Verification and Validation of UML/OCL Object Componenets Models

Bhutto, Arifa

Publication date:
2018

Document Version
Publisher's PDF, also known as Version of record

Link to publication from Aalborg University

Citation for published version (APA):
Bhutto, A. (2018). *Verification and Validation of UML/OCL Object Componenets Models*. Aalborg Universitetsforlag. Ph.d.-serien for Det Ingeniør- og Naturvidenskabelige Fakultet, Aalborg Universitet

VERIFICATION AND VALIDATION OF UML/OCL OBJECT COMPONENT MODEL

BY
ARIFA BHUTTO

DISSERTATION SUBMITTED 2018

AALBORG UNIVERSITY
DENMARK

VERIFICATION AND VALIDATION OF UML/OCL OBJECT COMPONENT MODEL

by

Arifa Bhutto

AALBORG UNIVERSITY
DENMARK

Dissertation Submitted 2018

.

Dissertation submitted:	September 2018
PhD supervisor:	Associate Prof. D.M. Akber Hussain, Aalborg University, Denmark
PhD committee:	Associate Professor Daniel Ortiz-Arroyo, Aalborg University
	Prof. Dr. Engr. Syed Hyder Abbas Musavi Indus University
	Associate Professor Dr. Sadiq Ali Khan, University of Karachi Pakistan (UoK)
PhD Series:	Faculty of Engineering and Science, Aalborg University
Department:	Department of Energy Technology

ISSN (online): 2446-1636
ISBN (online): 978-87-7210-194-1

Published by:
Aalborg University Press
Langagervej 2
DK – 9220 Aalborg Ø
Phone: +45 99407140
aauf@forlag.aau.dk
forlag.aau.dk

Printed in Denmark by Rosendahls, 2018

CV

Work Address:
Institute of Information and Communication Technology
University of Sindh, Jamshoro, Pakistan.
Cell No: +92-346-8964209
Email: arifa.bhutto@usindh.edu.pk

Work Experience:

2006 to Present:	Assistant Professor, IICT, University of Sindh, Jamshoro, Pakistan.

Research Interest:

Currently doing PhD studies at Department of Energy and Technology, Aalborg University, Denmark.
Software Engineering is my field of interest, major focused on Verification and Validation of Software development models. My research topic is Verification and Validation of UML/OCL Object Components models.

ENGLISH SUMMARY

Modern software application development is a complex and difficult process. In development of applications, specification and verification are the key components; both specification and verification are part of the development process for any project. Various techniques are used for the components' development; however, in general there are well-established methods available for specification based on mathematical theories. These methods are used and practiced for every step involved in the development of a software project. Modern systems are hybrid; meaning they are composed of software and hardware. The correct functioning of any hardware is dependent on the software running on it.

Traditional design specification is illustrated using UML, a graphical notation, contacting numerous types of diagrams that enable modeling of different aspects of the design related challenges. The aim of our research is to use existing model checking tools and techniques to analyze and verify the properties of the design system. These system specifications are designed using the UML object components diagrams, integrated with the OCL constraints, which enables a more semantically specification focusing on structural and behavioral properties of the system so that the object components' concepts are accompanied with an application to an industrial case study.

The thesis is a combination of two parts: Part I defines the Introduction of the problem, state of art methods including case study, and Part II appendix consisting of publications related to the topic "Verification and validation of UML/OCL object components' models".

DANSK RESUME

Moderne software applikationsudvikling er en kompleks og vanskelig proces. Ved udvikling af applikationer er specifikation og verifikation nøglekomponenterne, både specifikation og verifikation er en del af udviklingsprocessen for ethvert projekt. Forskellige teknikker anvendes til komponentudviklingen; Men generelt er der veletablerede metoder til rådighed til specifikation, der er baseret på matematiske teorier. Disse metoder anvendes / praktiseres for hvert trin involveret i udviklingen af et software projekt. Moderne systemer er hybrid betyder, at de består af software og hardware. Korrekt funktion af enhver hardware er afhængig af den software, der kører på den.

Traditionel designspecifikation er illustreret ved hjælp af UML, en grafisk notation, der kontakter flere typer diagrammer, der gør det muligt at modellere forskellige aspekter af de designrelaterede udfordringer. Vores forskningsmål bruger eksisterende modelkontrolværktøjer og teknikker til at analysere og verificere designsystemets egenskaber. Disse systemspecifikationer og design ved hjælp af UML-objektkomponentdiagrammerne, der er integreret med OCL-begrænsningerne, muliggør en mere semantisk specifikation med fokus på systemets strukturelle og adfærdsmæssige egenskaber, objektkomponenter koncepter ledsages af en ansøgning til en industriel casestudie.

Afhandlingen er en kombination af to dele: Del I definerer indledningen, problemet, tilstanden og metoderne, herunder casestudier og bilag, der indeholder publikationer relateret til emneverifikation og validering af UML / OCL objektkomponenter.

ACKNOWLEDGEMENTS

Pursuing a PhD is a painful and enjoyable experience. It is just like a sitting in the roller coaster with full of excitement, energy, fear, trust to not step down. When I found myself on top to view the beautiful scenery, I realized that it was teams work without that this cannot be achievable.

First and foremost, I deeply grateful to my supervisor Assoc. Prof. Dil Muhammad Akbar Hussain who invited me as PhD student Energy and Technology, Aalborg University, Denmark. I always respect his research enthusiasm and micro-viewing ability; he has lots of patience and always give meaningful guidelines during the supervision.
He is always encouraging, supportive and believes on freedom and trust, which create a bond between us. Specially he helpful in all circumstances when I was sick he was available all the time he supports me morally a lot. He believes in me for completing my PhD studies.

I would like to thanks, to all my colleague, John Kim Pedersen and Jens Bo holm Nielson and supporting staff, Mette Skjarbek, Tina Larsen, Corina they have been very kind, cooperative and supportive during my PhD studies.

However, I cannot share this great moment with my parents (Mother and Father). Indeed, it was one of their dreams, which come true, Baba; it was your dream that I must do PhD. I started on this mission to fulfill your dream. Nevertheless, destiny had other ways. I wish you were here with me to see that I have completed my thesis. I know that wherever you are you will be very proud and happy that I finished what I started. You were the driving force behind me, instilling courage and determination in all my difficult moments. Ammi, I know my PhD means to you, I miss you on this special moment, I know you both look at me and I received your blessings wherever you are.

I owe my deepest gratitude to my husband Shahid Hussain Soofi, thank you for being understanding and cooperative specially whenever I was not able to spend quality time with the family, especially when I was sick and writing my thesis work. I am extremely grateful to my kids Shah Fahad Hussain and Annabel Saeeda both were so well behaved and cooperative and never fussed about me to not being able spend time with them.

I would like to thank Department of Energy and Technology, Aalborg University Denmark awarding a scholarship as a Tuition fees wavier for my PhD studies and University of Sindh, Jamshoro for awarding the stipend scholarship to complete my PhD.

Last but not the least I thank to my colleagues and friends Dr. Kamran Taj Pathan, Imran Ujan and Mehran, who give their precious time and support and encouraged me to finalize my PhD thesis in my hard time.

Thanks all.

Arifa Bhutto

Aalborg University, 2018

Thesis Details

Title:
Verification and Validation of UML/OCL Object Component Model

PhD Student:
Arifa Bhutto

Supervisor:
Associate Professor, Dil Muhammad Akber Hussain

The main body of this thesis consists of following papers:

A. Arifa Bhutto, D.M. Akbar. "Formal Verification of UML Profile" in Australian Journal of Basic and Applied Sciences, 5(6): 1594-1598, 2011.

B. Arifa Bhutto, D. M. Akber Hussain. Imran Anwar Ujan, Mehran Syed," Formal Approach for UML components based Development Profile" University of Sindh, Journal of Information and Communication Technology, 2(2): 125-129, 2018.

C. Arifa Bhutto, D.M. Akbar Hussain, "Validate UML model and OCL Expressions using USE Tool" Pertanika J.Sci.& Technology, 26(3):1465-1480,2018.

http://www.pertanika.upm.edu.my/Pertanika%20PAPERS/JST%20Vol.%2026%20(3)%20Jul.%202018/39%20JST(S)-0444-2018-3rdProof.pdf

D. Arifa Bhutto, D.M. Akber Hussain "An Android Based Cooperative Knowledge Acquiring Application" Mehran University Research Journal of Engineering and Technology, 37(3): 453-460, July 2018
DOI: 10.22581/muet1982
publications.muet.edu.pk/index.php/muetrj/article/download/486/211/

E. Sobia Mansoor, Arifa Bhutto**, "**Improvement of Students Abilities for Quality of Software Through Personal Software Process" Abilities for Quality of Software Through Personal Software Process", International Conference on Innovation in Electrical Engineering and Computational Technologies (ICIEECT), 2017, IEEE
DOI: 1109/ICIEECT/2017.7916550

Copyright:

TABLE OF CONTENTS

LIST OF THE FIGURES

LIST OF TABLE

CHAPTER 1. INTRODUCTION

This chapter highlighted the goals and objectives of the research and summarized the existing literature available related to the verification and validation of UML/OCL object components model.

The main findings of this chapter is based on Paper [A].

1.1. MOTIVATION

Our daily routines are guided and guarded by automaticity of systems, which are becoming inherently more and more complex and incorporates constantly in our environment.

The span of the science and the field of technological knowledge has long been too vast for most people to comprehend at a level needed for satisfying demands. Engineers must today be highly specialized and educated in order to master the relevant skills and the numbers of special engineering branches are almost as vast as the industrial sections where engineering is needed.

Software Engineering Development is one of the fields having very complex framework, because development of software is based on right way of integration of all components in one application that control and accurately run the system. In such scenario, designing and specification identifying of the software applications is very critical and difficult process. In software development, Unified Modeling Language (UML) have been used successfully. UML models represent different level of system development structures. The UML models are based on the "Object- Oriented" methodology for creating graphically notations of the systems [1]. UML has been created for several domains including software system engineering, component development specification and software process modeling, all above modeling techniques are based on the model –driven development process [2],[59].

