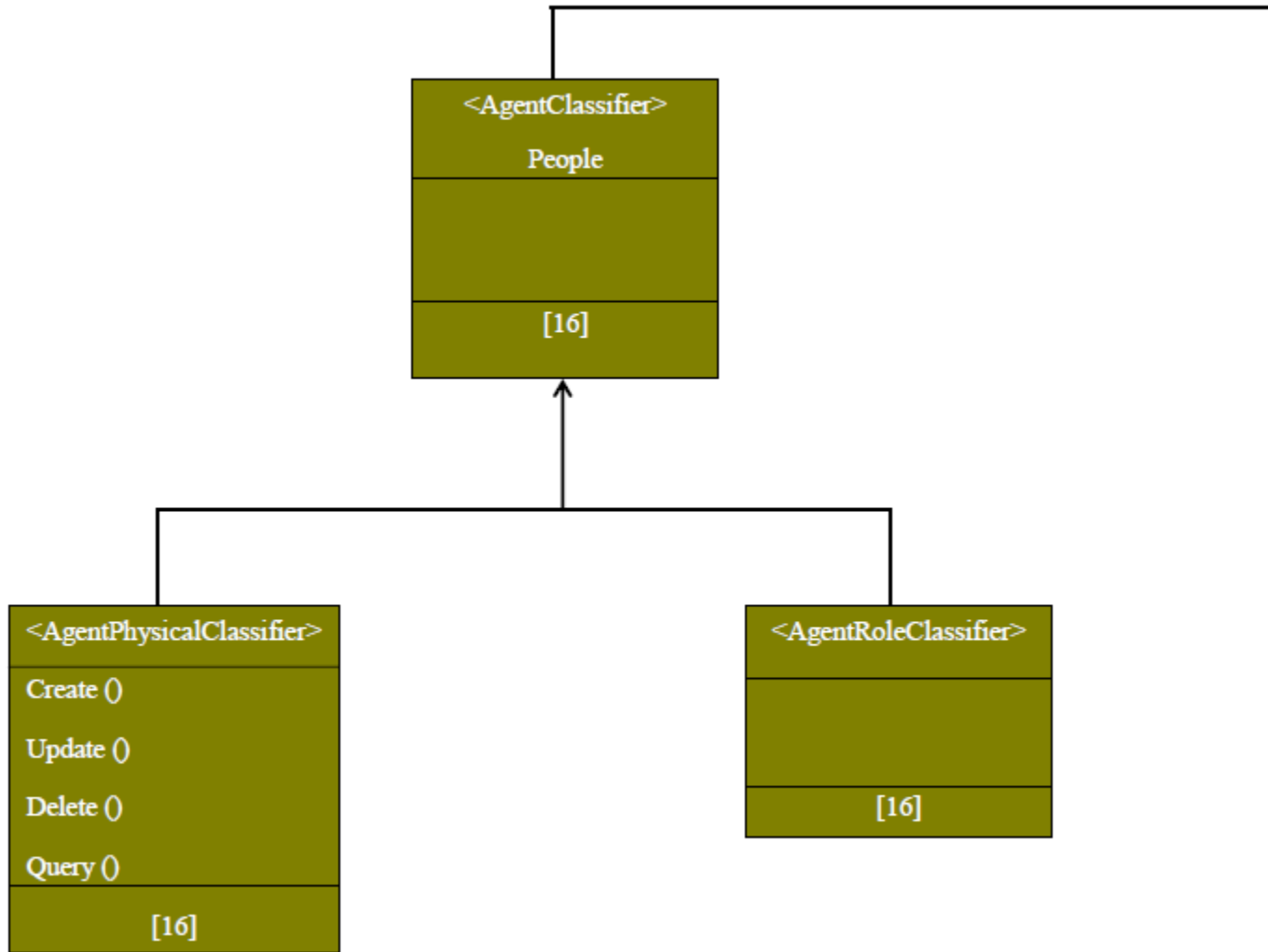


Figure 4.28: Second part of original UML use case diagram meta-model elements and integrated elements

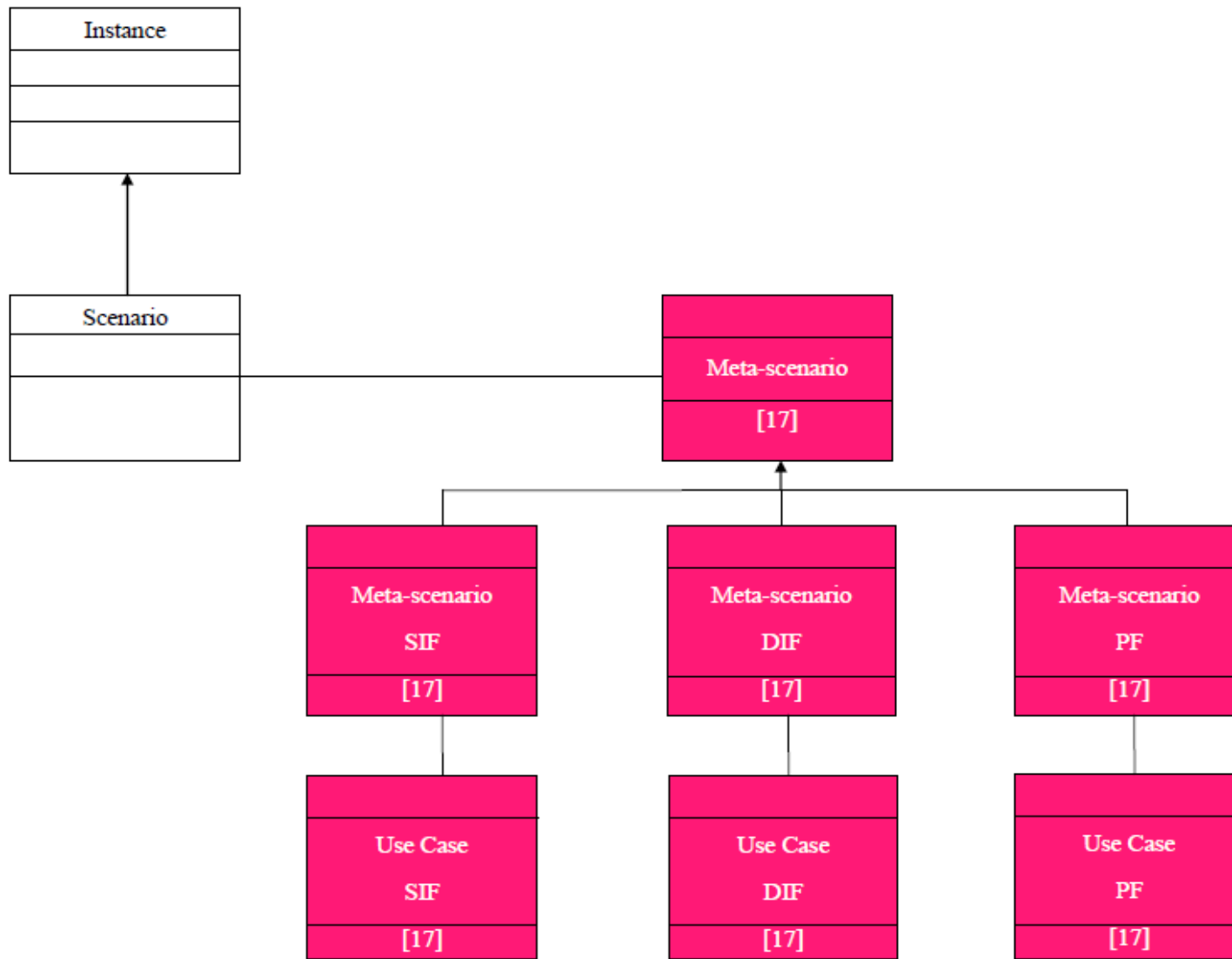


Figure 4.29: Third part of original UML use case diagram meta-model elements and integrated elements

## **Step 2: Categorizing the Elements**

Next is categorizing elements as <<Stereotype>>. No Tagged Value elements were found in the literature. Figure 4.30 shows the stereotype category. Due to the large size of the meta-model, categorizing the extensions' modeling elements will be done in three parts. The first part of the meta-model is shown in Figure 4.30. The second part is shown in Figure 4.31 and the third part is shown in Figure 4.32.

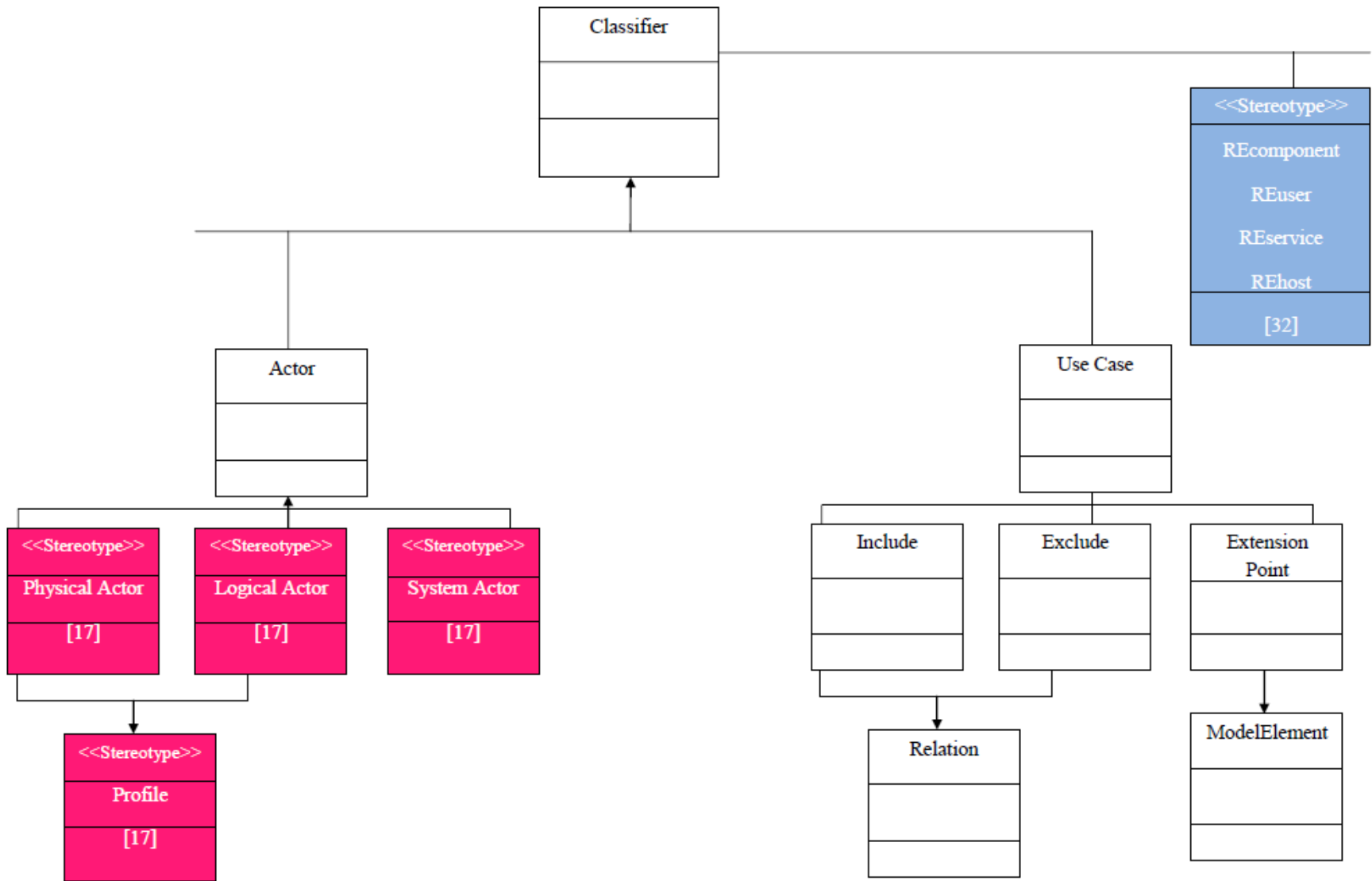


Figure 4.30: Stereotype category of the first part of elements

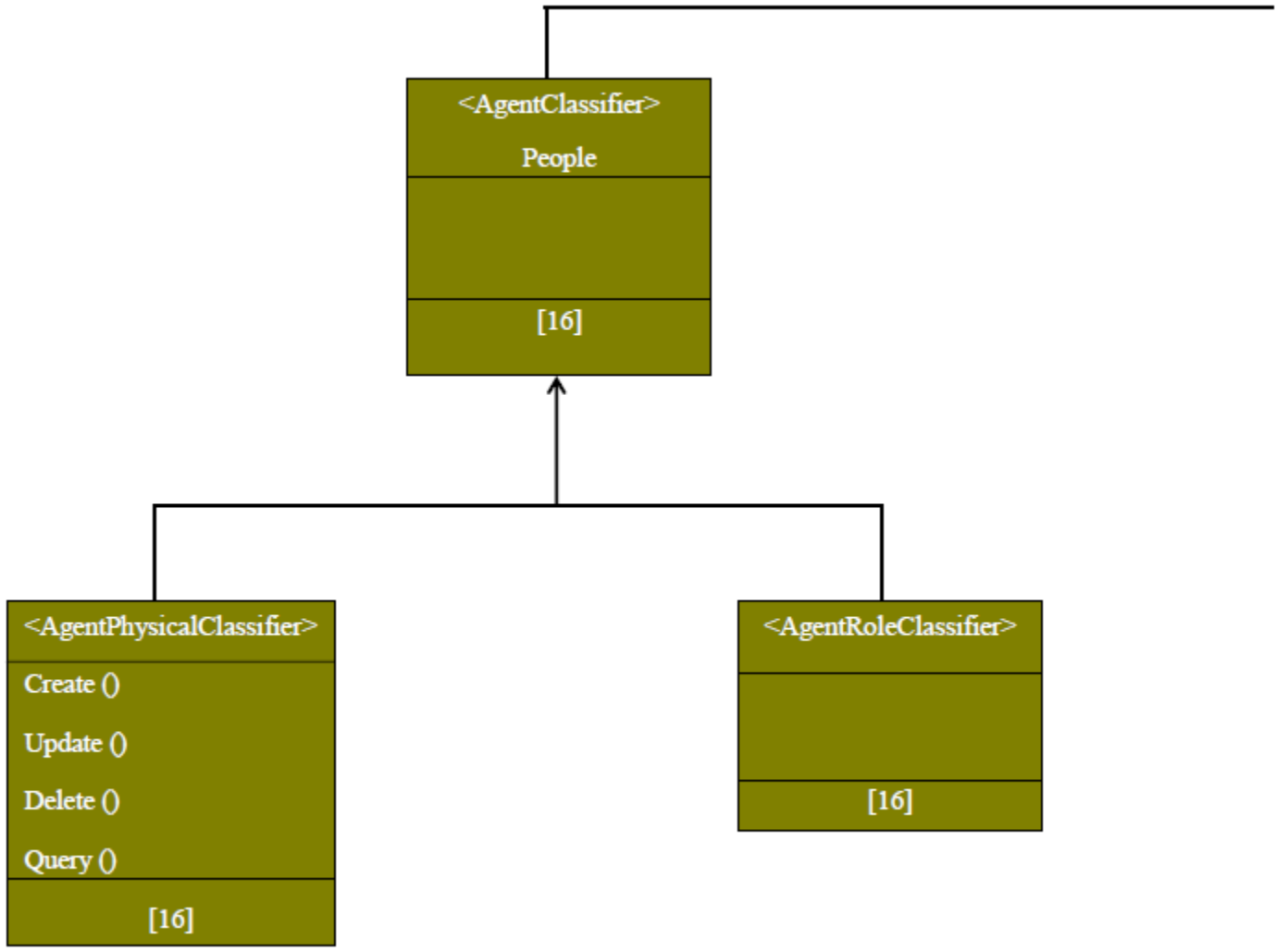


Figure 4.31: Stereotype category of the second part of elements

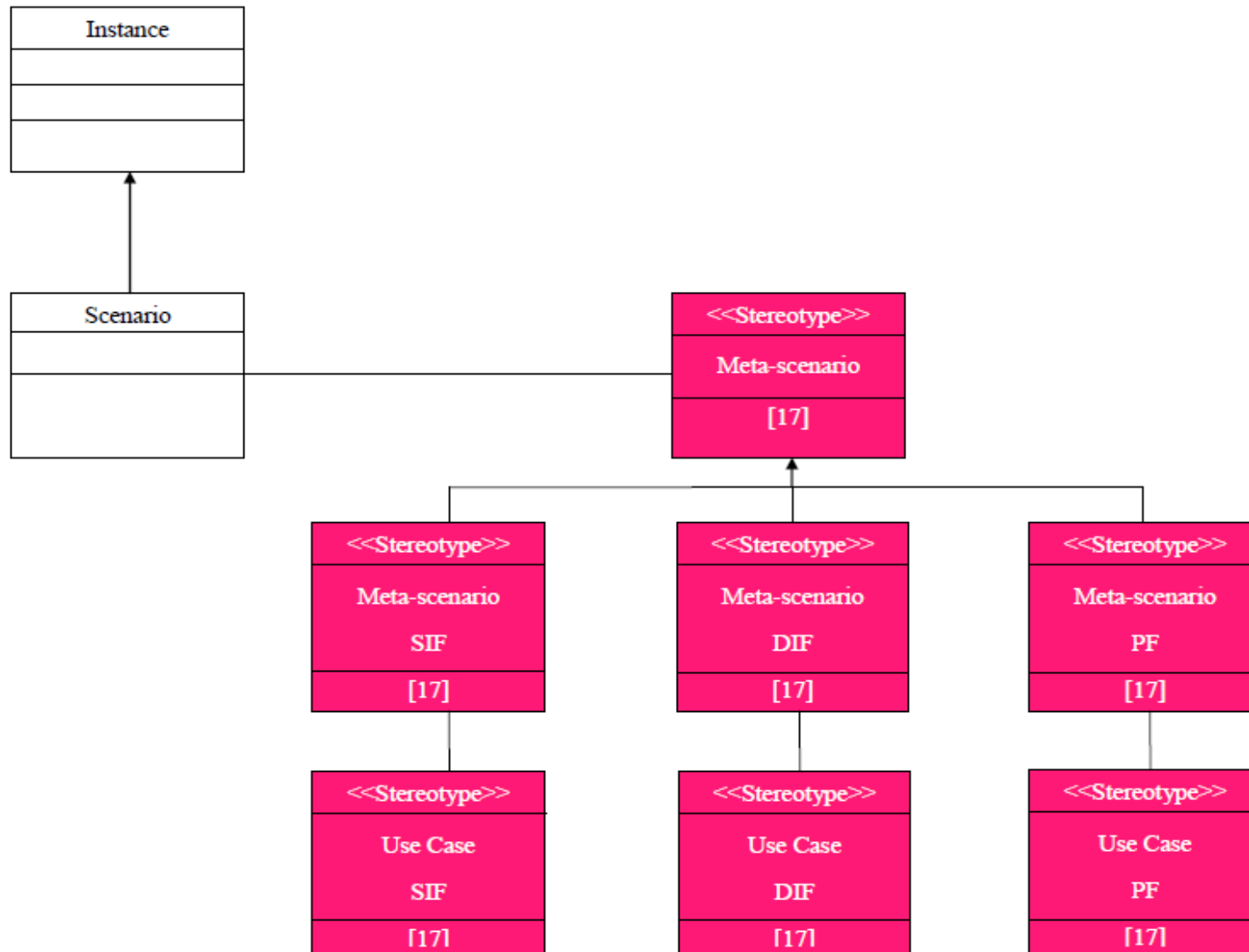


Figure 4.32: Stereotype category of the third part of elements

### Step 3: Defining Meta-classes or other classifiers

No meta-classes or other classifiers were also found in the reviewed literature to be integrated into the meta-model.

### Step 4: Case A: Combination

In the process of integrating UML use case diagram extensions, no extensions shared the exact same domain. The extensions were separately integrated in the meta-model.

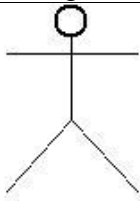

### Step 5: Case A: Conflict

No conflicts occurred since no attempts of combining extensions of the same domain happened.

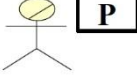

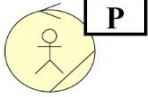
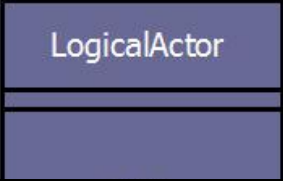

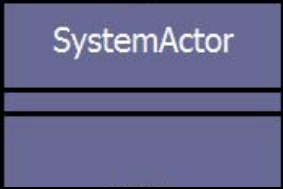



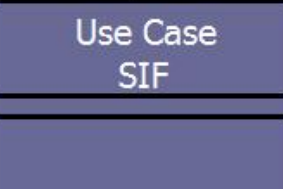

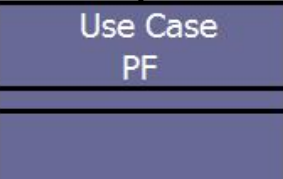
### Consistency between the Use Case Diagram Graphical Symbols and the Use Case Diagram Meta-Model

The use case diagram graphical symbols found in iUML library are checked for consistency in iUML meta-model. Table 4.13 lists every use case diagram graphical symbol found in iUML library and its location in iUML meta-model.

Table 4.13: Mapping iUML use case diagram graphical symbols into the meta-model

Modeling element	Source.	Location in iUML meta-model
 <Agent>	Fei and Yan 2008 [16]	



 <p>physical actor</p>	Djemaa et al. 2006 [17]	
 <p>Logical actor</p>	Djemaa et al. 2006 [17]	
 <p>System actor</p>	Djemaa et al. 2006 [17]	
	Djemaa et al. 2006 [17]	
	Djemaa et al. 2006 [17]	
	Djemaa et al. 2006 [17]	

### Results of integrating meta-model concepts

Figure 4.33 through Figure 4.35 show the meta-model for the UML use case diagram extensions. The white boxes are the original elements of the use case diagram meta-model [16, 17] and the colored boxes are the UML extensions.

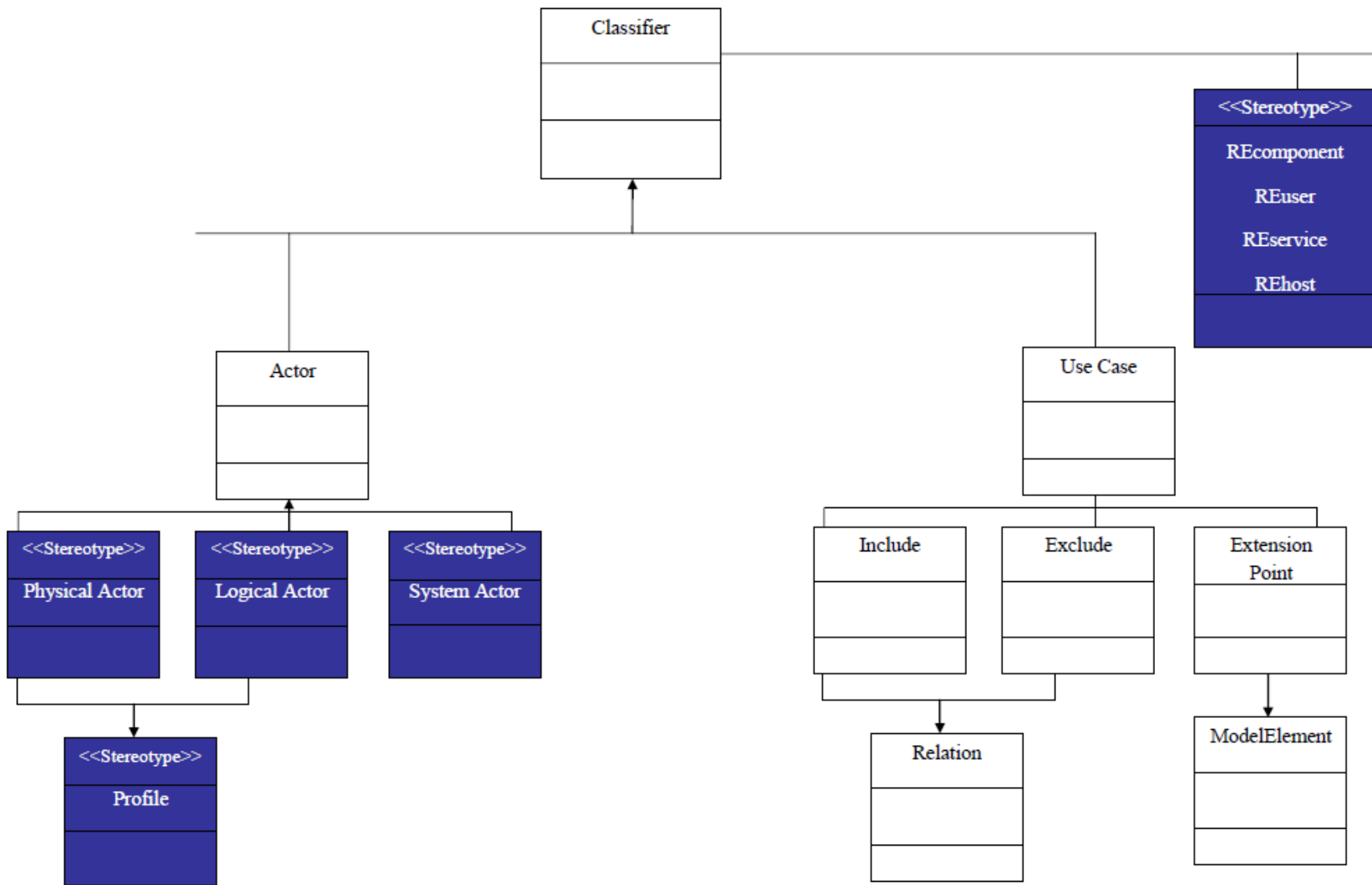


Figure 4.33: First part of iUML use case diagram meta-model

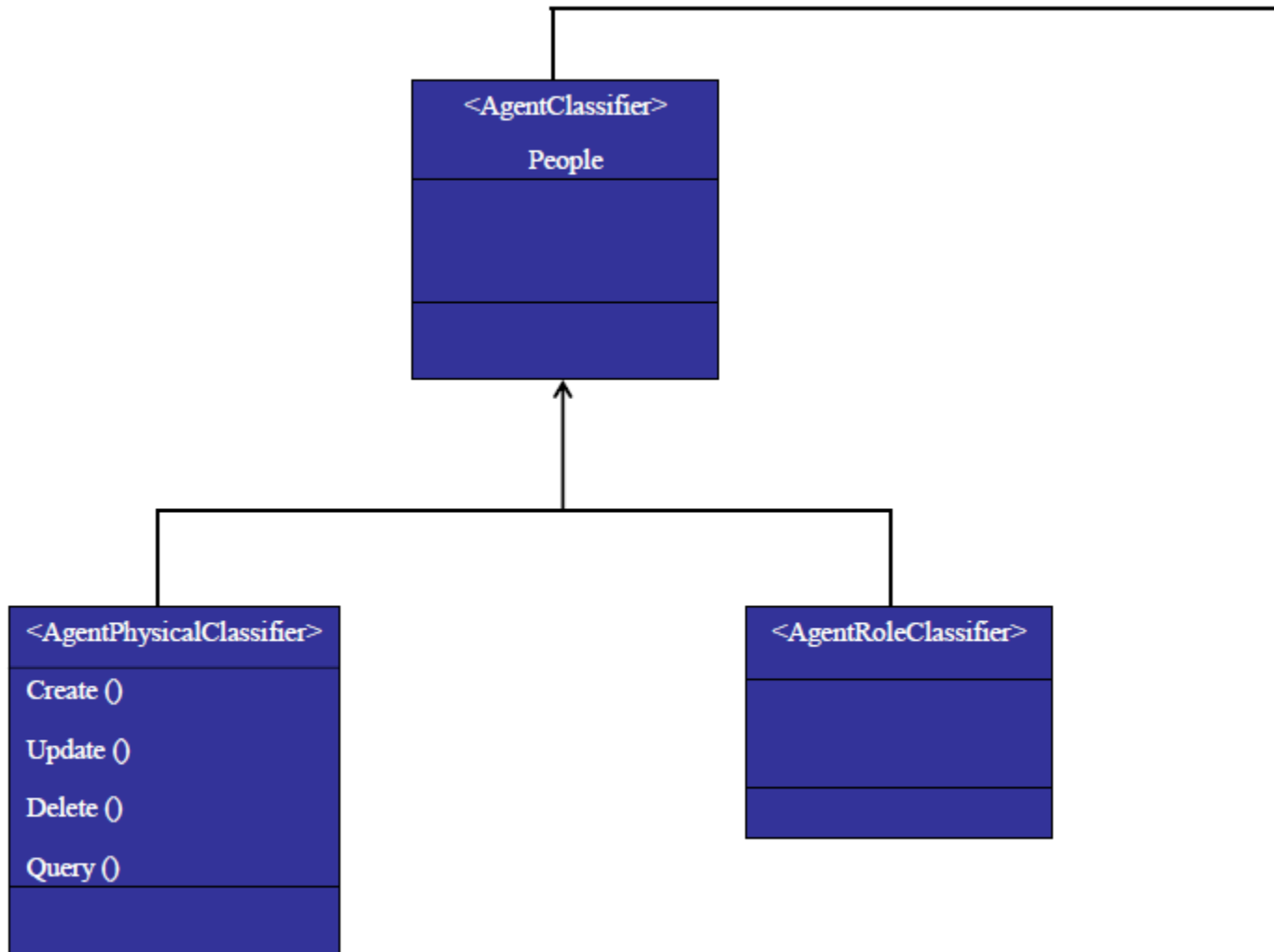


Figure 4.34: Second part of iUML use case diagram meta-model

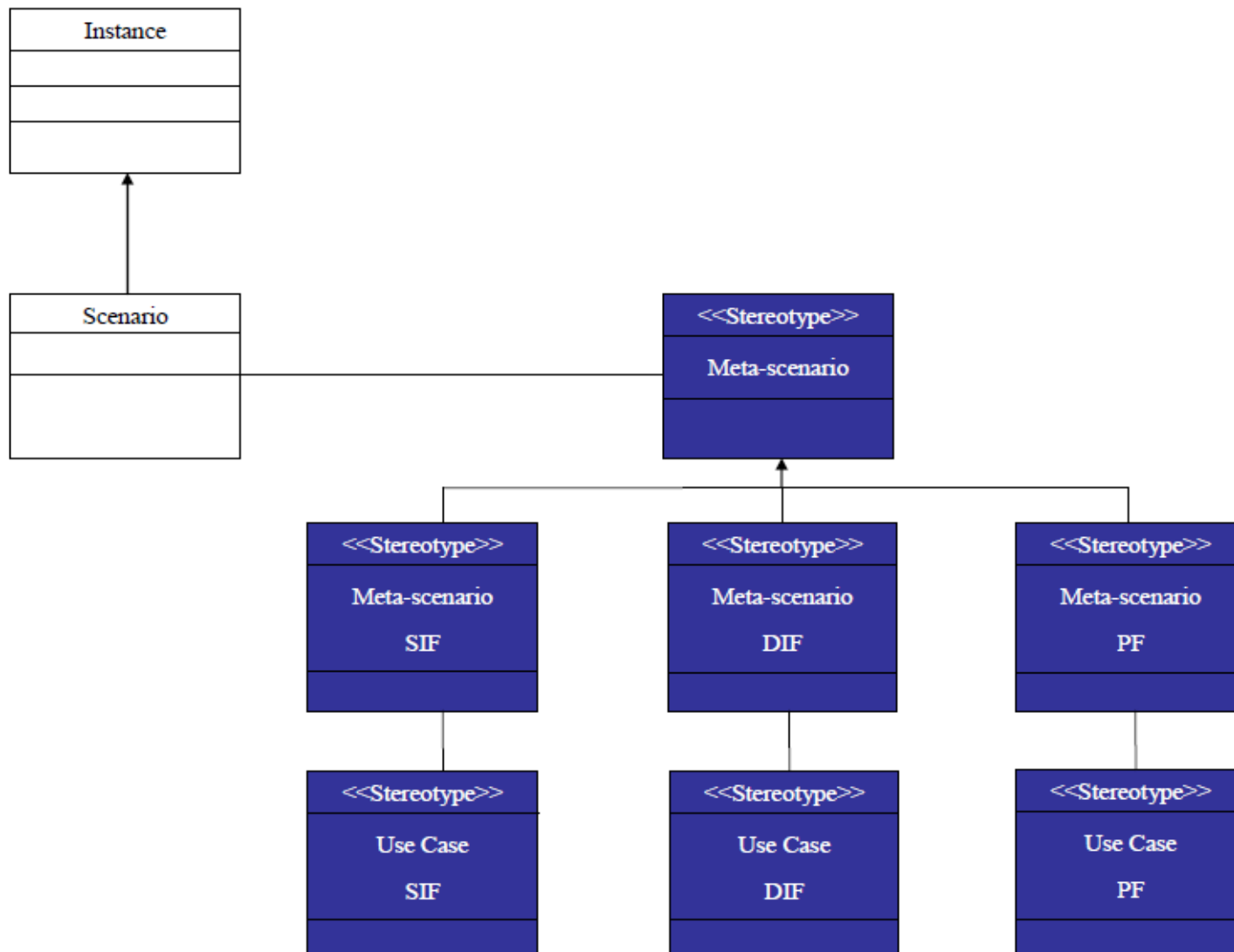


Figure 4.35: Third part of iUML use case diagram meta-model

# CHAPTER 5

## 5. Tool Support

All of the UML extensions' modeling elements and the integrated meta-model were modeled and integrated by a special diagram editor tool, called Dia [39]. Dia is free software that allows the user to create diagrams with the aid of a wide selection of modeling elements. Elements come from domains like Cisco, Database, Electric, Flow Chart, UML and others. Dia tool is known for its simple and easy-to-use environment. Dia makes it easier to control and manage the drawn elements of diagrams through the provided properties attached to each element. The drawing mechanism in Dia is as easy as using the Paint tool found in Microsoft Windows releases. It is easy-to-handle and flexible. Figure 5.1 shows Dia environment interface.