However using UML some problems are identified in design techniques, like separation of correctness, accuracy and time parameters [3]. In this regard, UML models are encrypted with the Object Constraints Language (OCL). This type of specification now-a-days exists in the form of Components Based Software Development (CBD), which is, based on the Object-Oriented software development design methodologies (OOD) [4]. Most of the existing OOD are based on formal methods such as UML/OCL [3], [4], [5].

We also look for the structural and behavioral part of the designed models, by applying the constraints to check the model correctness, consistency and accuracy [10]. However, for the verification and validation, a process is required to reason rigorously on formal specification, verification of design patterns, their applications, compositions and evolutions [6], [7].

Our research methodology is to analyze UML/OCL analytical and theoretical based models in order to elicit sound semantic foundations for object components system modeling. We then plan to proceed to a constructive phase, using the foundation to verify concrete examples in a number of experiments in the form of a case study which is presented in chapter 4.

1.2. BACKGROUND AND RELATED WORK

Mostly research is going on verification and validation on UML/OCL class object models which is available worldwide. The author first time introduced the visualization modeling methodology by B method, but because of non-availability of semantics in B method in research community it is not much popular [8], [58], although author has the precision to support formal verification of models using the animation. However, lack of semantics support many practitioners received B notation as an actions supported by the constraint parameter for the UML models, which look like UML models are translated into the B [8],[9],[42]. However, UML-B profile [8] provides supports to UML model interim for refinement and visualization of the Object behavioral models. The most and popular use of UML -B [9] is used for the industrial applications that have found very concrete results [42],[47].

A somehow similar idea has been proposed by authors in UML to CSP [3],[11]. UML to CSP tool is used for the formal methodology in verification of UML to OCL models. “Given a UML class diagram annotated with OCL constraints, UML to CSP” [3],[11],[12] tool checks automatically system models correctness properties, for example strong, weak and satisfiability of the models by checking redundant constraints on the UML to CSP which basically is formation of constraints programming paradigm underlying the constraints solver on Ecl^{i}ps^{e} environment for the verification[32],[33]. As a software developer, Eclipse environment is not easy to use for most researcher’s engineering development, hence complexity of the system design researcher find difficulty in using this approach [13],[14]

The most popular and well-defined methods are used in constraints programming, but we know that the constraints programming can only be utilized if we want to verify or validate object class model of UML. The authors define the way out to declare full class model in specification language and then apply the constraints on it. Usually all researchers do manually in all tools and methods. Over all up to now, compliance of the diagram with respect to correctness properties of the models are verified [18],[16],[17],[19],[48].

This is also the case when authors describing in [2], [6], [15], [16], [18] the various formal verification methods like First Order Logic methodology[2], which is itself is very much expensive way of describing the model verification of UML class diagrams annotated with OCL constraints, However, first-order logic (FOL) itself is more mathematically reasoning mechanism [20],[21]. In general, OCL is more expressive than FOL. Therefore, to avoid ambiguity, we need to define limit in the UML-OCL diagrams or we keep more emphasis to adopt more graphical form of visualization of models at run time [20],[21],[22],[23].

Table 1.2 Summary of methods using for V&V of UML models

UML Notations	**Formal methods for Verification and validation process**	**Analysis of the methods.**
Class Diagram	Object Oriented Modeling Techniques.	The authors provide UML models can access graphical view of models and communication of various models using the animation and verification.[3],[28].
Class Diagram	OCL constraints	This method checks automatically various properties like correctness, strong and weak, according to the system models, but method lacks redundant constraints checking [29],[30],[31].
Class Diagram	Constraints Programming	Using this methods authors define approach of Model Driven Development where the UML models are the key models of the design and development framework. This method having an automatic uses of OCL constraints programming to check the UML class diagram annotated with OCL Constraints [32], [33],[34].

Class Diagram	Communicating Extended Timed Automata (CETA), verification tools.	The authors present a technical tool for validating UML models and verifying through the simulation [59]. The CETA verification methods check the system properties and operations, which are part of the inheritance and polymorphism including the state machine models having the well-defined semantic profile for communication sequences and concurrency checking among the different objects. In the CETA, authors define the UML profile representation of timing constraints [35],[36],[37],[39].
State charts	HTA hierarchical timed automata	In this tool authors define the formal logical language which included the real time properties with the formal representation by using TCTL. The Timed computational Temporal Logic is unambiguous but it validates and verifies the possible class diagram of the system.[38],[45],[46]
Sequence, State Machine , class and Package Diagrams	UML 2.0 and SysML	According to this method, authors define V&V based on formal verification and model checking of the desired system by the flow analysis of data and control constraints. Overall analysis is based on the abstraction level of interpretation [40],[41],[43],[44].

However, many authors believe that in software engineering, Model Driven Development is growing and helps the developer community to trust on such methods for the software design and specification level [26],[57] as they are never aware to find out specification and design errors until reached at the phases of development or implementation of the systems [25]. The formal reasoning is not used because until the implementation stage, it increases the cost of the development process. In Table 1.1 we described the currently adopted UML notations, formal methods tools and techniques [24], [27]. However, not all of above define the one complete method for the UML/OCL object components model graphical verification and validation process.

1.3. CHALLENGES AND CONTRIBUTION

The objective of this research study is to investigate the UML Object-Oriented and components-based design models and defines the specification and verification of object class model by semi-formal methods, which visually and graphically check the correctness, relationship and dependency of the models. The scientific challenges that we see in analyzing the Object Components-based development modeling applications are the following:

- An analysis and verification of the structural, behavioral properties of the UML/OCL Object Component methodology using model checking tools and techniques.
- To analyze, verify and suggest compensation mechanisms for some concrete case study.
- Study and learn the state of the art techniques in the area of specification and verification of UML models like objects, class and components model, so that we can apply and utilize the relevant knowledge.

The above-mentioned objectives are the key points towards the scientific contribution in the area of my research. I am confident that this will provide further enhancement towards knowledge contribution and it will be very beneficial for those who wish to do research/development in this area either in this university or elsewhere.

However, there are still numerous challenges regarding how to integrate UML/OCL with formal specification language like Z or object- Z that are directly connected and that generate UML models.

1.4. THESIS OUTLINE

The organization of this thesis is in two parts. Part – I of the thesis is from research work and background of the research, Part-II is produced publications, which is part of PhD research work.

PART –I

The organization of Part-I follows:

Chapter 1. Representation of introduction, background literature and contributions.

Chapter 2. Illustration of the applied theories and notations used for research work.

Chapter 3. Development of Case Study.

Chapter 4. Submission of Conclusion and Future Work.

PART- II

1.5. LIST OF PUBLICATIONS

A list of publications is given below that is included in thesis Part- II:

A. Arifa Bhutto, D.M. Akbar. "Formal Verification of UML Profile"

in Australian Journal of Basic and Applied Sciences, 5(6): 1594-1598, 2011.

B. Arifa Bhutto, D. M. Akber Hussain. Imran Anwar Ujan, Mehran Syed," Formal Approach for UML components based Development Profile" University of Sindh, Journal of Information and Communication Technology, 2(2): 125-129, 2018.

C. Arifa Bhutto, D.M. Akbar Hussain, "Validate UML model and OCL Expressions using USE Tool" Pertanika J.Sci.& Technology, 26(3):1465-1480,2018

http://www.pertanika.upm.edu.my/Pertanika%20PAPERS/JST%20Vol.%2026%20(3)%20Jul.%202018/39%20JST(S)-0444-2018-3rdProof.pdf

D. Arifa Bhutto, D.M. Akber Hussain "An Android Based Cooperative Knowledge Acquiring Application" Mehran University Research Journal of Engineering and Technology, 37(3): 453-460, July 2018
DOI: 10.22581/muet1982
publications.muet.edu.pk/index.php/muetrj/article/download/486/211/

E. Sobia Mansoor, Arifa Bhutto**, “**Improvement of Students Abilities for Quality of Software Through Personal Software Process” Abilities for Quality of Software Through Personal Software Process”, International Conference on Innovation in Electrical Engineering and Computational Technologies (ICIEECT), 2017, IEEE

DOI: 1109/ICIEECT/2017.7916550

In addition to the main papers included in the thesis work, the following publications have also been made:

1. I. A. Ujan, Arifa Bhutto, “An Overview of Health Information System” Published in 11th International Conference on Statistical Sciences at Islamia College Peshawar on March 5th to 8th 2018.

2. I. A. Ujan, A. Bhutto, M. M. Rind, M. A. Shamimi “Acceptance of HMIS by Healthcare Professionals of Private Sector Hospitals “ Sindh University Research Journal (Science Series) Vol. 48 (4D) 165-168 (2016)

3. Arifa Bhutto, Mehran Shah, Dr. Kamran Taj “Online Doctor Appointment System” http://ibt.edu.pk/ibt/jurnals/1_ibt.biztek.(2018).

CHAPTER 2. APPLIED THEORIES AND NOTATIONS

In this chapter theories and notations are defined which serve the purpose of research. Setting on notation is a matter of preference and understanding more than anything else. To make the message clear it is important that the chosen notation conventionally can express what is required and that it is well established so that other parties will be able to participate in the evaluation of the contribution.

The most widely used notation in the software engineering industry is UML [49]. It is the main contribution in designing the system structure by the UML notations. Our research is focused on how we can verify and validate the UML integrated with the OCL constraints to verify and validate object components models at the design level. As UML is the modeling notation and design model diagrams and OCL is the constraints language, which applies constraints on the class diagram, but both are not able to verify and validate the model at the design level to check correctness, association and constraints applied on the models. For that reason we propose verification and validation of UML/OCL [49], [50] diagrams by UML Based Specification Environment (USE) [52]. Using USE tool, we verify and validate the UML/OCL models at design level [53].

The main findings of this chapter are based on Papers [A], [B] and [D].