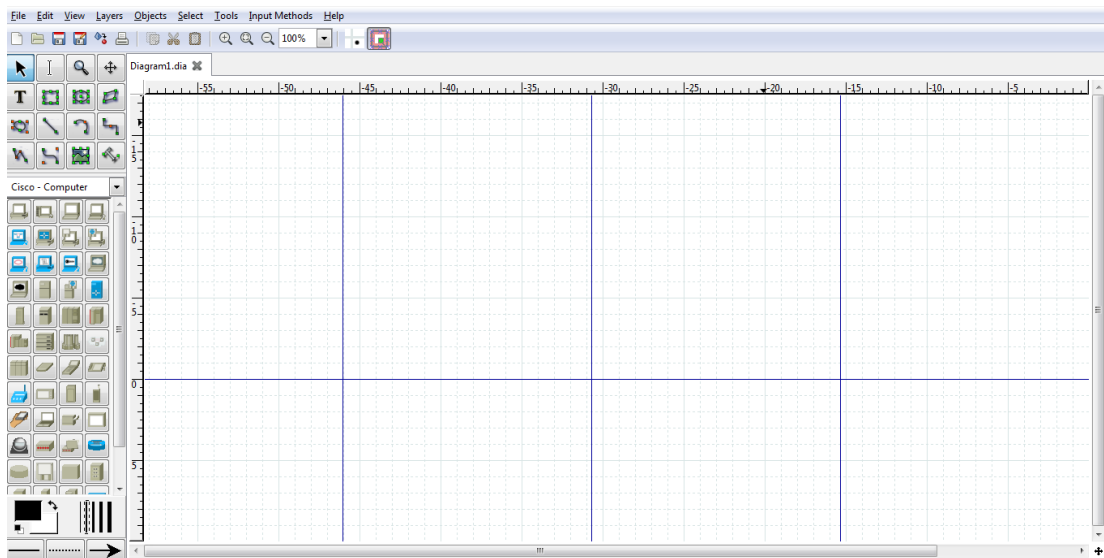


Figure 5.1: Environment of Dia

Using Dia, the user can insert text, control the size of the drawn elements and enter properties for such elements. What makes Dia more interesting than the other diagram editor tools is its ability to control and specify the diagram elements. Each element in the diagram has properties. For example, the element Class has properties like name, attributes, operations, etc., can be specified by the user by double clicking the element in the diagram and then entering the desired information. The user can also choose if he wants the class to be an abstract or the class's attributes to be visible or not. Another feature is the ability to create stereotype for the user's class which makes the procedure of extending the diagram easier, just a simple text-entering procedure. Figure 5.2 shows a screen shot of how Dia allows the user to specify the properties of a class.

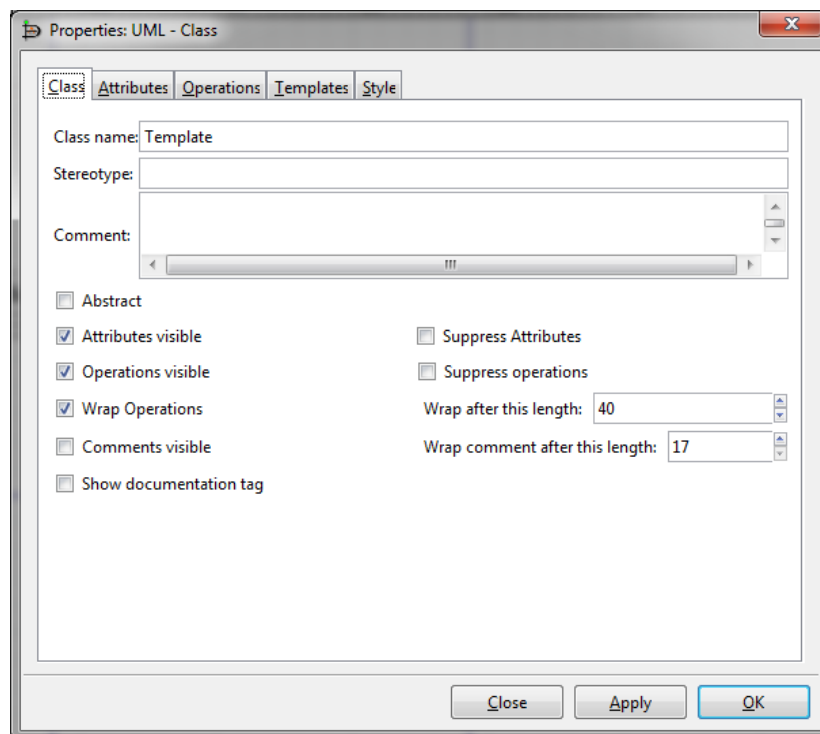


Figure 5.2: Properties of Class

Another extraordinary feature found in Dia is the option to create a sheet of modeling elements, in other words, drawing elements from scratch and save them in a special library or sheet. This sheet can be then listed in the main menu of sheets and can be easily used.

In this research, Dia was used to help in creating the integrated graphical extensions. The need was for a diagram editing software that provides flexible editing tools that makes the process of integrating graphical symbols easy and straightforward. Besides that, there was a need for software like Dia to store the final integrated symbols in a ready-to-use library and as mentioned earlier, Dia provides a way to store the created symbols in sheets. After saving the symbols in a sheet, they will be easily selected and used during the creation of diagrams.

An iUML sheet was created using Dia [39]. This sheet contains modeling elements from the collected UML extensions, plus, the integrated ones. Figure 5.3 shows the iUML sheet.



Figure 5.3: iUML sheet

An example of created modeling elements is the three integrated classes proposed by Fernandez-Medina et al. [12] and Byeon et al. [23] is shown in Figure 5.4. Fernandez-Medina et al. proposed security constraints, like security levels and roles to be placed on the elements of a hospital system and Byeon et al. suggested that the class graphic format can be vertically divided to include helpful graphical iconic notations. The result is integrated classes like the ones shown in Figure 5.4.

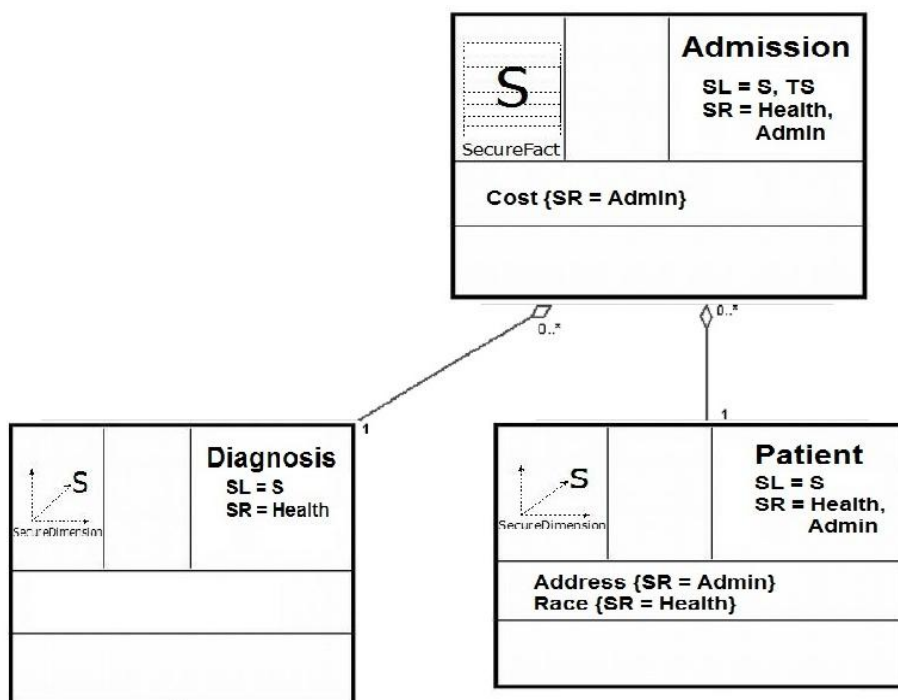


Figure 5.4: iUML integrated classes created using Dia

The class diagram shown in Figure 5.5 is created using Dia. Three classes are created; Student, GPA and Registrar. Class Student is a component class that satisfies the requirements of class GPA, a requirement class. The three classes (symbols) in this



example are iUML symbols. The way the classes are drawn is the result of integrating two extensions, Mahmood and Lai [28] and Byeon et al.[23].

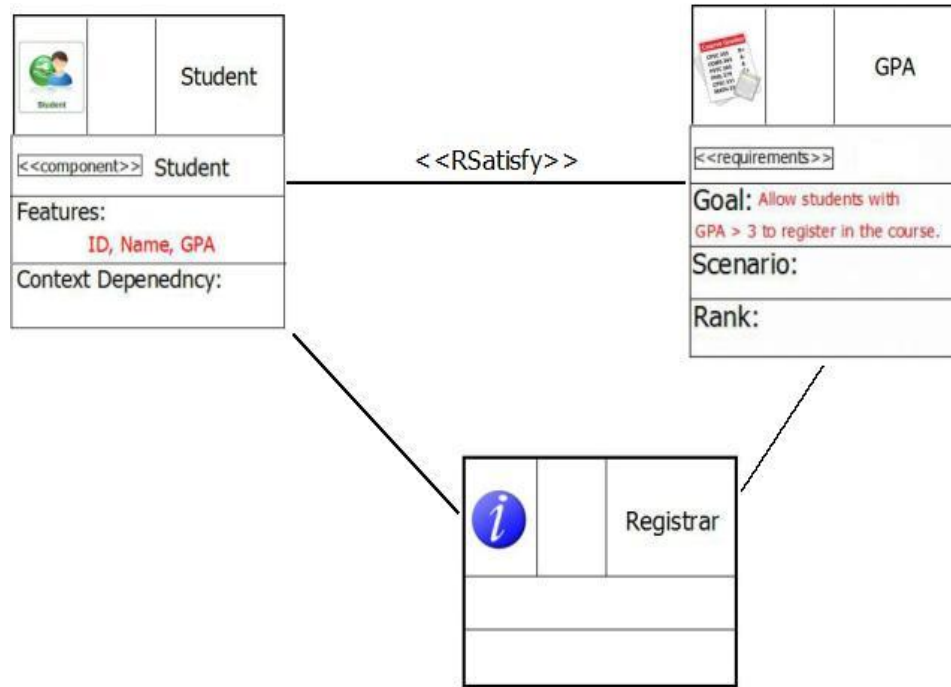


Figure 5.5: iUML class diagram example created using Dia

# CHAPTER 6

## 6. Validation

### 6.1 Class diagram case studies

This chapter gives a number of case studies derived from the literature to show the use of the integrated UML extensions.

#### 6.1.1 Case study # 1: Secured Health Care System (Data Warehouse + Security + GNSS)

This case study addresses the issue of systems' security, especially, health care systems'. Health care systems, placed in hospitals, handle tremendous amounts of indoor and outdoor patients' records. Such records store information about patients, like; personal information, financial issues, physical tests results, medical history background, current health condition, etc.

##### 6.1.1.1 Problem Description

Some of the hospital information are considered private matters and need to be only checked and accessed by the concerned staff or in other words, treating physicians. The health care system must be secured for many reasons. For example, patients' confidential and sensitive data need to be tightly locked away not only from outsiders but also from

non-concerned personnel, like; receptionists or laboratories staff who are privileged to access certain information only.

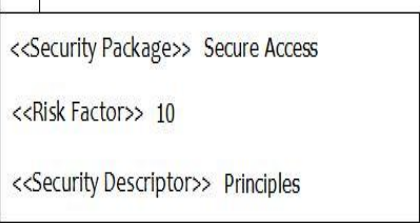
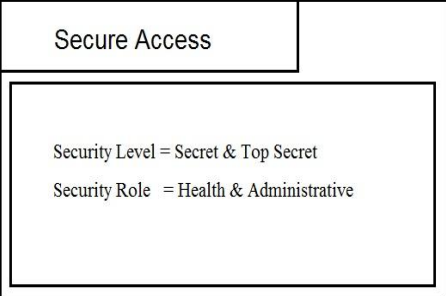
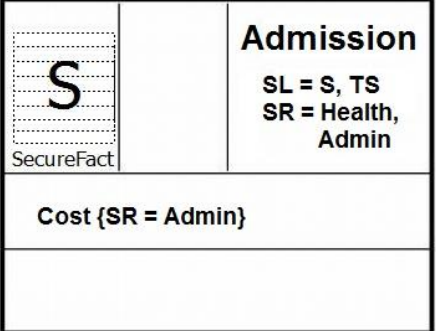

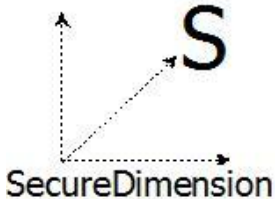
Using UML to enforce security measures would require extensions to UML that add different modeling elements with different techniques which ensure that the modeled system is secured enough. It would also focus on only one domain.

In iUML, the user uses one integrated form to cover security concerns for multiple domains; data warehouse and secured class diagram design. The previous extensions to UML, by Fernandez-Medina et al. [12] and Peterson et al. [24], are security techniques that are limited to specific domains. On the other hand, in iUML, the user can take advantage of all the integrated security techniques available to address security concerns using modeling elements, i.e. stereotypes and tagged values that are general enough to work on any problem domain.

#### **6.1.1.2 Applying the iUML**

To create the class diagram for this system, we can take advantage of the stored graphical symbols in the library. Table 6.1 shows the iUML graphical symbols that will be adopted and used in the creation of class diagram.

Table 6.1: Excerpt of iUML library

Modeling element	Source	Use of the Modeling element
	<p>Peterson et al. [24] and Fernandz-Medina et al. [12]</p>	<p>The security package will be inserted into the class diagram and will be attached to the classes that need to be protected from security attacks. Each security package has three attributes: Risk Factor; which calculates the security attack, Security Tile; protects the main parts of a system and finally, Security Descriptor: which describes the security categories that protect specific parts of the system.</p>
	<p>Peterson et al. [24] and Fernandz-Medina et al. [12]</p>	<p>A Security Tile which protects parts of a system. It mostly contains tagged values specified by security analysts and it can be attached to security packages to cover more security concerns.</p>
	<p>Fernandz-Medina et al. [12] and Byeon et al [23]</p>	<p>A class icon with iconic representation to display graphical information along with textual information such as, class's name, security levels and roles.</p>
	<p>Fernandez-medina et al.2007 [12]</p>	<p>Used to represent security information and constraints.</p>
	<p>Fernandez-medina et al.2007 [12]</p>	<p>Used to represent dimensions within a multidimensional model.</p>

The following figure, Figure 6.1, shows the meta-model elements. The white boxes represent the original UML elements and the red-boxed ones represent the iUML stereotypes that are used in this case study from the iUML meta-model.