2.1. UNIFIED MODELING LANGUAGE

Unified Modeling Language or UML [18] was initiated as the unification of three notations for designing of Object-Oriented software systems. In the early 1990s, James Ram Baugh and Grady Brooch, in each of their affiliation, worked on methods for supporting the development of object-oriented software, before they in 1994 joined the Force at Rational Software and so merged their methodologies and produced unified modeling Language and Rational Rose Unified Process. Since 1996, the standardization of UML has been organized by Object Management Group (OMG) which is a non-profit organization of researchers interested in the development of UML and other projects.

UML 2.2 is the most recently published version of UML, which provides thirteen different kinds of diagrams that are used to model structural, behavioral and interaction aspects of software systems as defined in Fig 2.1.

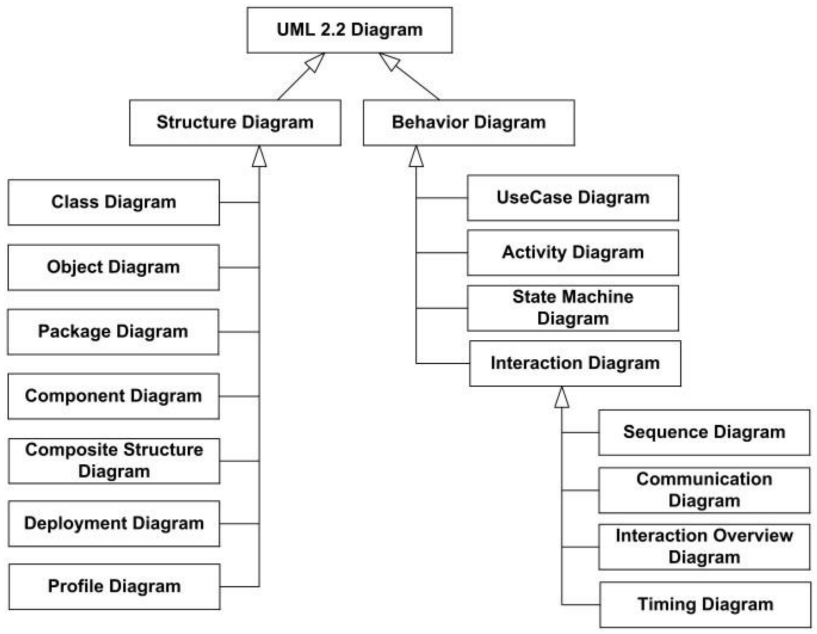


Figure 2.1 UML 2.2 Diagrams Overview. Source https://www.uml-diagrams.org/uml-22diagrams.html

Structural Diagrams: Class Diagram, Object Diagram, Component Diagram, Composition Structure Diagram, Deployment Diagram, Profile Diagram

Behavioral Diagrams: Use Case Diagram, Activity Diagram, State Machine Diagram

Interaction Diagrams: Sequence Diagram, Communication Diagram, Interaction Overview Diagram, Timing Diagram

The UML diagrams in combination are used to model different views of a software system on a level of richness that is beyond the scope of this work. In section 2.1.1 through 2.1.3 Class Diagram, Object Components Diagram and Sequence Diagrams will be presented, as this subset of the UML language provides a sufficient syntax for reasoning about verification and validation of UML models.

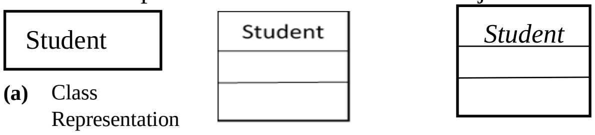
2.1.1. CLASS DIAGRAM FOR MODELING STRUCTURE

The Class Diagram is used to model relationships between classes of objects, i.e. the structural design of the system. In Class Diagram, therefore, we can represent it as a graph. Using the graph, nodes show the classes, and two types of edges that represent the relationships are called association and dependencies.

Class

A class is a set of objects that has the same semantics, attributes, operations and constraints.

The attributes of a class relate instances of the class to values of the attributes types. Attributes may represent a navigable end of a binary association, which will be described further. Operation of an object manipulates attributes, which might cause the further operation to call to other such objects.



(a) Class Representation n with class

(b) Class representation with name and empty compartments.

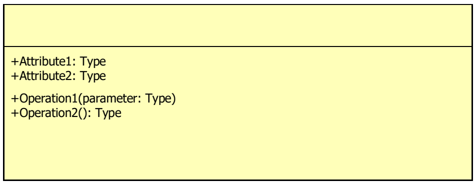
(c) Abstract Class representation with name and

Figure 2.1.1.1**:** The class Student represented as (a) a rectangle with the class name, (b) a rectangle with the class name and two empty compartments and (c) as an abstract class rectangle with the class name in italic and two empty compartments.

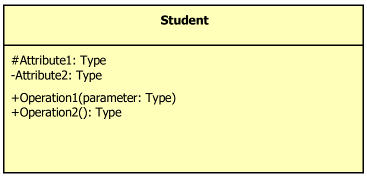
A class is illustrated using a rectangle that is optionally divided horizontally. If the class is illustrated as a simple rectangle, this rectangle contains the name of the class, as shown in Figure. 2.1.1.1(a). If the rectangle is subdivided, as it usually happens because the rectangle contains three compartments as shown in Figure 2.1.1.1 (b), the more compartments can be used. The top compartments specify the class name. If the class name is written using an *italic* font, as in Figure 2.1.1.1(c), the class is abstract. That class abstract means that no object instances of the class is created.

The two additional compartments illustrated in Figure 2.1.1.2 (a) and 2.1.1.2(b) are used to make more detailed specifications of the class properties. The middle compartment is used to specify class attributes and the bottom compartment is used to specify which operations the class offers. The level of the detail is illustrated in Figure 2.1.1.2(a), where attributes and operations are specified in the class description and are called the *design* level.

At the implementation level, shown in Figure 2.1.1.2(b), attributes and operation visibility is included. The visibility of attributes and operations is stated by prefixing the name, usually with:



(a) Class With Attributes



(b) Class With attributes, Operations and Visualization of Symbols.

Figure 2.1.1.2: The class student illustrated with (a) design level information on attributes and operations and (b) implementation level detail including visibility

+ for *public* element (object, attributes, operations etc.) that are visible / accessible for object that can access the namespaces that the public element belongs to.

# for *protected* element that are visible to objects that have a generalization relation to the namespaces that the protected element belongs to.

_ for *Private* element that are only visible inside the namespace it belongs to.

~ for *package* element that are visible to objects of the same package that its namespaces belong to.

2.1.2. ASSOCIATIONS AND DEPENDENCIES

In UML four different types of relations are defined: aggregation, association, generalization and dependency. However, the relations are represented as shown in Figure 2.1.2.1.

Association relation: An association relation reflects that objects are aware of the existence of each other and are aware of the association that exists between them. Thus links constitute the association and it is only valid as long as both objects agree on it unless one object ceases to exist, the association or link is naturally discontinued.

The UML specification allows two different ways to represent navigability between objects, using arrows to indicate the direction and crosses to indicate un-navigable association end.

The association relation is annotated with the symbols specified the multiplicity, i.e. the number of objects that are participating in the association. The Figure 2.1.2.2 shows association relationships with different annotations.

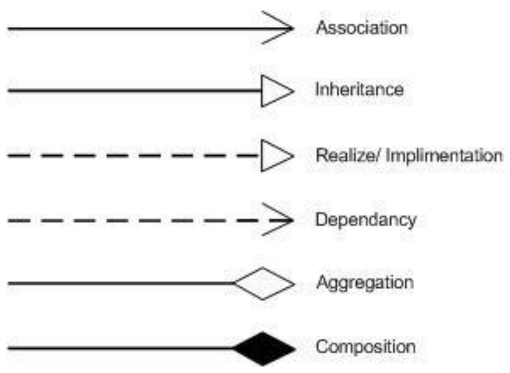


Figure 2.1.2.1: The types of relations between classes.

The association relation is an interpretation of the Class Diagram. The fact that the Class Diagram refers to relations between objects could be misleading as objects have a dynamic nature in them being instances of classes. A dynamic view on the associations is however problematic. When, e.g. an association is navigable in both directions and the multiplicity is one in both ends, the mutual awareness among the involved objects should be established instantaneously in order to fulfill the obligation of the association. Such instantaneous creation of association and objects is hard to achieve; thus Class Diagram provides a static view or a view when no object instantaneous are in progress.

Inheritance relation: The inheritance relation is used when classes have common attributes and/or operations. These common features are then generalized in parent super class, which may be abstract from which the child class inherits. It can extend or redefine the set of operation and attributes of the present class.

Aggregation relation: The aggregation relation is used where it is relevant to model a whole from its parts. In this case, the whole class relates to its parts. A special type of aggregation is composition where the square symbol is filled. The difference in the two aggregation types is multiplicity as the composition relation indicates that at least one object must be present. It is the responsibility of objects of the object of the composite class.

Dependency relation: The dependency relation is the weakest relation between classes in UML and can be considered an abstraction of associations. The dependency relation models a “client” and “supplier” relationship between classes. The semantics of the client part depends on the supplier, and if the attributes or operations of the supplier change the client may have to be changed too. As the constraints on the dependency relations are so weak that they could substitute all other relation in a design.

We define in Figure 2.1.2.2 as an example of classes, attributes, operations and relationships. Further detail of the class diagram and relationships we define using a case study in Chapter 3.