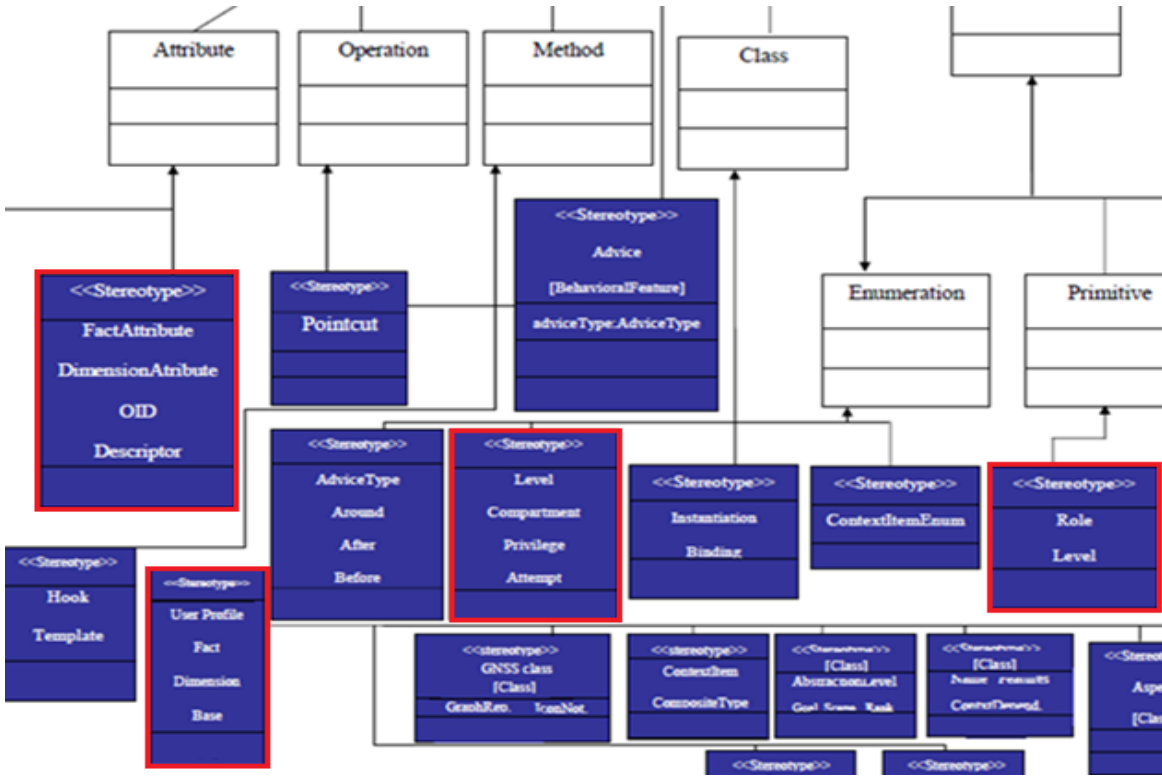


Figure 6.1: Excerpt of iUML class diagram meta-model

Figure 6.2 shows the iUML stereotype association “Protects” that will link the classes that need to be secured with the specified security packages.

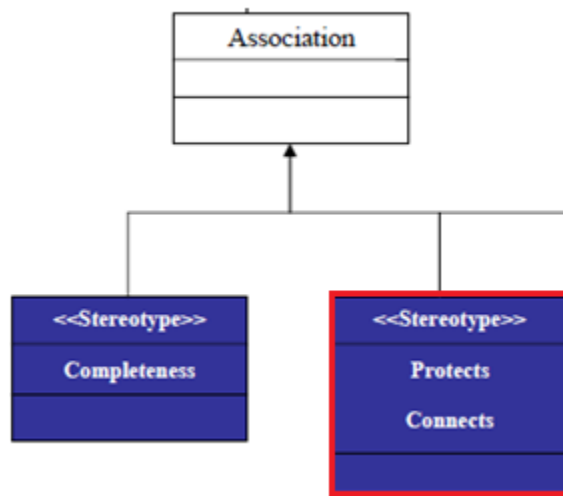


Figure 6.2: Excerpt of iUML class diagram meta-model

The overall goal is to incorporate security packages and tiles that are previously specified into the main elements of the system, i.e. elements that need security measures, such as; patient’s history records, diagnosis files, financial arrangements, etc. These security measures will ensure that these important data are only accessed by privileged users.

First, we have to define users of the system. Figure 6.3 specifies the health and non-health employees of the hospital. This helps in defining the privileged and non-privileged users of the system.

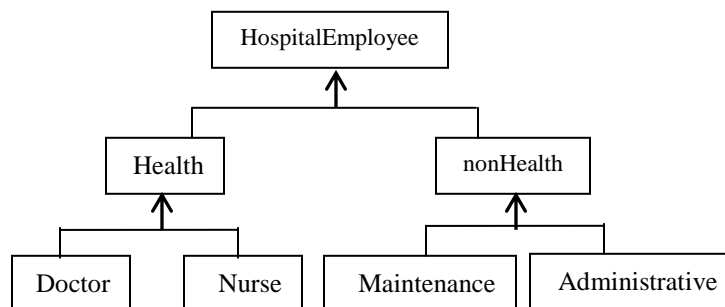


Figure 6.3: Hierarchy of users as suggested by Fernandz-Medina et al in [12]

The next step is defining the levels of security. These levels will be assigned to patients' data in their stored records. The constraints on these levels are placed on their values. The security levels must have values range only from [confidential, secret and top secret]. Figure 6.4 shows the defined levels of security.

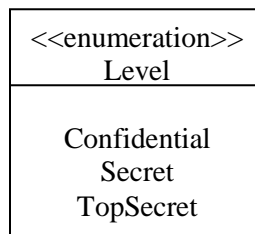


Figure 6.4: Levels of security as suggested by Fernandez-Medina et al. in [12]

After defining the users and levels of security, we have to define the information that has to be secured. We will define the privileged users who have access to the information (Security Role) and what levels of security will be placed over such information (Security Level). The following table, Table 6.2, describes the different types of records that need to be secured.

Table 6.2: Different types of Hospital's records

Element	Description
Admission	Contains individual admissions of patients of one or more hospitals.
Diagnosis	Contains the information of each user diagnosis.
Patient	Contains the patients' information.
Diagnosis Group	Contains a set of groups of diagnosis.
City	Contains the information of cities.
User Profile	Contains the users who will access the model.

Table 6.3 shows the assignment of security roles and levels over the hospital's records. Security roles and levels are expressed as sets of tagged values.

Table 6.3: iUML security roles and levels

Element	Tagged Value
Admission	<ul style="list-style-type: none"> <li>• Access by users who have: Security Level = Secret &amp; Top Secret &amp; Security Role = Health &amp; Administrative</li> <li>• The attribute "Cost" is accessed only by: Security Role = Administrative</li> </ul>
Diagnosis	<ul style="list-style-type: none"> <li>• Access by users who have: Security Level = Secret &amp; Security Role = Health</li> </ul>
Patient	<ul style="list-style-type: none"> <li>• Access by users who have: Security Level = Secret &amp; Security Role = Health &amp; Administrative</li> <li>• The attribute "Address" is accessed only by: Security Role = Administrative</li> <li>• The attribute "Race" is accessed only by: Security Role = Health</li> </ul>
Diagnosis Group	<ul style="list-style-type: none"> <li>• Access by users who have: Security Level = Confidential</li> </ul>
City	<ul style="list-style-type: none"> <li>• Access by users who have: Security Level = Confidential</li> </ul>

The tagged values shown in Table 6.3, will now be inserted into security tiles, as shown in Figure 6.5 through Figure 6.8.

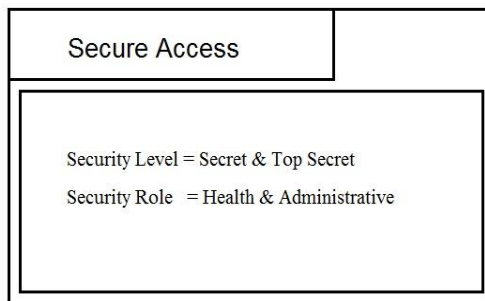


Figure 6.5: iUML security tile # 1



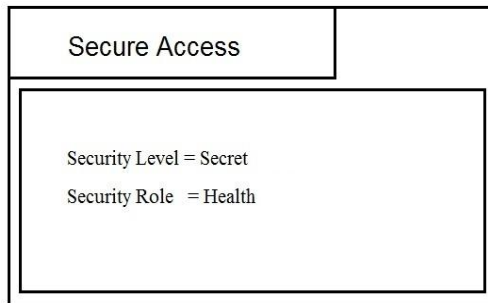


Figure 6.6: iUML security tile # 2

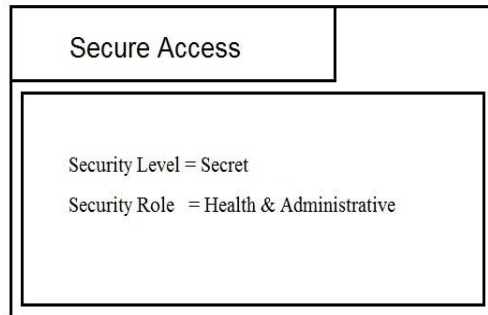


Figure 6.7: iUML security tile # 3

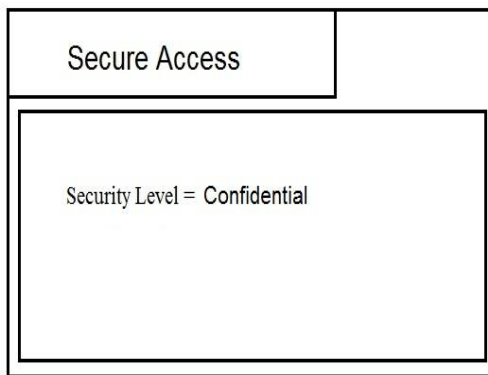


Figure 6.8: iUML security tile # 4

The next step is creating security packages. Security packages have to refer to the previously defined security tiles. This is done by writing the security tile's name next to <<Security Package>> label in the package, as shown in Figure 6.9 and Figure 6.10.

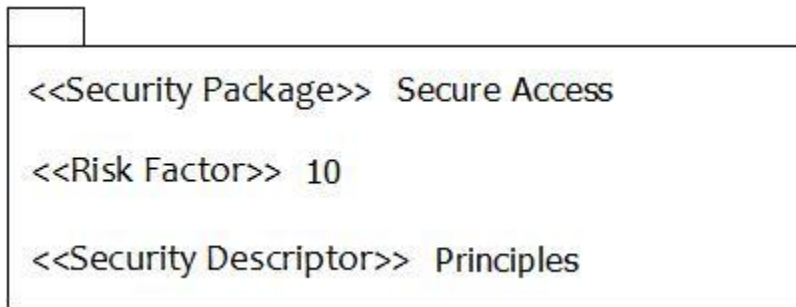


Figure 6.9: iUML security package (Secure Access)

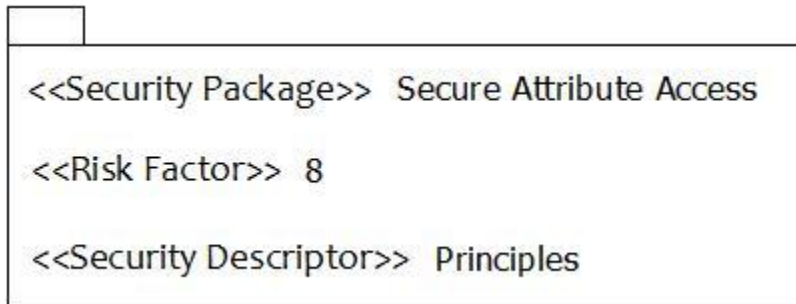


Figure 6.10: iUML security package (Secure Attribute Access)

The next step is to create the classes that represent the main elements of the system, Admission, Patient, Diagnosis, Diagnosis Group and City. Figure 6.11 shows an example of iUML class Admission. The goal of this design is to have unique and helpful graphical notations attached to the created classes.

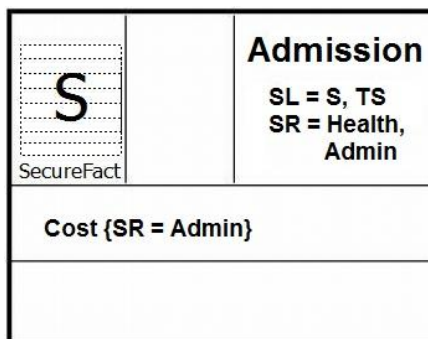


Figure 6.11: iUML classes Admission created using Dia

The final step is integrating security packages into the UML class diagram, as shown in Figure 6.12. Each security package protects a certain type of hospital's records, which are represented as classes in the diagram.



### 6.1.1.3 Discussion

For this case study, some modeling elements were used from iUML to consider some issues that were not handled and addressed by UML. The graphical symbols found in this case study were used to emphasize the issue of security and how to map it graphically to iUML class diagram. Figure 6.13 shows an example of a security package that was especially created to be used in domains that require security measure.

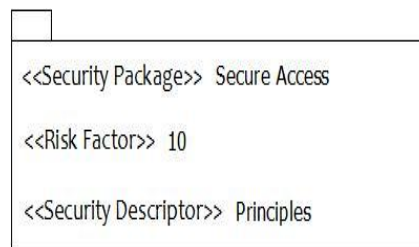


Figure 6.13: iUML security package

Attaching graphics to classes also helps the classes to be more readable. Having the first row of the class vertically divided helps attaching more and more information about the class in small compartments, such as; iconic notations, class's name, security levels and roles. Figure 6.14 shows iUML design of an Admission class.

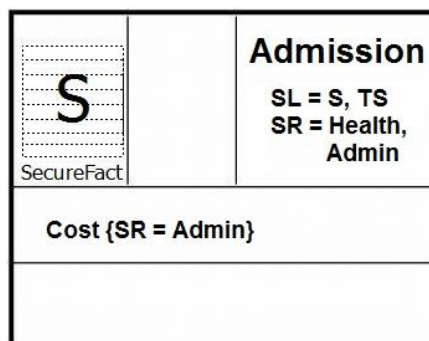


Figure 6.14: iUML classes Admission created using Dia

Beyond that level, the meta-model elements were introduced to make UML more specific to the introduced domains. For example, the iUML stereotypes shown in Figure 6.15 were used to add security levels and roles on every element in their domain model.

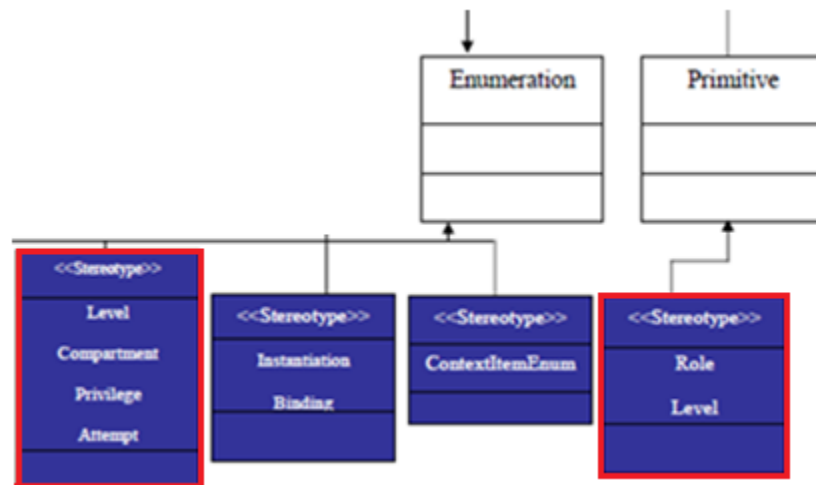


Figure 6.15: iUML stereotypes

The essence of UML is the ability to model the targeted system using a set of graphical notations. The limited set of UML graphical notations can help the system's designer to better visualize the system's internal and external elements but at the same time and as mentioned before, this set is limited. Unfortunately, UML was found unable to address some problem domains. UML has to be adapted and extended for such domains. Fernandez-Medina et al.[12] applied their extension to UML for the conceptual design of a secure Multi-dimensional model within the context of a typical health care system. Byeon et al.[23] provided notational help to obtain precise measurements and precise calculations of real-world geographical entities. And Peterson et al. in [24] used a UML class diagram to represent an ATM model integrated with UMLpac for possible security considerations. Without extending UML, it would be challenging for UML to model the secured health care system using the regular notations and other modeling elements.