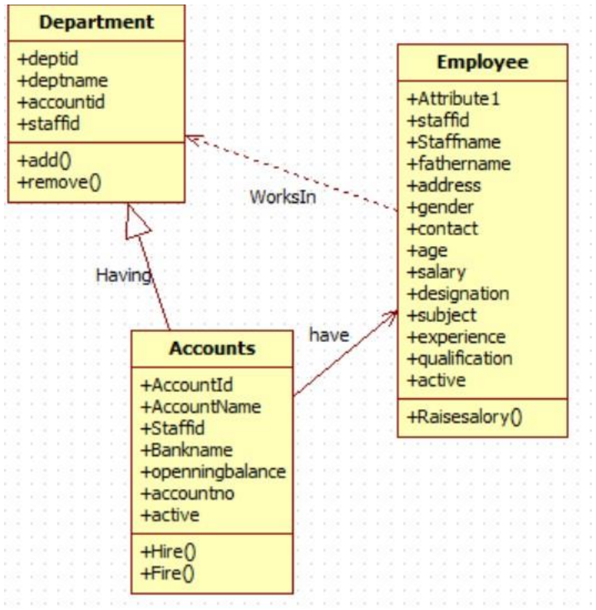


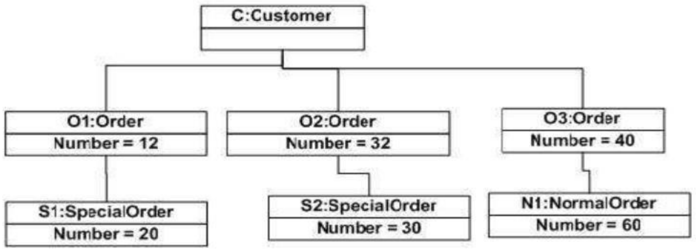
Figure 2.1.2.2 Example defines the classes structure, attributes, operations and association.

2.1.3. OBJECT DIAGRAM

Use of UML Object Diagram is dependent on the class diagram, in other way object diagram is the instance of the class diagram. An object diagram is another static view of the class diagram of the instances. An Example is shown in the Figure 2.1.3.1 which represents the notations in the UML object diagram.

The importance of the object diagram can be defined as:

- It shows the object relationship of the system.
- A static view of the system interactions is explored.
- To understand the object behavior in the system and relationship of the interaction as a practical form.



-

Figure 2.1.3.1 Example of Object Diagram- Represents the instance of the class diagram.

2.1.4. COMPONENT DIAGRAM

The UML Components Model represents the various software component that will be built and form one complete system. Component Model usually builds from the class model as we know that Components Model is part of the Object Oriented Methodology-based. Components model is the high level of the design of the system which shows the overall architecture of the system.

Components Notation: In UML 2.2, Components Diagram is represented with the notation of the rectangle box and in the corner two further boxes are drawn as shown in Figure 2.1.4.1 which represents the example of the UML Components Model notation.

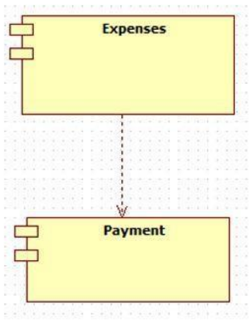


Figure 2.1.4.1 UML notation of Components and relationship

Components relationship Interface: Using UML 2.2 Components Diagram is connected through the interface in the form of relationship that is represented with the sender and receiver in the form of a circle and half circle as notation form. In practical, the component interface is defined by the class diagram. Using an example, we represent the same in Figure 2.1.4.2 which shows UML 2.2 Components Diagram notation whereas Figure 2.1.4.3 shows the internal interface of the components diagram.

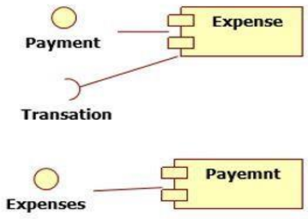


Figure 2.1.4.2 UML Components diagram with receiver and sender notations.

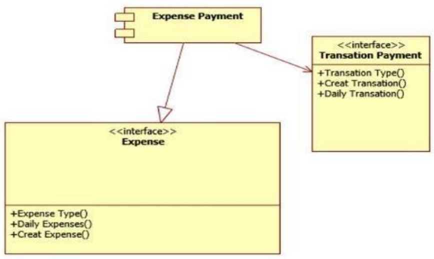


Figure 2.1.4.3 Components diagram represent the interface class.

2.1.5. SEQUENCE DIAGRAM FOR MODELING INTERACTION

The interaction between objects models can be modeled in UML by Sequence Diagram. Sequence Diagrams are used to model the sequence, i.e. the time ordering of events between the objects of a system. The objects of focus are shown as boxes at the top of the diagram, each box with a dashed line descending from it that illustrates a timeline. Events are drawn between the objects related to each other in time. A sample sequence diagram is given in Figure 2.1.5.1

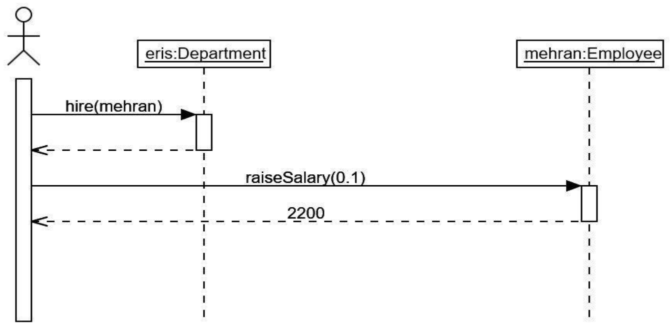


Figure 2.1.5.1 A sequence diagram illustrates department employee raise their salary sequence on the object time parameter.

2.2. OBJECT CONSTRAINTS LANGUAGE (OCL)

The Object Constraints Language (OCL) is a standard for the UML models' checking and validation. The OCL was first developed in 1995 inside IBM as an evolutionary language but later on it became an important factor for the Model-driven environment. Initially OCL was only used for the constraints language for model correctness parameters, but later on OCL constraints usually were applied on the class model, which were encrypted during the design of the structure of the class diagram [3].

OCL is a general-purpose formal language, which is currently a standard by the OMG group [15].

OCL is a specification language which is a declarative way of defining the rules on the UML models. OCL is integrated with many other applications but most popular is to define constraints on the UML class diagram in the form of Invariants, variants and pre-post conditions.

The important features are following:

- Initialization of the class
- Initialization of the class properties
- Using Invariants to check all conditions must have satisfied for the model.
- Pre- Postcondition

- Query Operations.

2.3. USE- UML BASED ENVIORNMENT

The **USE** tool is UML-based Environment for Specification [53]. It is a tool for UML models checking and execution. It applies the OCL constraints to design the model-driven development for software. USE tool assists developers to perform work as a mediator for a subset of UML models and OCL constraints. USE is a utilization process for case studies, teaching, development and analysis [5],[54],[55].

USE in textual form describes class diagrams and its attribute, operations, and association with its centric role of the system; it allows object diagrams to check the behavioral part of the UML models to apply the restrictions in the form of pre- and post-conditions.

In command shell of USE, a user can visualize the class diagram and its association as well as it generates the sequence diagram by applying the object data values in object forms. Model checker utilities always check the model consistency by applying the invariants restriction to validate the model. The USE tool checks that the Pre- and post-conditions are satisfied and analyzed in detail [55].

Model Structure: It validates the class attributes, relationships and structure by applying the variants and constraints.

Model Behavior: It verifies and validates the operations by applying the pre and post-conditions.

However, more practically adoptable knowledge is described in case study produced in chapter 3 which illustrates detailed framework of our research methodology.

CHAPTER 3. CASE STUDY

In this chapter, we illustrate the Verification and Validation of UML/OCL object Components model by presenting a case study. The case study represented the running example of the application of the organization in Hyderabad SYS builds of Employee Project Management System Application.

The main findings of this chapter are based on Paper [C].

In order to test our methodology, we define the following procedures for the solution of the problem.

Step 1. The design of the application described by the structure model in UML class, components model diagram and behavior of the system including the constraints by the OCL.

Step 2. Using the USE specification, we illustrated the UML classes, write the schema in textual format in any editor, that schema consists of attributes, operations, and associations in OCL textual language.

Step 3. Define constraints in OCL language in form of invariants, relationships and pre and post-conditions.

Step 3. Open the USE specification textual file and generate the graphical view of the class model, inherited with attributes, operations, relationships, variants, and invariants.

Step 4. Verify the model structure if it is correct to verify the behavioral properties of the system model by analyzing the object model that automatically further generates the sequence model in connection.

Step 5. The USE environment checks the UML class, sub-class, associations, operations, aggregations, composition.
Further USE model validates OCL constraints and verifies the constraints by applying the query to the class model.

The methodology of the research is represented in Figure 3.1 in detail.

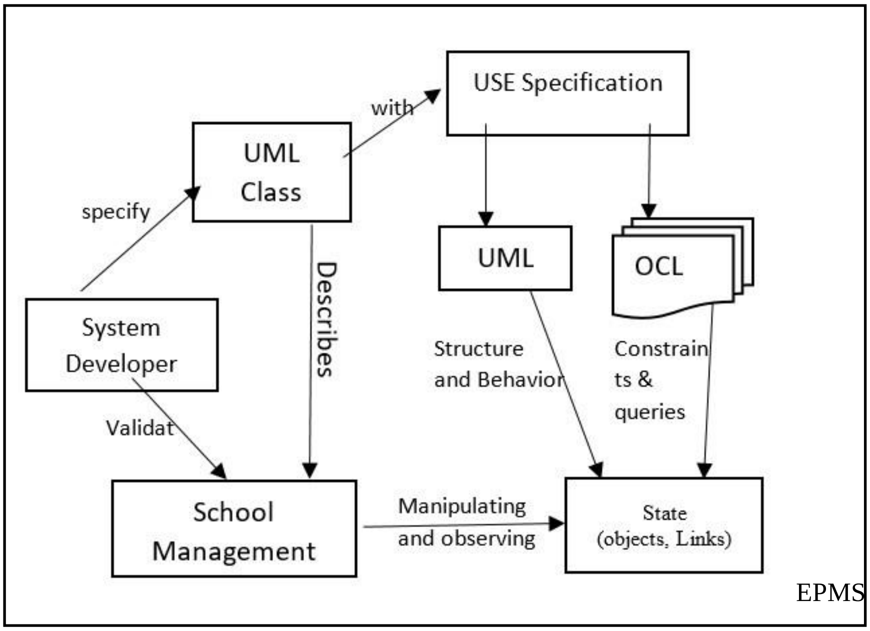


Figure 3.1.1 Verification and Validation of UML/OCL Object Components Model [C]