Stereotypes and especially tag definitions must be defined in order to enforce secured access to patients' records. Along with that, security packages and tiles, mentioned in this case study, create another shield to prevent such important records from security attacks. The key issue is to specify more and more security measures and techniques to protect the stored information.

iUML integrates different extensions, concerning different and similar domains, for the sake of using one comprehensive set of graphical and meta-model concepts when dealing with a number of domains. Without using iUML, one cannot place more security techniques over the multidimensional elements like patient, admission, diagnosis, etc. iUML handles the security by setting tagged values and constraints in the data warehouse application domain and that can be greatly enhanced, security-wise, by attaching security packages to the elements found in the data warehouse domain.

### **6.1.2 Case study # 2: Grade Recording System (GNSS + Component + Design Patterns)**

The presented grade recording system in this case study is not something new. But what makes this system different than the other grading systems is its framework. The proposed framework involves a combination of three ideas; requirements and requirements satisfaction, helpful graphical notations and composed design patterns visualizations. This case study shows a normal grade recording system but from a different point of view; a requirement satisfactory point of view. The simple idea of a student is being or not being able to register in a course due to his GPA will be presented as a requirement. The

student, with his grades recorded in a database, will have to score a certain GPA in order to be able to register. This GPA condition is shown as a requirement.

#### **6.1.2.1 Problem Description**

The proposed grade recording system requires three ideas; requirements and requirements satisfaction, helpful graphical notations and composed design patterns visualizations. UML has to be extended to achieve the objective of this case study. It must include Mahmood and Lai work in [28] where they suggested a requirement-component relationship that states a certain component has to satisfy the customer's requirements through the presented features. In addition to that, UML has to be graphically extended to include the new graphical representation of classes which was suggested by Byeon et al. in [23]. The new format of the class icon allows the designer to attach helpful graphical notations.

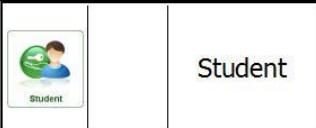
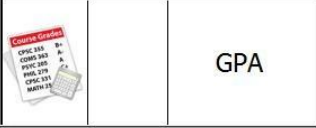
In that sense, UML will not be able to model the targeted system, it has to be extended. The problem with the needed UML extensions is that each extension is specific to one and only one problem domain, hence, the needed extensions cannot work together to achieve the objective of this case study.

#### **6.1.2.2 Applying the iUML**

The modeling elements of the class diagram for this case study will be taken from the iUML graphical library. Table 6.4 shows the iUML graphical symbols that will be used in this case study.



Table 6.4: Excerpt of iUML library

Modeling element		Ref.	Description	
 <p>Student</p>	<p>Student</p>	<p>Byeon et al. 2004 [23]&amp; Mahmood and Lai 2009 [28]</p>	<p>The new main elements of the class are three vertical compartments to indicate symbolic icons, iconic notations and class name, and &lt;&lt;component&gt;&gt; to specify component features.</p>	
				<p>&lt;&lt;component&gt;&gt; Student</p>
				<p>Features: ID, Name, GPA</p>
				<p>Context Dependency:</p>
 <p>GPA</p>	<p>GPA</p>	<p>Byeon et al. 2004 [23]&amp; Mahmood and Lai 2009 [28]</p>	<p>The new main elements of the class are three vertical compartments to indicate symbolic icons, iconic notations and class name, and &lt;&lt;requirements&gt;&gt; to specify stakeholder requirements.</p>	
				<p>&lt;&lt;requirements&gt;&gt;</p>
				<p>Goal: Allow students with GPA &gt; 3 to register in the course.</p>
				<p>Scenario:</p>
				<p>Rank:</p>

The following figure, Figure 6.16, shows the meta-model elements. The white boxes represent the original UML elements. The red-colored boxes clarify the proposed modeling elements from [23], [26] and [28].

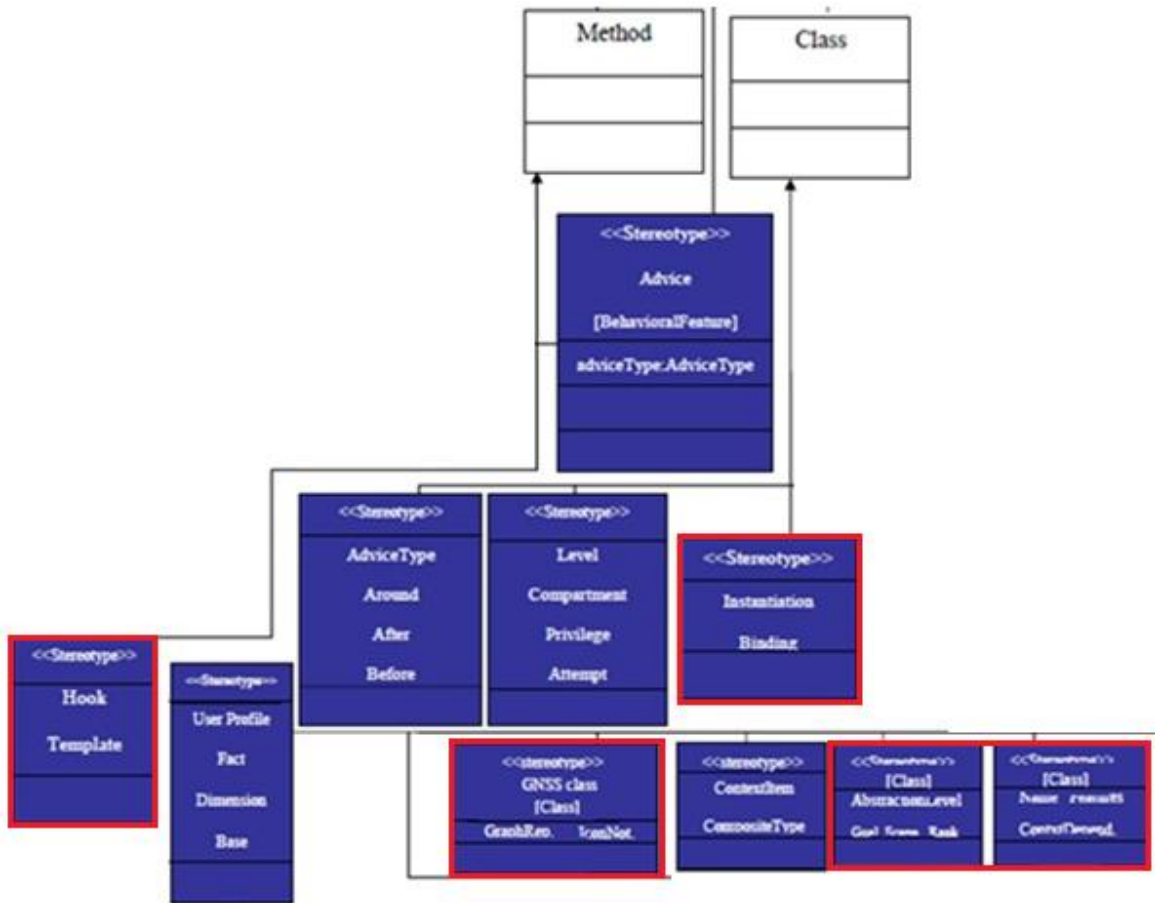


Figure 6.16: Excerpt from the integrated class diagram meta-model

The first step is to create the classes that represent registrar, students, teachers, lectures and tests. Figure 6.17 shows the iUML classes. The goal of this design is to have unique and helpful graphical notations attached to the created classes.

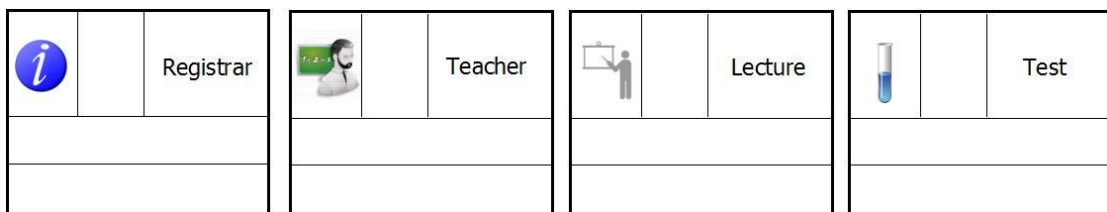


Figure 6.17: iUML classes' design inspired by Byeon et al. [22] created using Dia

The following classes are advanced iUML classes. The GPA class is a requirement class that is handled by the registrar to represent a certain requirement that must be satisfied by a component class, a student class.

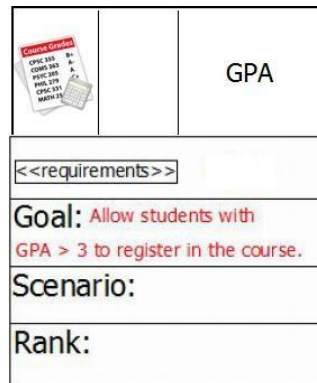


Figure 6.18: The GPA requirement class created using Dia

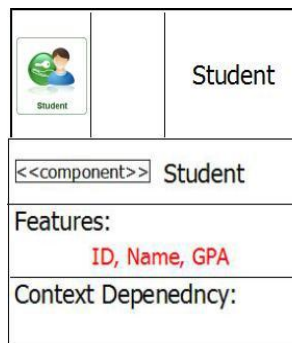


Figure 6.19: The Student component class created using Dia

The GPA requirement class are handled by the registrar class where each student who wants to register the course (or lecture) has to satisfy the requirment (GPA > 3).

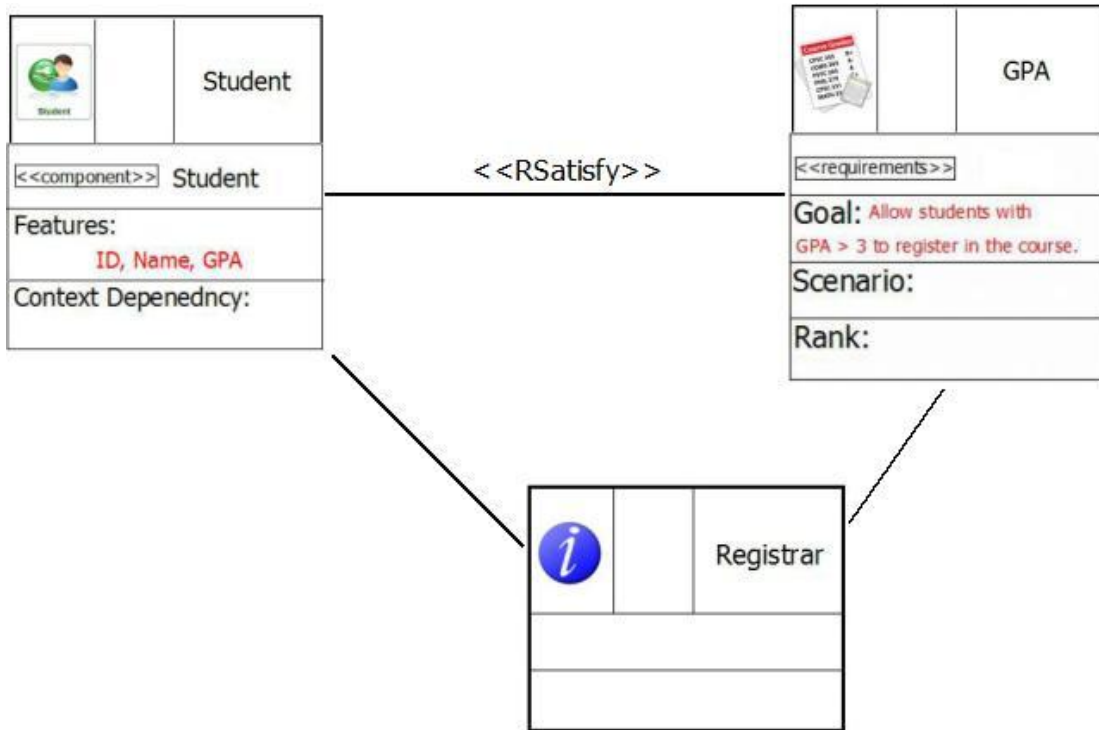


Figure 6.20: R Satisfy relationship created using Dia

The stereotypes that are used in this case study are presented in Table 6.5. Stereotypes “Template” is used to define abstract behavior and “Hook” is used for implementation. For example, to compute the grades, a method called “Compute” will be used as a Template method in class “Test” while the implementation of this method is handled in a Hook method in another class, the Lecture class.

Table 6.5: iUML modeling elements (stereotypes)

Stereotype	Base Class
Hook <<Hook>>	Method
Template <<Template>>	Method

The previous concepts are presented in the following iUML class diagram, Figure 6.21.

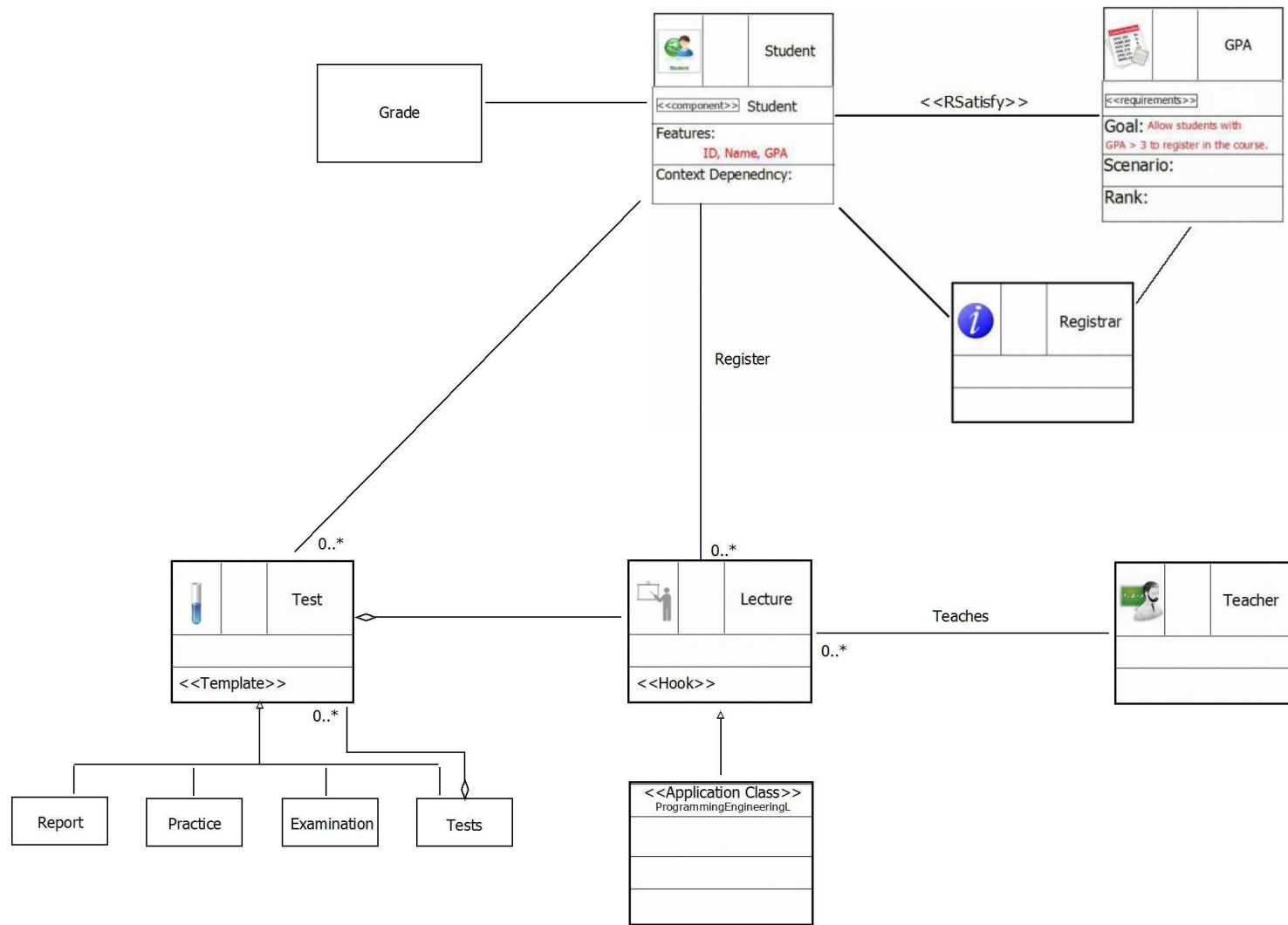


Figure 6.21: iUML UML class diagram (Grade Recording System)

### 6.1.2.3 Discussion

This case study was presented and discussed only to show another advantage of using iUML. Grade recording systems' design can be created and illustrated using UML constructs but using the iUML helps this kind of systems from two perspectives; graphical and analytical.

The graphical advantage of iUML is the use of graphical notations that are attached to the classes. Attaching graphics to classes helps the classes to be more readable and distinguishable. Also having the first row of the class to be vertically divided helps attaching more information about the class in these small compartments, such as; iconic notations and class's name. Other information may include references to other classes or dependencies on other classes. Figure 6.22 shows iUML design of a student class.

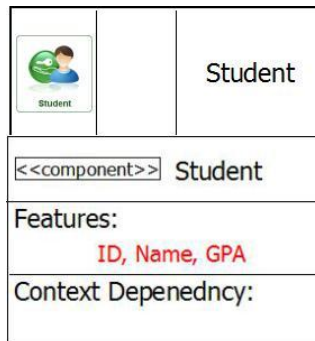


Figure 6.22: iUML student class created using Dia

In iUML, the graphical symbols are integrated and can be used to graphically model any problem domain or multiple domains at the same time. In Figure 6.22, the class icon integrates three graphical notations in which can be used to present both textual and graphical information about that class. One of these extensions suggested adding graphical icon, another extension proposed dividing the first row of the class icon

vertically so it can include textual and graphical information. The last extension proposed adding the tag “component” or in other cases “requirement” to include more information about what this class can present or require to or from other classes. Using UML would require using the three extensions one at a time and each in a different problem domain.

On the other hand, the analytical advantage of using iUML comes from the use of requirement and component classes. These classes help the analysts of the system under study to enforce requirements satisfaction between classes. In other words, a requirement class will require a certain condition that must be met by a component class and then and only then the relationship between both classes would be labeled as a satisfactory relationship. In this case study, the GPA and student classes were presented as requirement and component classes, respectively.

UML has to be extended in order to address the covered domains. It has to be extended to include Byeon et al. work in [23] where they provided notational help to obtain precise measurements and precise calculations of real-world geographical entities. UML has to be also extended to cover the component-based systems which come from [28] where Mahmood and Lai specified satisfaction and risk assessment for evaluating customer demands against component features. Finally, and in order to model and visualize composed design patterns and represent frameworks., UML has to be extended to include Sanada and Adams work in [26]. All of the previous extensions have to be done separately using UML while iUML provides an integration of these extensions that can be integrated altogether to achieve the objective of this case study.

### **6.1.3 Case study # 3: Meeting Scheduling System (Object-Oriented + Mobile Distributed System)**

Meetings scheduler is a needed system in companies, universities and other forms of organizations. This system can be embedded in an available system to handle and arrange upcoming meetings. It can look up for available dates, available meeting rooms, and available participants. It also can send memos regarding upcoming and previous meetings.

#### **6.1.3.1 Problem Description**

The meeting scheduling system is considered a mobile distributed system, with elements like Person, Meeting, Calendar, Current Activity, etc. Some of these elements need to be shown as static or dynamic elements in the model. This type of indication is needed in systems like mobile distributed systems due to their changing statuses. An example of that is the status of a staff's availability whether he is available or not available for an upcoming meeting.

This issue can be modeled using UML but it requires tag definitions, the ones proposed by Fontoura et al. in [22]. A tag definition can be attached to an element of a class diagram to indicate that the element is a static or a dynamic element. In other words, using UML, the user has to extend UML and adopt specific modeling elements to model mobile distributed systems. In other words, UML will not be able to model the targeted system unless it is extended. But each UML extension is specific to one problem domain, hence, the needed extensions cannot work together to achieve the objective of this case study.



### 6.1.3.2 Applying the iUML

One modeling element is used on the class diagram for this case study that is taken from the iUML graphical library. Table 6.6 shows the iUML graphical symbols that will be used in this case study.

Table 6.6: Excerpt of iUML library

Modeling element	Ref.	Description
{dynamic}	Fontoura et al. 2000 [22]	Used to indicate runtime variation point.

Figure 6.23 shows excerpt from the integrated class diagram meta-model. The red-colored boxes represent the proposed modeling elements from [22] and [13].

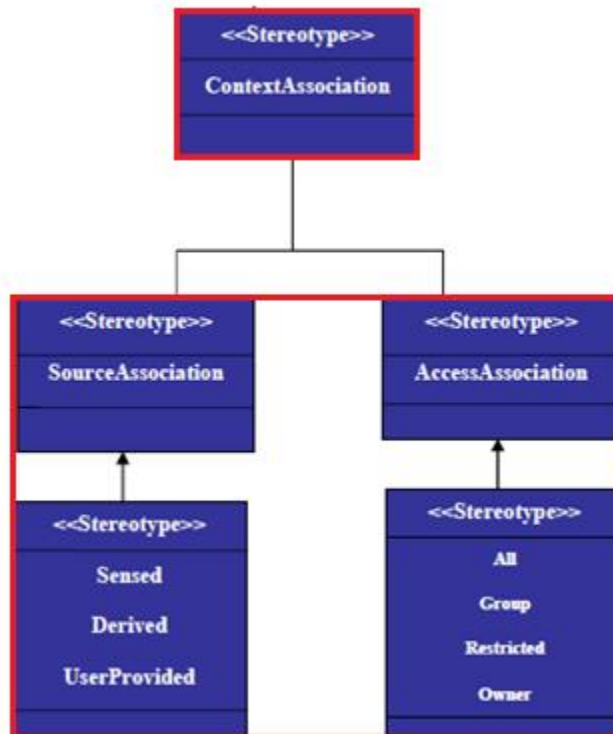


Figure 6.23: Excerpt from the integrated class diagram meta-model

For this case study, a Person class must be created. This class has a personal public calendar to show the occupied Time Slots. The current status of this Person must be

presented as Available or Busy. A Meeting class must also be created and it must show certain information such as; meeting's participants, meeting date, meeting place and topic description. Meeting Notes are attached with the Meeting class and can be accessed to all the participants. Finally, a Room class must be created as the meeting place. Figure 6.24 shows the UML class diagram composed of the previous elements.

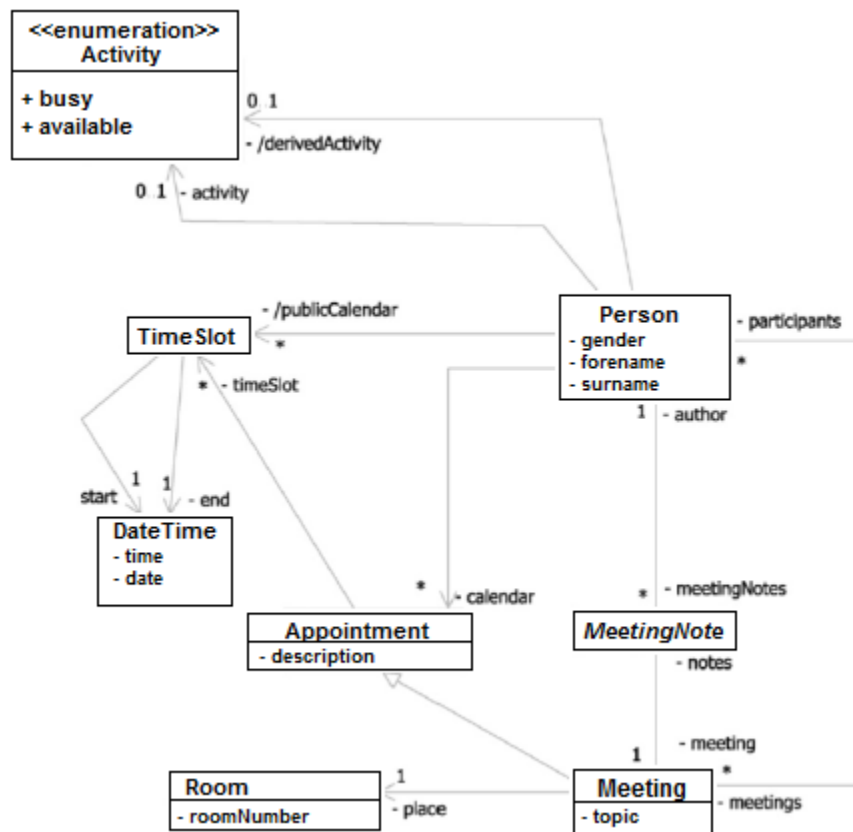


Figure 6.24: Class diagram for the Meeting system

The next step would be integrating the graphical extensions (Boolean tag) from Table 6.6 into the UML class diagram shown above. The Boolean tag, dynamic, was applied to the classes; Activity and Room since their information must be provided only during runtime. The integrated UML class diagram is shown in Figure 6.25.

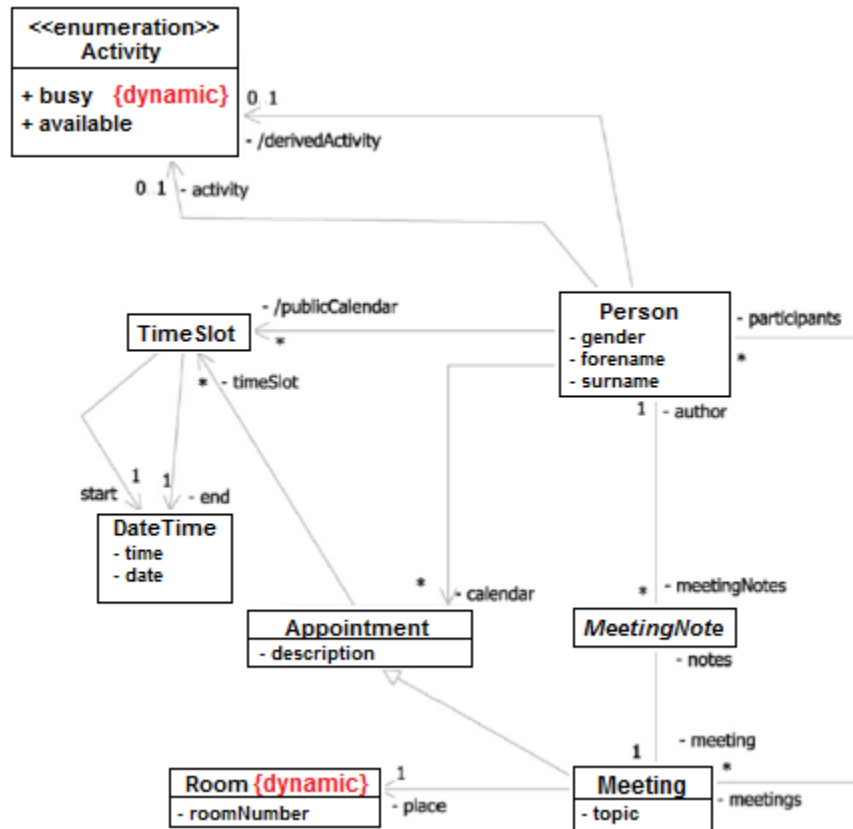


Figure 6.25: Integrated UML class diagram (Meeting Scheduling System)

### 6.1.3.3 Discussion

iUML integrates two UML extensions from the literature to model the system in hand. The first UML extension comes from [22] where Fontoura et al. proposed a UML extension that contains a set of Boolean tags to describe the structure of variation points in the object-oriented framework. The second extension comes from [13] where Simons and Wirtz presented the Context Modeling Profile (CMP), an extension to the UML to support the development of context-aware mobile applications. The drawback of UML manifests in its inability to cover the previous domains at the same time. UML can be extended to address one and only one problem domain while in iUML, a set of integrated modeling elements can be easily applied to model different domains at the same time.

Using iUML has enriched this case study in two ways; the first one is by using a new kind of association between classes that is considered an extension to UML. iUML provides an extended type of association called ContextAssociation and it is composed of two types; Source and Access. The ContextAssociation allows classes to associate with each other in new forms of relationship. The context concept enforces handheld or mobile devices in a system to create a specific type of association that implies that these devices can communicate with each other by exchanging signals, hence, update their status or behavior based on other devices' current status. To model that, iUML uses a new type of associations between classes. This also implies the role of having indicators attached to active or non-active classes. For this case study, a tag definition named Dynamic was placed on such classes to focus more on the idea of having classes or objects with a dynamic status in the environment. Their status will be only known during runtime.

## **6.2 Sequence diagram case studies**

### **6.2.1 Case study # 4: Elevator Control System (Quality of Service + Component)**

The elevator system is a simple system and can be easily modeled using UML. The functions available for the system are straightforward and require an input from the user. But using UML with its limited set of modeling elements and notations do not focus on issues like the quality of the provided service through its modeling techniques. UML had to be extended to cover such issues in order to be able to model domains like component-based systems more effectively. In iUML, such concerns are considered by including all

the necessary notations. The modeling of an elevator control system can be improved to a certain degree to handle issues like quality of service in a requirement satisfactory point of view. The stakeholder or user requires a certain requirement (calling the elevator), the working system has a set of components and each one of them covers a specific angle of that working system in a way that makes the whole system responds to that requirement, hence, satisfying the user's requirement.

### 6.2.1.1 Problem Description

The system has to schedule elevators and control the motion of the between floors. To ensure the quality of the provided service, the interacting components of this system must provide the desired features that fulfill the requirements of the user. Using UML, the user can only use one domain-specific set of stereotypes to model one and only one domain.

### 6.2.1.2 Applying the iUML

The following table, Table 6.7, describes iUML stereotypes that will be used to model the elevator control system.

Table 6.7: iUML modeling elements (stereotypes)

Stereotype	Description
REservice	Represents a sequence of actions and interactions.
REcomponents	Represents the main interacting objects.
REconnectors	Represents the means in which the REcomponents interact through.
REuser	Represents the party that triggers the actions.
REhost	Represents the hosting party of REcomponents.

Figure 6.26 shows excerpt from the integrated sequence diagram meta-model. The red-colored boxes represent the proposed modeling elements from [28] and [32].

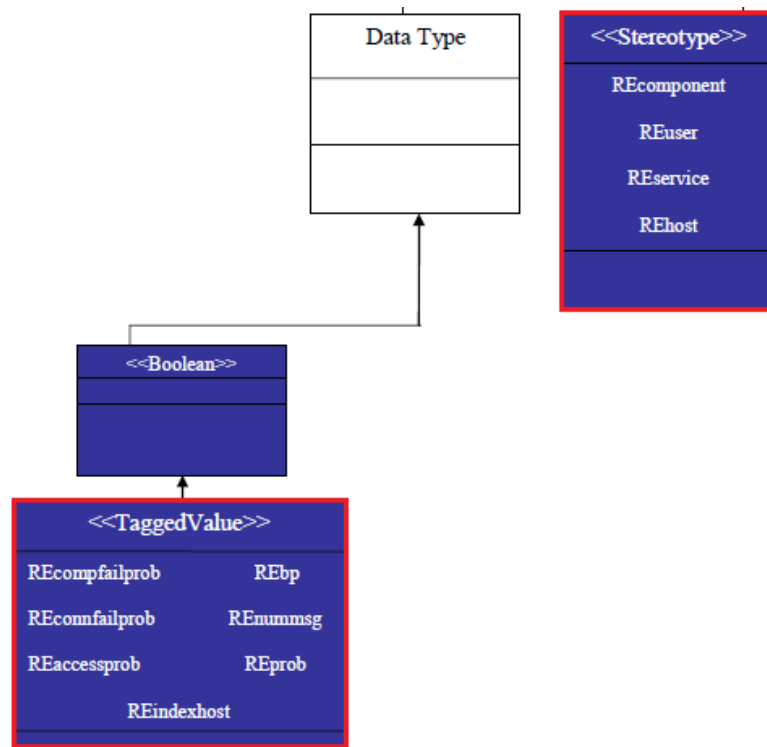


Figure 6.26: Excerpt from the integrated sequence diagram meta-model

Cortellessa and Pompei in [32] developed the following sequence diagram based on the proposed stereotypes.