3.1. COMPONENTS MODEL

The Employee Project Management System is the application for the management of the Land projects of an organization, which is running locally at the Hyderabad. The SYS build is the building company they need to develop their Employee, project management system. In this regard, we found the following requirements of the system in the main module of the PMS application:

1. Admin has to manage the major three components:
i. Expenses ii. Set head of department iii. set head of components 2. In the PMS there is Payment mode which manages the:
i. Employees payment ii. General Payment 3.Employee are of different types:
i. staff ii. Labor
4. Opening the new project has a type:
i. Site Area ii. Building Project

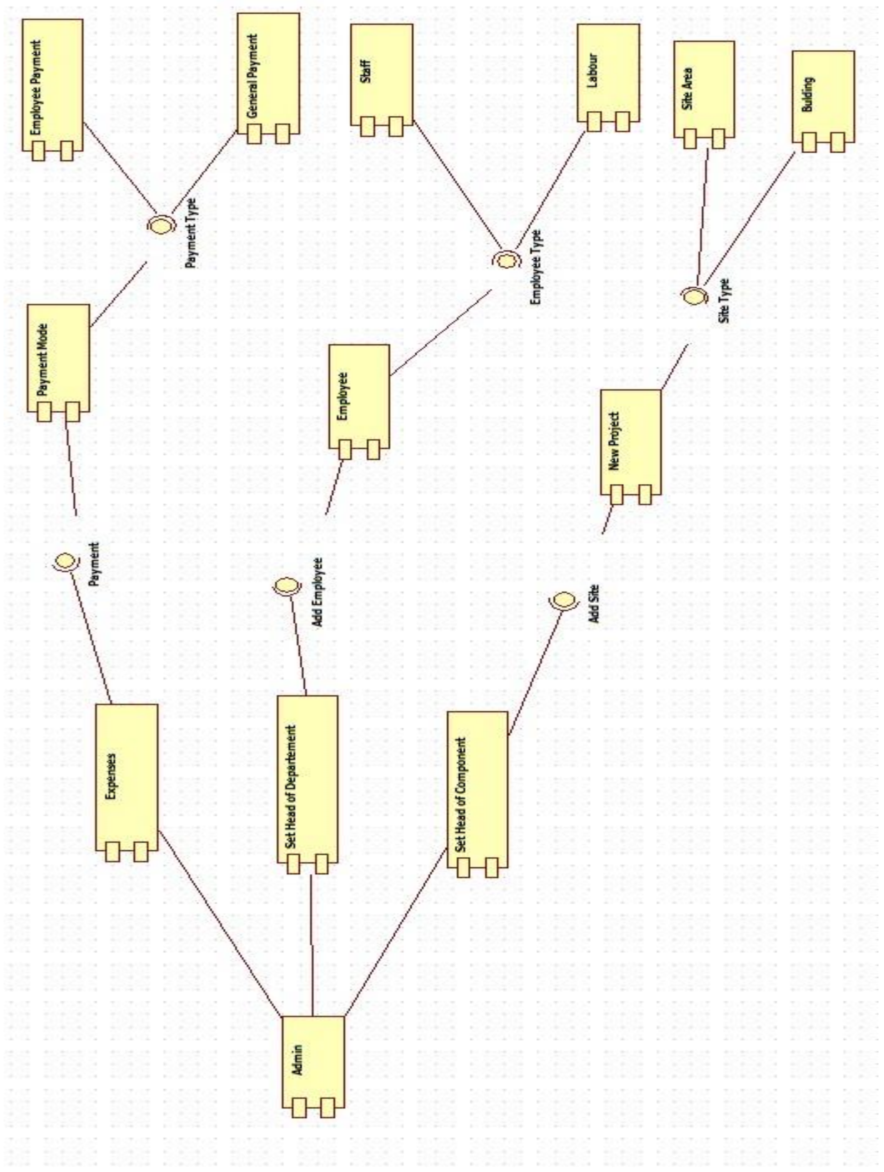


Figure 3.1.2 UML Components diagram View of PMS of SYSbuild

According to the above requirement, we first design the UML components Model diagram, which described the overall structural view of the system shown in Figure 3.1.2, and section 3.2 describes the class diagram including the interface diagram of the EPMS.

3.2. CLASS INTERFACE REPRESENTATION OF COMPONENTS

The UML2.2 Components diagram is the top level of the model and internal structural model is represented in the classes and interface of the receiver and sender classes. In Figure 3.2.1 the class interface of the PMS case study is further described in the form of USE textual specification by applying the constraints language by OCL.

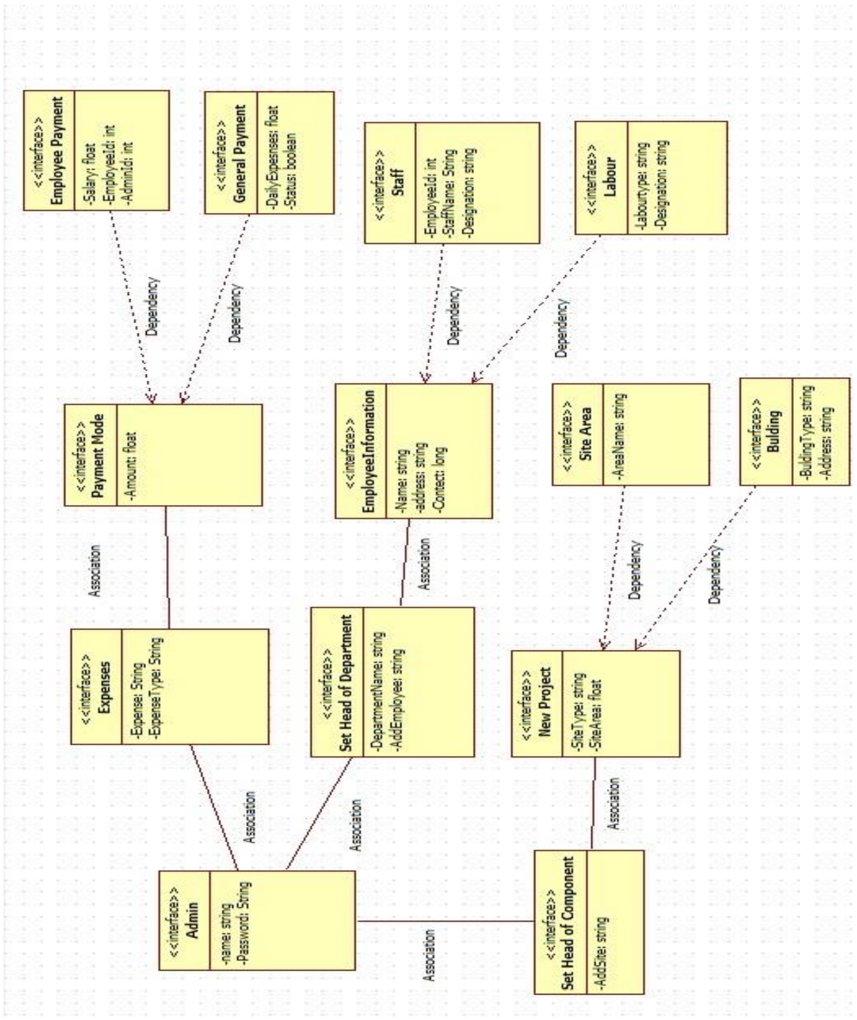


Figure 3.2.1 UML 2.2 Class dependency Diagram of PMS of SYSland

3.3. USE SPECIFICATION IN TEXTUAL FORM

In this section, we illustrated the USE specification in the textual representation of the classes that include the attributes, operations and associations, which are further integrated with the OCL invariants, pre-post conditions to enforce the rules checking and verification and validation of the operations applying on the system models.

In table 2, we define the PMS SYS Land where class Admin having three attributes and one operation can be viewed when it runs this specification using USE tool. Model checker checks automatically the structure of the model as well as defines how many classes, variants, invariants, associations and post-pre conditions are available in the model. Figure 3.3.1 represents the class interface model and Figure 3.3.2 shows the USE environment class diagrams that validate the structure of the model to show in the following window with the correct command message structure.

Class name	**Attributes and types**	**Operations**
Admin	adminid : Integer name : String password : String	creatNewAccount(account : Real) : Real
Expenses	adminid : Integer expenseid: nteger expense : String expenseType: string	add (a : Expenses) remove (a : Expenses)

Payment	expenseid : Integer
	paymentid : Integer
	payment : Real
		sender(p: Employeepayment)
		reciever(p : Employeepayment)

Employee Payments	paymentid : Integer
	employeeid : Integer
	empname : String salary : Real
		salary(p : Payments)
		advance(p : Payments)

General Payments	paymentid : Integer
	generalpaymentid : Integer
	pattycash : Real
		recievedamount(p: Payments)
		dailexpenses(p : Payments)

Table 3.1. Classes, attributes, and operations defined in USE Textual specification