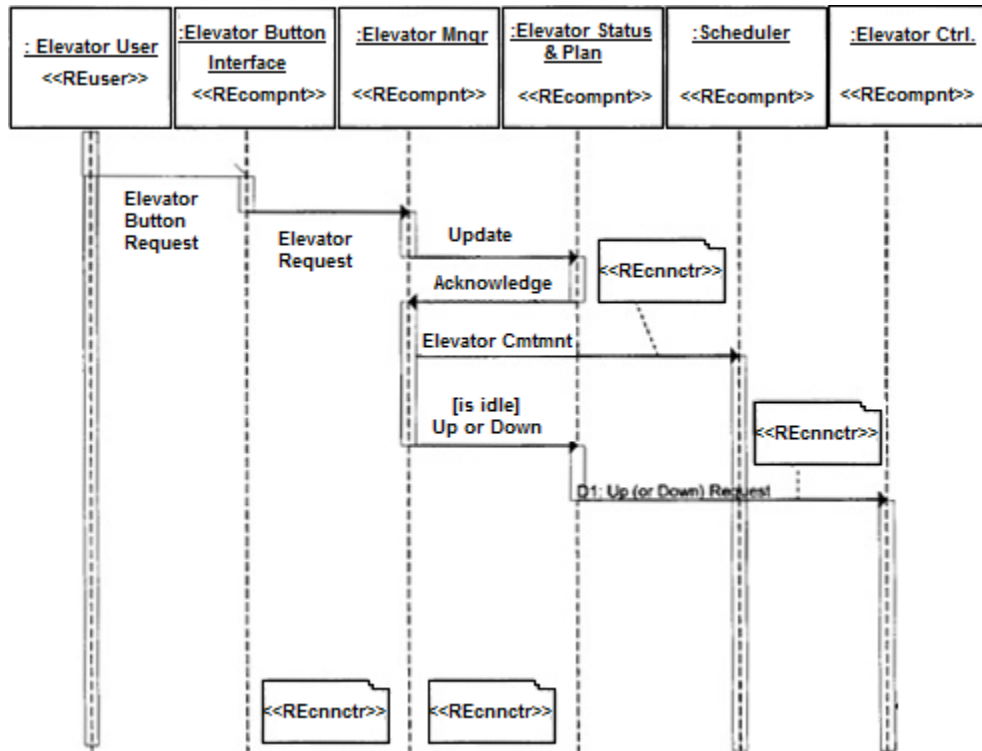


Figure 6.27: Sequence diagram for Select Destination System

iUML Requirements Class and Component Class will replace REuser and REcomponent in the previous diagram. The goal is to treat the service as a requirement inquired by the user and provided by the system.

Figure 6.28 shows the replacement of <<REuser>> with <<Rstakeholder>> and <<REcomponent>> with <<Rcomponent>>C<sub>n</sub> and finally place Satisfaction box around the interactions.

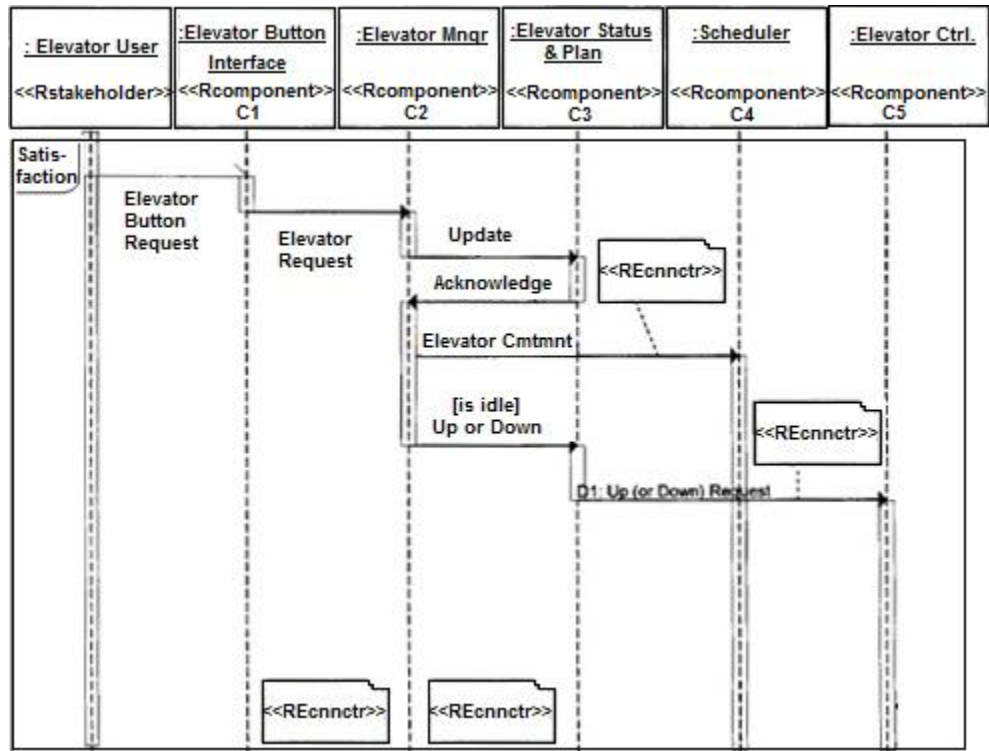


Figure 6.28: Integrated UML sequence diagram (Elevator Control System)

### 6.2.1.3 Discussion

As mentioned before, the elevator controlling system can be modeled using UML but to focus on the issue of quality of service, UML has to be extended in a way to include more modeling elements such as stereotypes that model a set of connected components that provides services to the user's system. UML can work on a single domain at a time. In other words, it has to be extended once to represent only the non-functional attributes such as Quality of Service and Fault Tolerance and once again to address only component-based systems.

The novelty of iUML is its integration of two extensions that helped modeling this case study, i.e. the elevator controlling system. It models the non-functional issues such as quality of service and fault tolerance and models also requirements engineering issues.



The way this system was modeled is as a requirement satisfaction system. The system schedules elevators to respond to requests from users at various floors and controls the motion of the elevators between floors. The system is composed of a set of components; these components must provide features that are required by the user or stakeholder.

## **6.3 Use case diagram case studies**

### **6.3.1 Case study # 5: E-Commerce System (Agent + Adaptive Web Application)**

Electronic commercial websites like Amazon and eBay provide electronic shopping experience for the users. Such websites store large amounts of merchandises and build a database to include information about their names, categories, quantities and descriptions and they made them available to be accessed by users when they search for them. This kind of websites requires a very robust monetary transaction embedded system that is linked to the user's credit card account. For that reason, modeling a commercial system must show and enforce a secured and easy to use model.

#### **6.3.1.1 Problem Description**

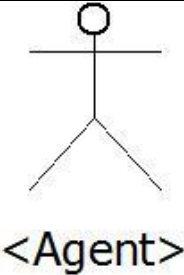
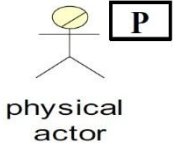
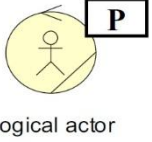


Electronic commercial systems require modeling two important things; first, by enforcing easy to use interface to the user and secondly by securing user's financial information and transaction. The former one is a favored concern but focusing on the issue of security is a mandatory matter.

This system can be done using UML, but the user has to extend UML to address agent-oriented systems once and address adaptive web applications once again. The point is the user cannot model both domains at the same time; he can only use one domain-specific set of stereotypes to model one and only one domain.

### 6.3.1.2 Applying the iUML

To create the use case diagram for this system, we can take advantage of the stored graphical symbols in iUML library. Table 6.8 shows the iUML graphical symbols that will be used in this case study.

Table 6.8: Excerpt of iUML library

Modeling element	Source.	Description
	Fei and Yan 2008 [16]	Expresses that the entity is seen as an agent instead of a class.
	Djemmaa et al. 2006 [17]	Represents the human user who visits the web application.
	Djemmaa et al. 2006 [17]	Represents the role played by a human user (physical actor) to maintain the web application.
	Djemmaa et al. 2006 [17]	Represents the hardware aspect of the system, whether it is a computer system, device hardware or web service.
	Djemmaa et al. 2006 [17]	“SIF: Static Informational Functionality used to represent a static Web page.”



	Djemaa et al. 2006 [17]	“DIF: Dynamic Informational Functionality used to represent a dynamic Web page.”
	Djemaa et al. 2006 [17]	“PF: Profession Functionality used to represent a dynamic Web page using update request.”

Figure 6.29 shows excerpt from the integrated use case diagram meta-model. The red-colored boxes represent the proposed modeling elements from [16] and [17].

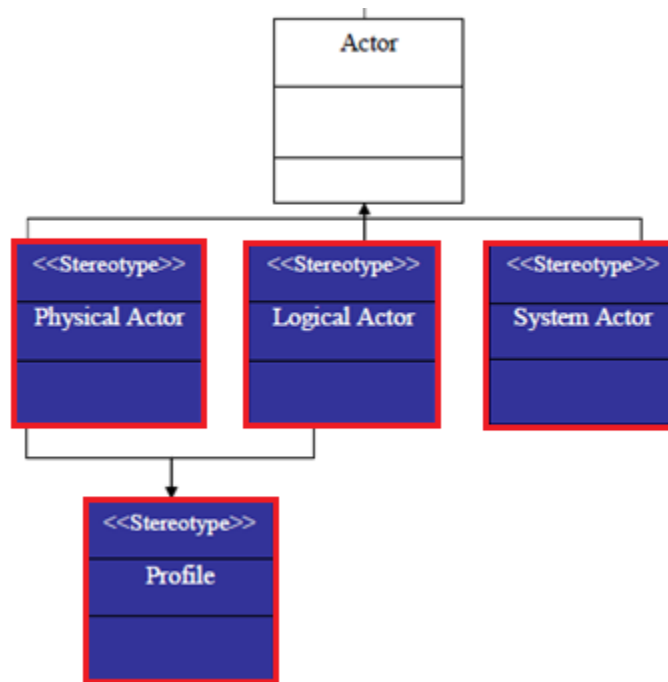


Figure 6.29: Excerpt from the integrated use case diagram meta-model

The system offers to the client when he logs in a number of options represented by use cases. These options are; Consult new, Search for book, Manage the basket and Pass command. Figure 6.30 depicts the system's environment.

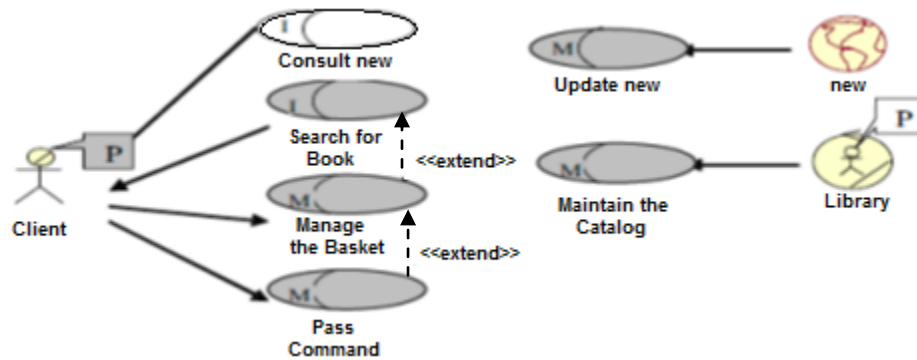


Figure 6.30: E-commerce system environment

A verification party (agent) will be added to the E-Commerce system. This verification party will be a System actor where it will ask the client for his/her credentials (User name & Password) by Static Information Functionality use case and when the client enters the correct information, the System actor will allow him to log in. Figure 6.31 shows the verification process.

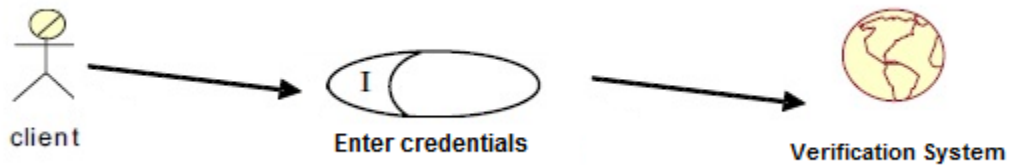


Figure 6.31: The verification process

### 6.3.1.3 Discussion

One of the advantages of using iUML is the ability to specify more functions (use cases) required to display information about the system in terms of static and dynamic information. For example, a special use case called Profession Functionality is used to display the dynamic elements of the commercial system like the user's basket that has a changing status. This advantage helps the system to be built in a defined and robust

manner. On the other hand, the static functionalities, like recommending products to the user, help the designers to add more ways to make the targeted system easier to use.

Using UML to achieve the objective of this case study is almost impossible, as UML cannot address different domains at the same time. It can be extended to cover one specific domain only. On the other hand, iUML models the e-commerce system by integrating two UML extensions from the literature. The first one comes from [16], where Fei and Yan presented a system called SPAERIS "Shipping Pollution Accident Emergence Reflecting Information System" using AUML (Aspect Unified Modeling Language). The second UML extension comes from [17], where Djemaa et al. proposed a UML profile called WA-UML (Web Adaptive-UML) to model Adaptive Web Applications.

### **6.3.2 Case study # 6: Elevator Control System (Quality of Service + Adaptive Web Application)**

The elevator system is a simple system and can be easily modeled using UML. The functions available for the system are straightforward and require an input from the user. But using UML with its limited set of modeling elements and notations do not focus on issues like the quality of the provided service through its modeling techniques. UML has to be extended to cover such issues in order to be able to model domains like component-based systems more effectively. In iUML, such concerns are considered by including all the necessary notations. It also models and displays the functions of the system in a set of static and dynamic information. This way of modeling helps the analyst builds a notational and graphical bridge between the analysis and design of the system.

### 6.3.2.1 Problem Description

The system has to schedule elevators and control the motion of the between floors. To ensure the quality of the provided service, the components of this system must be categorized whether they provide static service or a dynamic one. Using UML, the user can only use one domain-specific set of stereotypes to model one and only one domain.

### 6.3.2.2 Applying the iUML

The following table, Table 6.9, describes iUML stereotypes that will be used to model the elevator control system.

Table 6.9: iUML modeling elements (stereotypes)

Stereotype	Description
REservice	Represents a sequence of actions and interactions.
REcomponents	Represents the main interacting objects.
REconnectors	Represents the means in which the REcomponents interact through.
REuser	Represents the party that triggers the actions.
REhost	Represents the hosting party of REcomponents.

Figure 6.32 shows excerpt from the integrated use case diagram meta-model. The red-colored boxes represent the proposed modeling elements from [17] and [32].

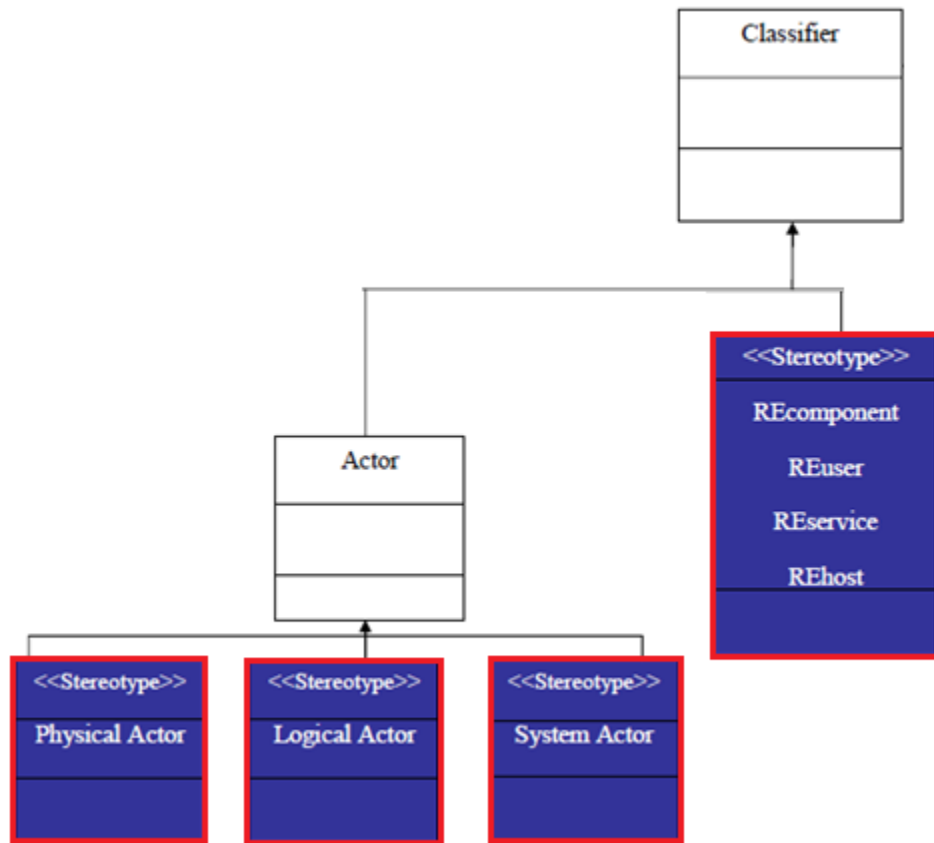


Figure 6.32: Excerpt from the integrated use case diagram meta-model

Cortellessa and Pompei in [32] developed the following Use Case diagram based on the proposed stereotypes.