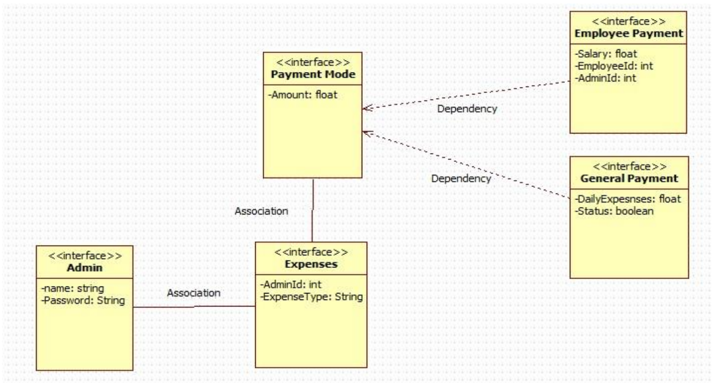
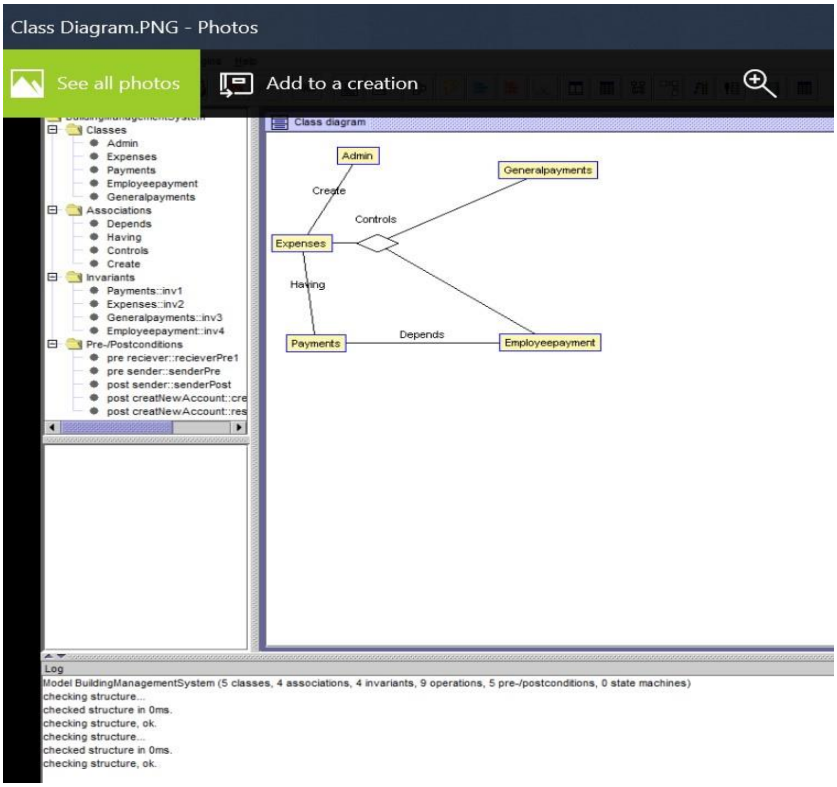


Figure 3.3.1 UML Class Interface Model of Case study



```
Model BuildingManagementSystem (5 classes, 4 associations, 4 invariants, 9 operations, 5 pre-/postconditions, 0 state machines)
checking structure...
checked structure in 0ms.
checking structure, ok.
checking structure...
checked structure in 0ms.
checking structure, ok.
```

Figure 3.3.2 USE Specification Environment of Graphical view of Class Diagrams including relationships, variants, pre-post conditions.

Below is the list of the USE specification textual commands in which classes, attributes, Operations, associations, and pre-post condition are defined.

```
-- $ProjectHeader: use 0.393 Wed, 15 March 2018 14:10:28 +0200$
-- Example illustrating pre- and postconditions
 Model BuildingManagementSystem

 -- classes
 class Admin
       attributes
              adminid :
Integer
name : String
password : String
       operations
              creatNewAccount(account : Real) :
       Real
end
class Expenses
       attributes
              adminid :
Integer
              expenseid : Integer
              expense : String
              expenseType : String
       operations
              add (a :
Expenses)
              remove (a : expenses)
end
class
Payments
       attrib
utes
              expenseid :
Integer
              paymentid :
Integer
payment : Real
       operations
```

```
              sender(p :
Employeepayment)
              reciever(p : Employeepayment)
end
class Employeepayment
       attributes
              paymentid :
Integer
              employeeid :
Integer
              empname :
       String
              salary : Real
       operations
              salary(p : Payments)
              advance(p :
Payments)
end
class
Generalpayments
       attributes
              paymentid : Integer
              generalpaymentid : Integer

              pattycash : Real
       operations
              recievedamount(p :
Payments)
              dailexpenses(p :
Payments)
end
```

Association in USE Specification by OCL constraints:

The following are the association defined with the applied multiplicity constraints using OCL language and the association *Depends between* many to one relationship of payments class to Employee payment.

```
association Depends between
             Payments[*]
             Employeepayment[1..*]
```

```
end
```

In similar way, association *Having* Expenses between Payment class many to many relationship is as under:

```
association Having between

            Expenses[*]
            Payments[*]
End
```

In a similar way, Class Expenses *Controls* between Emplyeepayment and Between Generalpayments many to many relationship is as under:

```
association Controls between
            Expenses[*]
            Employeepayment[*]
            Generalpayments[*]
end
```

In a similar way, Class Admin *Creates* between Expenses many to many relationship is shown as under:

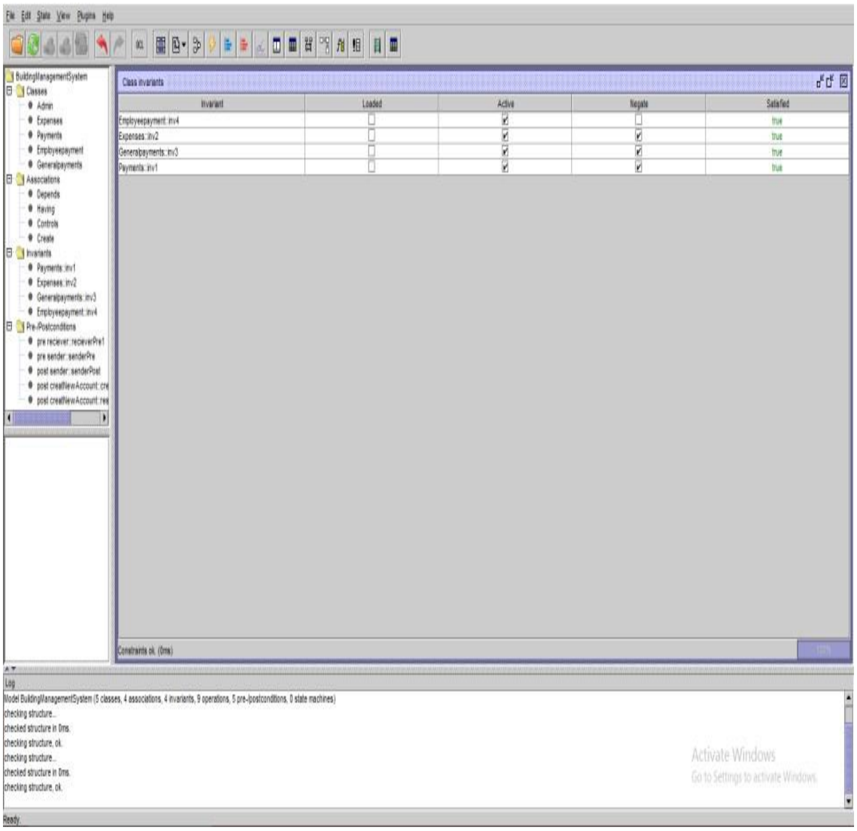
```
association Create between
            Admin[*]
            Expenses[*]
End
```

Constraints applying by USE OCL Model

The list of the constraints as defined below by the OCL invariants is applied on the *Payment* and *Expenses* class in the figure 3.3.3 which shows that the invariants checked directly as is shown in graphical model.

```
      -- constraints
      constraints
             context Payments
                    inv: paymentid = 1
             context Expenses
                    inv: expenseid=
             1
             context
Generalpayments
```

```
        inv:   pattycash >=
context Employeepayment
        inv:   employeeid =
paymentid
```



Invariant	Loaded	Active	Negate	Satisfied
Employeepayment::inv4	☐	✓	☐	true
Expenses::inv2	☐	✓	✓	true
Generalpayments::inv3	☐	✓	✓	true
Payments::inv1	☐	✓	✓	true

```
Constraints ok. (0ms)
```

```
Log
Model BuildingManagementSystem (5 classes, 4 associations, 4 invariants, 9 operations, 5 pre-/postconditions, 0 state machines)
checking structure...
checked structure in 0ms.
checking structure, ok.
checking structure...
checked structure in 0ms.
checking structure, ok.
```

```
Ready.
```

Figure 3.3.3 “3 Invariants check by showing in green to validate correctly”

Pre-Post Conditions OCL Constraints:

Here the way of applying constraints in USE specification by applying OCL Pre-Post conditions on the structural model is produced. Following is the list of commands which shows constraints applied on the class structure of *Payments that* checks if payment should be received by the employee; but before the payment is made, checks the Pre condition weather payment is defined or not. Below is the list of constraints which checks that the model is well defined:

.

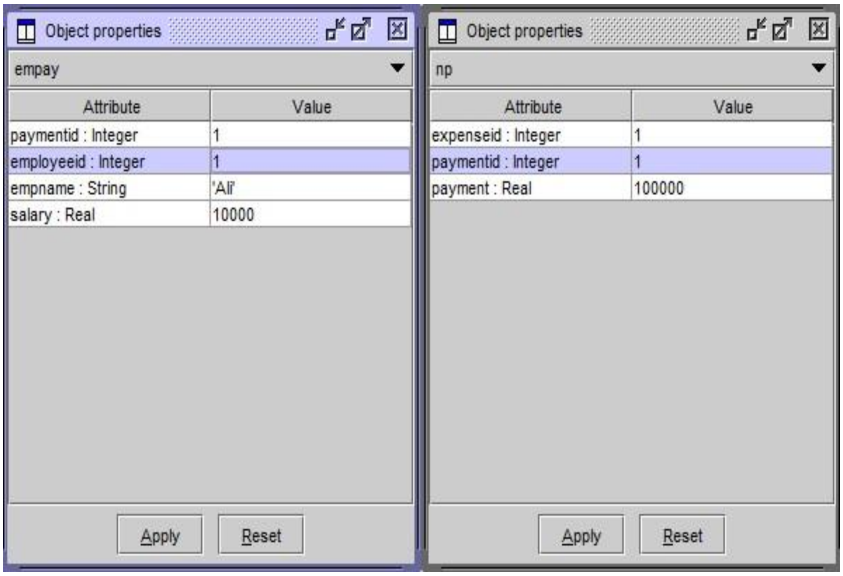
```
 constraints   context   Payments::reciever(p   :
Employeepayment)          pre       recieverPre1:
p.isDefined()   context   Payments::sender(p   :
Employeepayment)            pre        senderPre:
employeepayment->includes(p)     post  senderPost:
employeepayment->excludes(p)              context
Admin::creatNewAccount(account  :  Real)  :  Real
post creatNewAccountPost:
  account = account@pre * (1.0 +
account)   post resultPost:
result = name
```

3.4. VERIFICATION AND VALIDATION PROCESS USING USE

In section 3.4 we define the verification and validation process by using the USE graphical environment which gives more reliability and accuracy of Model Driven Development Environment. Figure 3.4.1 shows the USE object diagram of PEM by creating the object and sets the data whose mean time can be visualized by clicking the object Diagram.