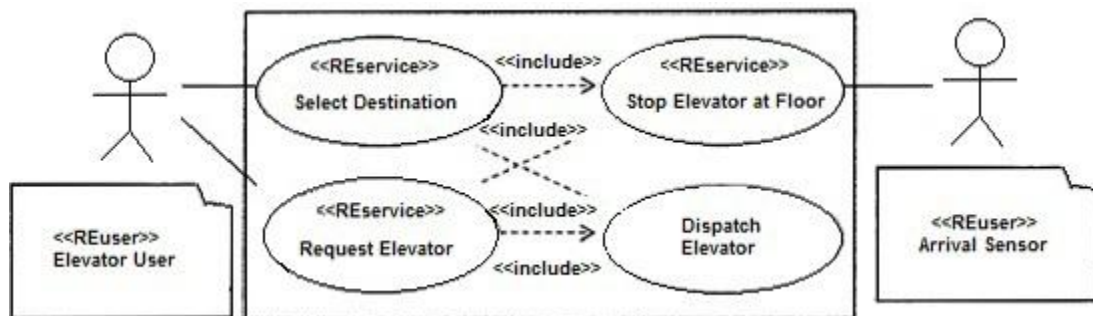


Figure 6.33: Use case diagram for Select Destination System

iUML replaces <<REuser>> actors in the previous Use Case diagram with iUML actors. The (Elevator User [32]) actor will be replaced by a (Physical Actor [17]) and the (Arrival Sensor [32]) by a (System Actor [17]) as shown in Figure 6.34.

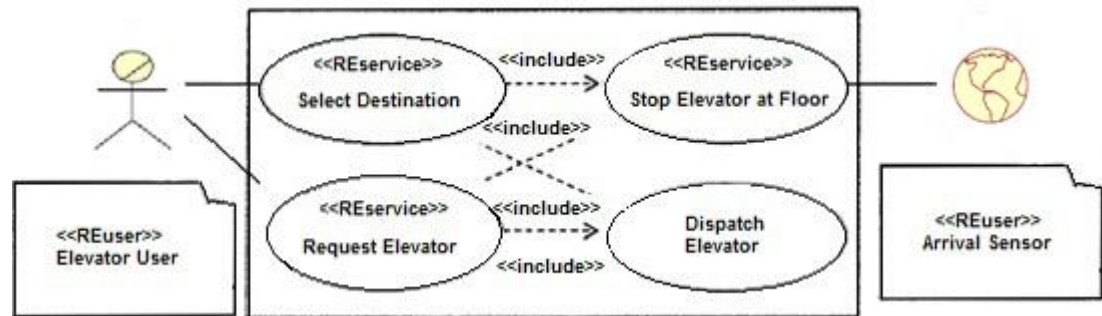


Figure 6.34: Extended use case diagram

iUML also replaces the <<REservice>> use cases with iUML use cases (functionalities). The <<REservice: Select>> and <<REservice:Request>> use cases are replaced by a SIF use case and <<REservice:Stop>> use case by a DIF use case as shown in Figure 6.35.

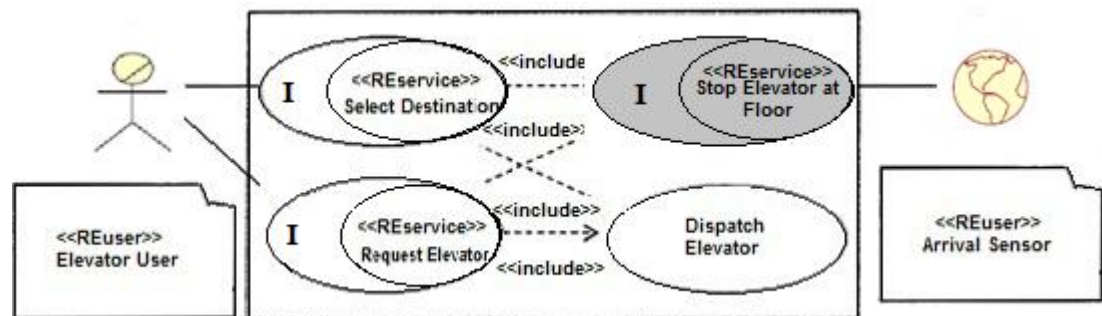


Figure 6.35: Integrated UML use case diagram (Elevator Control System)

### 6.3.2.3 Discussion

As mentioned before, the elevator controlling system can be modeled using UML but to focus on the issue of quality of service, UML has to be extended in a way to include more modeling elements such as stereotypes that model a set of connected components that



provides services to the user's system. But unfortunately, UML can cover one and only one specific domain. In other words, it has to be extended once to represent only the non-functional attributes such as Quality of Service and Fault Tolerance and once again to address only adaptive web applications.

The novelty of iUML is its integration of two extensions that helped modeling this case study, i.e. the elevator controlling system. It models the non-functional issues such as quality of service and fault tolerance and also models the adaptive functionalities. The way this system was modeled is as a display of functionalities in terms of static or dynamic. The selection and request functionalities were static because the elevator's status in that instant is idle, on other hand; the elevator in the stop request is busy.

# CHAPTER 7

## 7. Conclusion

The rationale behind the integration process was to come up with one form of UML in order to address a variety of problem domains. In the literature many UML extensions were proposed; each addressed a particular domain. Examples of these domains are; web hypermedia applications, aspect-oriented modeling, distributed systems, component-based software systems, data warehouses, design patterns, etc., but those UML extensions were specific to particular problem domains, in other words, such extensions are not applicable to other domains. The novelty of this research is to provide an integrated UML that supports, not just a single domain but a number of domains.

The first stage in this research was conducting a deep review of the literature in order to collect as much UML extensions as possible. The result was 23 UML extensions. Twenty extensions were categorized as lightweight and only three were heavyweight extensions. The second stage was studying those extensions in terms of domain, purpose of extension, type of extension and extended UML diagram. In this research, extensions that are made to three UML diagrams were only considered in the integration process. These UML diagrams are; class, sequence and use case diagrams. The third stage was the integration process. The process was applied to two types of extensions; the first type addresses the UML extensions that provide graphical symbols only, and the second type goes beyond

the graphical representations in the UML diagrams and deals with the proposed modeling elements that add to the meta-models. In this research, a diagram editor tool, called Dia, was used to create UML diagram notations and meta-model modifications. The last stage was developing case studies to validate the iUML. The case studies were inspired by examples and case studies from the literature. The result was 6 case studies. The case studies encompassed domains like data warehouse and security, object-oriented and mobile distributed system, quality of service and component, agent and adaptive web application and so much more.

## **7.1 Contribution**

The contribution of this research is:

- Developed iUML framework:

A framework for integrating UML extension was introduced. By following iUML integration processes, one can add any new graphical symbols and meta-model element extensions to be part of iUML.

- Developed iUML:

The proposed iUML is capable of modeling any problem domain since it has enough number of integrated extensions to cover current and possibly future domains.

## **7.2 Threats to Validity**

The validity of iUML is threatened by two main threats; the validity of the available extensions and the reliability of the integration process. In the former threat, the validity

of the available extensions, each UML extension must provide a rich and robust extension to UML. Having incorrect extensions would halt the integration process in its early stages. For example, having invalid modeling elements (stereotypes or tagged values) excludes the extension from the set of extensions to be integrated since the modeling elements cannot be added to iUML meta-model. The selection process of UML extensions must follow a systematic procedure that yields a reliable set of extensions. In this research, we assumed the validity of the available extensions, and thus, no validation of the available extensions was done from our side.

In the second threat, reliability of the integration process, applying the integration process in its two types; graphical and meta-model, must be also done carefully, especially the integration of meta-model elements since the integrated elements constitute the infrastructure of iUML. The steps of the integration process must be revised repeatedly to make sure whether the placement of the meta-model elements or even the integration of these elements was applied correctly. Failing to do so would ultimately produce an invalid model. In this research, the proposed integration process worked well while integrating the available extensions in the literature, however, new extensions may require the process to be modified.

### **7.3 Future work**

Additional research directions that need to be explored in future work include the following:

- Consider UML diagrams other than class, sequence and use case diagrams to cover more areas in the software development systems.

- Integrate iUML with available IDEs like Rational Rose or Enterprise Architect.
- Automate correctness and verification tasks (e.g., conflict analysis, etc.).

## References

- [1] Grady Booch , James Rumbaugh and Ivar Jacobson, *The Unified Modeling Language User Guide*, Second Edition ed.: Addison Wesley Professional, 2005.
- [2] *Rational Software Corporation Website*, Available: <http://www-01.ibm.com/software/rational/>.
- [3] *OMG, Unified Modeling Language*, Available: <http://www.uml.org/>.
- [4] Martin Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Third Edition ed.: Addison Wesley, 2003.
- [5] (2010). *OMG Version 1.0 of EXPRESS*, available <http://www.omg.org/spec/EXPRESS/1.0>.
- [6] (2007). *Behavior Trees*, Article: <http://aigamedev.com/open/article/bt-overview/>.
- [7] Grady Booch, *Object-oriented Analysis and Design with Applications*, 2nd ed.: Redwood City: Benjamin Cummings, 1993.
- [8] *OMT Introduction*, Available: <http://www.smartdraw.com/resources/tutorials/rumbaugh-omt-diagrams/#/resources/tutorials/Introduction-to-OMT>.
- [9] Ivar Jacobson, Magnus Christerson, Patrik Jonsson and Gunnar Overgaard, *Object-Oriented Software Engineering: A Use Case Driven Approach*: Addison-Wesley, 1992.
- [10] (2009). *OMG, UML Superstructure Specification Version 2.2*. Available: <http://www.omg.org/spec/UML/2.2/>.
- [11] Lidia Fuentes-Fernandez and Antonio Vallecillo-Moreno "An Introduction to UML Profiles," *The European Journal for the Informatics Professional*, vol. V, No. 2, 2004.
- [12] Eduardo Fernandez-Medinaa, Juan Trujillo, Rodolfo Villarroel and Mario Piattini, "Developing secure data warehouses with a UML extension," *Information Systems*, vol. 32, pp. 826–856, 2007.
- [13] C. Simons and G. Wirtz, "Modeling context in mobile distributed systems with the UML," *Journal of Visual Languages and Computing*, vol. 18, pp. 420 - 239, 2007.
- [14] Xiao-cong Zhou, Chang Liu, Yan-tao Niu and Tai-zong Lai, "Towards a Framework of Aspect-Oriented Modeling with UML," in *International Symposium on Computer Science and Computational Technology*, 2008.
- [15] Jan Hendrik Hausmann, Reiko Heckel and Stefan Sauer, "Towards Dynamic Meta Modeling of UML Extensions: An Extensible Semantics for UML Sequence Diagrams," in *International Symposium on Human-Centric Computing Languages and Environments (HCC 2001)*, Stresa, Italy, 2001.
- [16] Chen Fei and Chen Yan, "Spaeris: A Multi-Agent System Specified by Agent UML," in *International Seminar on Future Information Technology and Management Engineering*, 2008.
- [17] Raoudha Ben Djemaa, Ikram Amous and Abdelmajid Ben Hamadou, "WA-UML: Towards a UML extension for modelling Adaptive Web Applications," presented at the IEEE International Symposium on Web Site Evolution, 2006.
- [18] Manar El-Kady, Reem Bahgat and Aly Fahmy, "A UML Heavyweight Extension for MAS Modeling," in *International Conference on Quality Software*, 2008.

- [19] *OMG, OCL*, Available: <http://www.omg.org/spec/OCL/>.
- [20] Jordi Cabot, "Representing Temporal Information in UML.," presented at the LNCS, 2003.
- [21] Adam Przybyłek, "Separation of crosscutting concerns at the design level: An extension to the UML metamodel," in *International Multiconference on Computer Science and Information Technology*, Wisla, Poland, 2008.
- [22] Marcus Fontoura, Wolfgang Pree and Bernhard Rumpe, "UML-F: A Modeling Language for Object-Oriented Frameworks," in *European Conference on Object-Oriented Programming*, London, UK, 2000.
- [23] Wan-Seob Byeon, Bo Wang, Sa-Kyun Jeong and Ok-Bae Chang, "Extension and Implementation of Iconic Stereotype for GNSS Application in the UML Class Diagram," in *International Conference on Cyberworlds*, 2004.
- [24] Matthew J. Peterson, John B. Bowles and Caroline M. Eastman, "UMLpac: An Approach for Integrating Security into UML Class Design," 2006.
- [25] Jing Dong, "UML Extensions for Design Pattern Compositions," *Journal of Object Technology*, vol. 1, 2002.
- [26] Yasunobu Sanada and Rolf Adams, "Representing Design Patterns and Frameworks in UML: Towards a Comprehensive Approach," *Journal of Object Technology*, vol. 1, 2002.
- [27] Azrul Hazri Jantan, Putra Sumari and Shahida Sulaiman, "ComHDM: Extending UML Profiles for Modeling Complex Web Hypermedia Applications," in *International Conference on Advanced Computer Theory and Engineering*, 2008.
- [28] Sajjad Mahmood and Richard Lai, "RE-UML: An extension to UML for specifying Component-Based Software System," in *Australian Software Engineering Conference*, Australia, 2009.
- [29] Zohreh Sharafi, Parisa Mirshams, Abdelwahab Hamou-Lhadj and Constantinos Constantinides, "Extending the UML metamodel to provide support for crosscutting concerns," in *ACIS International Conference on Software Engineering Research, Management and Applications*, 2010.
- [30] (2007-12-02). *XML metadata interchange (XMI)*. Available: <http://www.omg.org/cgi-bin/doc?formal/2007-12-02>
- [31] St. Sauer and G. Engels, "Extending UML for modeling of multimedia applications," presented at the IEEE Symposium on Visual Languages (VL'99), 1999.
- [32] Vittorio Cortellessa and Antonio Pompei, "Towards a UML profile for QoS: a contribution in the reliability domain," in *Proceedings of the 4th international workshop on Software and performance*, 2004.
- [33] Ying Dong, Mingshu Li and Qing Wang, "A UML Extension of Distributed System," in *First International Conference on Machine Learning and Cybernetics*, Beijing, China, 2002.
- [34] José Raul Romero, José M. Troya and Antonio Vallecillo, "Modeling ODP Computational Specifications using UML," *The Computer Journal*, vol. 51, 2007.
- [35] C. Andre, F. Mallet and M-A. Peraldi-Frati, "Multiform Time in UML for Real-time Embedded Applications," in *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Washington, DC, USA, 2007.

- [36] Istvan Majzik, Gergely Pinter and Peter Tamas Kovacs, "UML Based Design of Time Triggered Systems," in *IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, 2004.
- [37] S. Pillana and T. Fahringer, "Modeling Parallel Applications with UML," in *International Conference on Parallel and Distributed Computing Systems*, 2002.
- [38] Xuandong Li and Johan Lilius, "Timing Analysis of UML Sequence Diagrams," Turku Centre for Computer Science 1999.
- [39] *Dia software*, Available: <http://dia-installer.de/>.



# VITA

## Personal Information

- **Name:** Nasser Salman Abed Khashan
- **Born:** 5/13/1985, Riyadh, Saudi Arabia
- **Nationality:** Jordanian

## Education

- B.S., Computer Information Systems, 2007, Applied Science University, Amman, Jordan.
- M.S., Computer Science, 2012, King Fahd University of Petroleum and Minerals, Dahrhan, Saudi Arabia.

## Research Interests

I have broad interests in software engineering topics, particularly in software requirements engineering, software design, software validation and verification.

## Contact Information

- **Present Address:** Corniche Street, Khobar, Saudi Arabia.
- **Permanent Address:** Al-Malaz, Al-Nahda Street, Riyadh, Saudi Arabia.
- **E-mail Address:** [khashan@live.com](mailto:khashan@live.com)
- **Cell Phone Number:** +966-569-843-813 / +966-555-203-405
- **Telephone Number:** +966-1448-7287 / +966-1472-4210