Below is the list of the commands which creates two objects and a graphical view in the Figure:

```
use> !create nd:Admin

use> !set nd.name

:='mehran' use> !set

nd.password :='eris' use>

!create np:Payments

use> !set

np.payment:=100000
```


Object properties

empay

Attribute	Value
paymentid : Integer	1
employeeid : Integer	1
empname : String	'Ali'
salary : Real	10000

Object properties

np

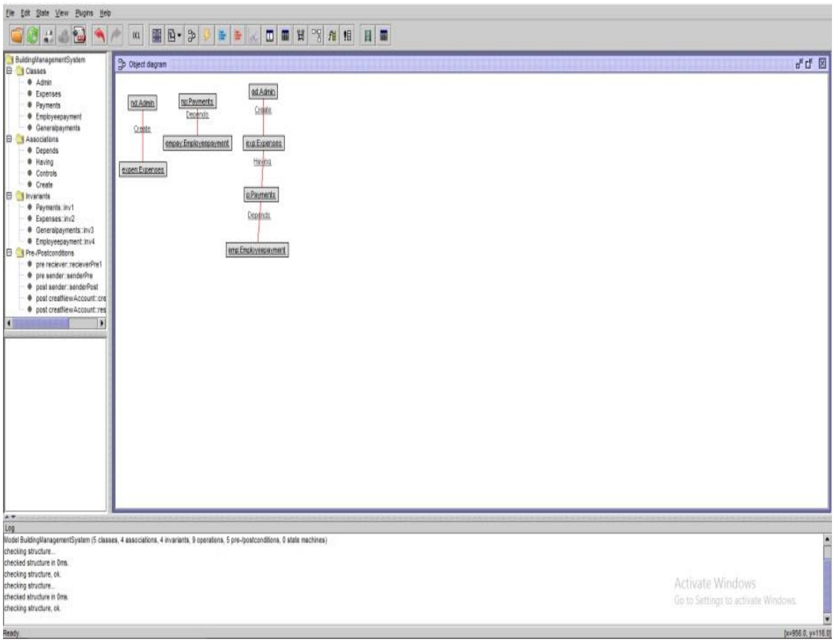
Attribute	Value
expenseid : Integer	1
paymentid : Integer	1
payment : Real	100000

Figure 3.4.1 USE Object Diagram represented the inserted object Data

We invoke the operation *Receiver* on the receiver object *new payment* and pass the object *empty* as one of the parameter. We also check that the preconditions also satisfy the condition and that the object model is working correctly.

```
use> !openter np reciever(empay)

precondition `recieverPre1' is

true use> info opstack

Payments::reciever(self:np,  p:empay)          [caller:   openter
  np reciever(empay)@<input>:1:0]
```

The above commands finally view the object model with the red line shown in Figure 3.4.2 between two objects ensures that the correctness properties are tested.



```
Model BuildingManagementSystem (5 classes, 4 associations, 4 invariants, 9 operations, 5 pre-/postconditions, 0 state machines)
checking structure...
checked structure in 0ms.
checking structure, ok.
checking structure...
checked structure in 0ms.
checking structure, ok.
```

Figure 3.4.3 USE Object diagram with red line represent the links counts

Now we have to verify the binding of the self-variable to identify the parameter P which represents the *Employee payment = empty*, as a result we find the graphical view of the object and the binding variable is shown as red link in Figure. 3.4.4.3.

```
use> info vars

  [frame 1]
    p : Employeepayment = empty
    self : Payments = np
  [frame   0]
  empty
  [object variables]   ad : Admin =
  ad   emp : Employeepayment =
  emp   empty :
  Employeepayment = empty
    exp : Expenses = exp
  expen : Expenses =
  expen   gp :
  Generalpayments = gp
  nd : Admin = nd   np :
```

```
Payments = np   p :
Payments = p
```

Operation Effects on classes:
In this section, we simulate and execute the operations which are defined by the system state. Using USE, system state can be visualized with the help of the state model. Now we check the pre-condition of the receive operation which is required by *the requires* in our model. We have link between the person class and the company class which can be visualized directly. In model we set the salary of the new employee to check the operation effects on classes' behavior.

```
use> !insert (np,empay) into
Depends use> !create
expen:Expenses use> !insert
(nd,expen) into Create use>
!insert (ad,exp) into Create
use> !insert (exp,p) into Having
use> !insert (p,emp) into Depends
```

Following are the steps to verify and validate the optional and required operations with a result value:

1. Using USE tool active operation is available in the call stack.
2. After viewing the call stack, if optional active result value is already provided , then the special OCL variable by default bound with the value of "result" variable is produced.
3. With this, all pre condition operation is satisfied, and as a result, answer appeared as true.
4. Now the local variable automatically cleared because it did not find the bonding value.

In our example, the precondition *Receiver* is satisfied by applying the following commands. use> info vars p : Employeepayment = empty self : Payments = np

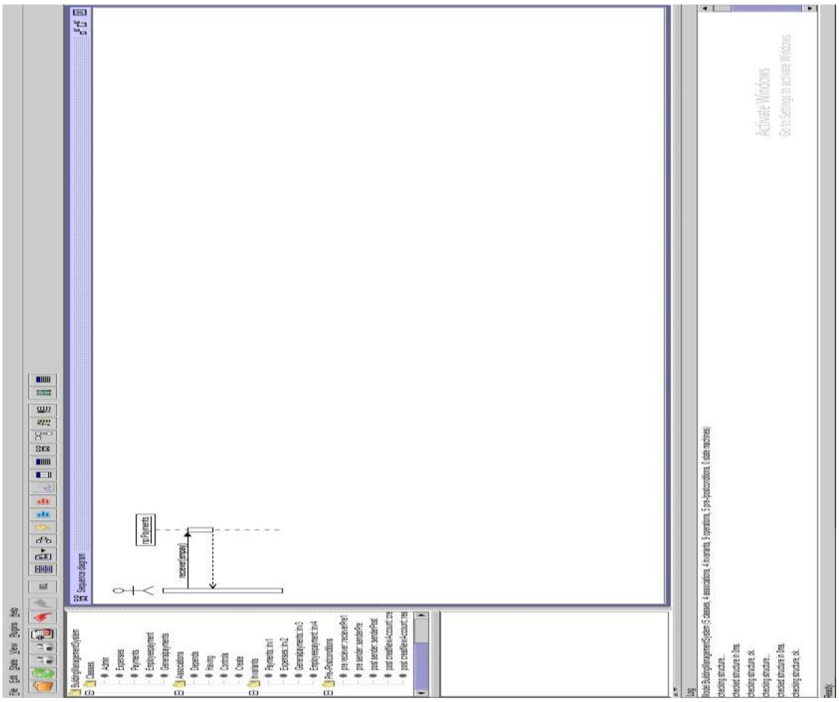
We call the operation AddNewAccount on the new employee newemp. The operation raise salary is given the new employee raise by the 10%.

use> !openter np reciever(empay)

precondition `recieverPre1' is true The above result we found that reciverPre1 is true check the operation is correctly working.

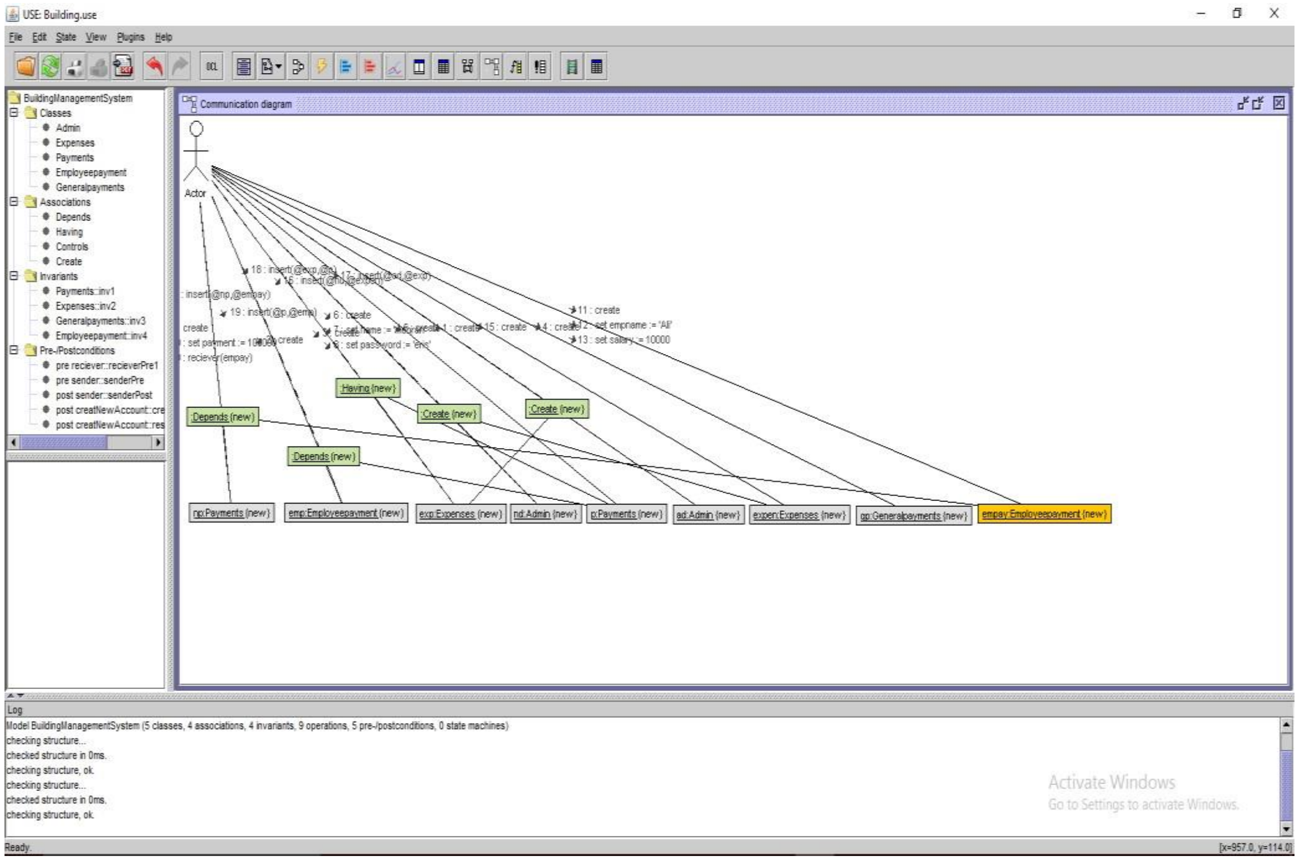
3.5. USE SUPPORTS SEQUENCE DIAGRAM

The USE methods identify and visualize a sequence of operations by calling the methods same as UML sequence models. Figure 3.5.1, Figure 3.5.2 and Figure 3.5.3 show design case study of PMS SYS Land which shows the sequence of objects call and response of the operations can be viewed. In this method, validation process is done in parallel automatically when we update each operation by applying the valid data.



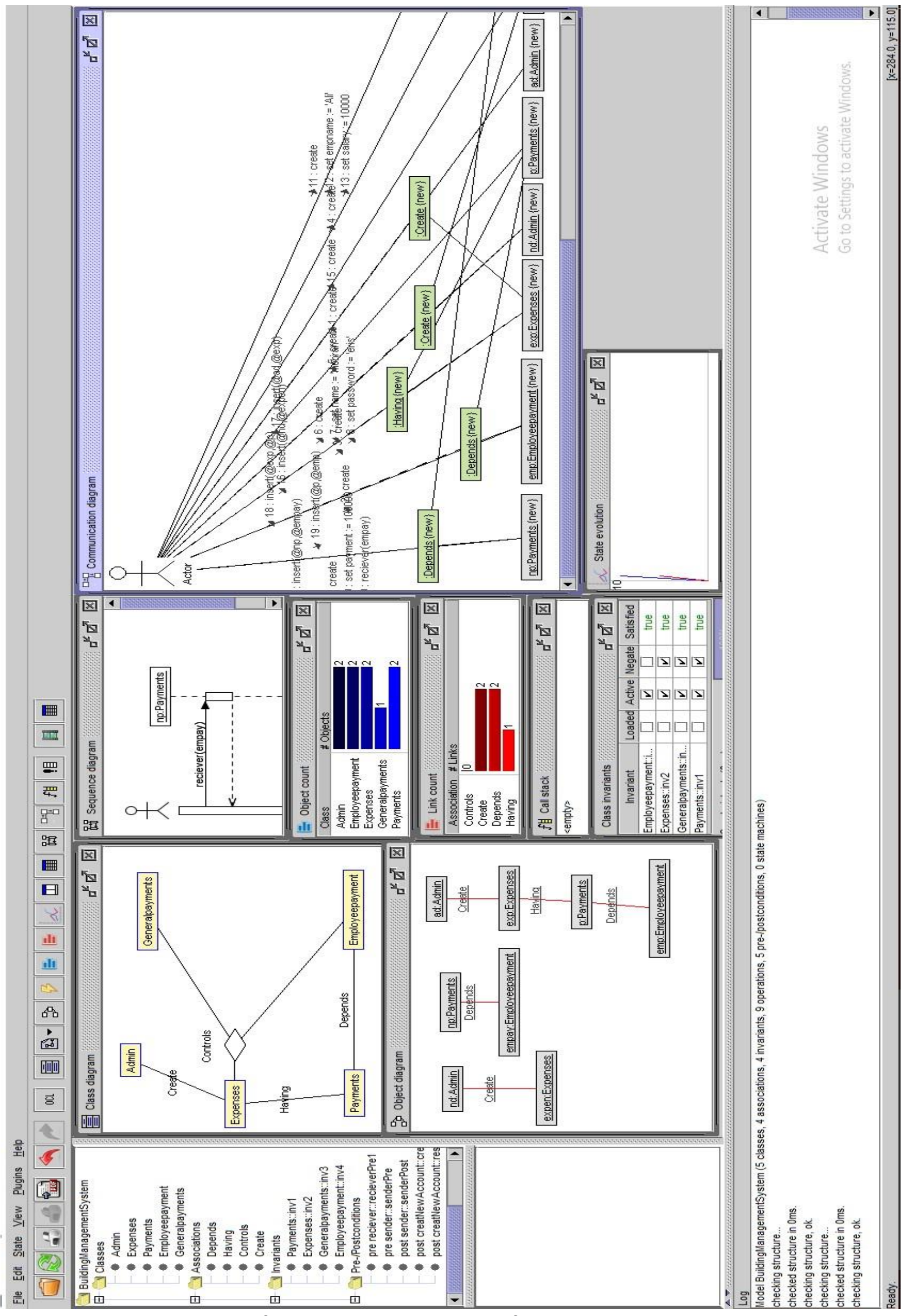
```
Model BuildingManagementSystem (5 classes, 4 associations, 4 invariants, 9 operations, 5 pre-/postconditions, 0 state machines)
checking structure...
checked structure in 0ms.
checking structure, ok.
checking structure...
checked structure in 0ms.
checking structure, ok.
```

Figure 3.5.1 Sequence Diagram for satisfying the operation call



```
Model BuildingManagementSystem (5 classes, 4 associations, 4 invariants, 9 operations, 5 pre-/postconditions, 0 state machines)
checking structure...
checked structure in 0ms.
checking structure, ok.
checking structure...
checked structure in 0ms.
checking structure, ok.
```

Figure 3.5.2 Communication Diagram including Object relationships.



Invariant	Loaded	Active	Negate	Satisfied
Employeepayment::i...	☐	✓	☐	true
Expenses::inv2	☐	✓	✓	true
Generalpayments::in...	☐	✓	✓	true
Payments::inv1	☐	✓	✓	true

```
Model BuildingManagementSystem (5 classes, 4 associations, 4 invariants, 9 operations, 5 pre-/postconditions, 0 state machines)
checking structure...
checked structure in 0ms.
checking structure, ok.
checking structure...
checked structure in 0ms.
checking structure, ok.
```

Figure 3.5.3 USE Specification Model diagrams of the Case study

CHAPTER 4. CONCLUSION AND FUTURE WORK

In this thesis, the concept of Verification and Validation of UML/OCL object Components model formally has been presented along with the framework for applying USE graphically specification methods.

The concept has been illustrated with structural and behavioral models of UML/OCL that are applied in a case study focusing on designed software for PMS SYS Land organization, using Model Driven Design Environment.

Discussion and Future Work

Development of systems that are based on the Model Driven Design Architecture indirectly supports the object oriented paradigm. Nowadays, OOD methodology is more popular and difficult to design. The verification and validation of models at design level are still very complicated and in this regard, the given methodology to some extent gives positive results but still numerous challenges, leading towards finding the right solution for various domains, need to be addressed.

A substantial part of the research regarding verification and validation of UML Object components model has focused on an efficient solution to architectural design challenges. One step in this direction is the illustration of object components model hypothesis presented in our research paper B. In contrast to the verification and validation of Object class diagram, we use the formal graphical method which gives more accurate and correct results at design level by enforcing some formal rules on the system design.

As far as matter of applying the supporting tools is concerned, it is clarified that we have formally not designed any new tool because it is beyond the scope of this research and due to complexity in achieving a satisfactory semantically description of design models, we have focused to utilize already available tool integrated with a new methodology for our case study. In such a scenario, OCL best fits in the problem and we have achieved the positive results produced in the case study in chapter 3 with the integration of UML/OCL by the USE graphical specification environment.

An additional interesting topic of research is to define and integrate more formal specification languages with UML modeling diagrams which can be easily developed and can generate the results commercially in software engineering field.

LITERATURE LIST

[1] Baresi, L., Blohm, G., el. Formal verification and validation of embedded systems: the UML-based MADES approach. *Software & Systems Modeling*, *14*(1), 343-363. 2016

[2] Alawneh, L., Debbabi, M., Hassaine, F., Jarraya, Y., & Soeanu, A. (2006, March). A unified approach for verification and validation of systems and software engineering models. In *Engineering of Computer Based Systems, 2006. ECBS 2006. 13th Annual IEEE International Symposium and Workshop on*(pp. 10-pp). IEEE.

[3] Cabot, J., Claris, R., & Riera, D. (2008, April). Verification of UML/OCL class diagrams using constraint programming. In *Software Testing Verification and Validation Workshop, 2008. ICSTW'08. IEEE International Conference on* (pp. 73-80). IEEE.

[4] Martin Gogolla and Khanh-Hoang Doan. Visualizing and Analyzing Discrete Sets with a UML and OCL Software Design Tool. In Yuri Sato and Zohreh Shams, editors, *Proc. 6th Workshop Set Visualization and Reasoning (SetVR 2018)*. CEUR Proceedings, 2018.

[5] Ober, I., Graf, S., & Ober, I. Validating timed UML models by simulation and verification. *International Journal on Software Tools for Technology Transfer*, *8*(2), 128-145. 2006

[6] David, A., Möller, M. O., & Yi, W. (2002, April). Formal verification of UML state charts with real-time extensions. In *International Conference on Fundamental Approaches to Software Engineering* (pp. 218-232). Springer, Berlin, Heidelberg.

[7] Soeken, M., Wille, R., Kuhlmann, M., Gogolla, M., & Drechsler, R. Verifying UML/OCL models using Boolean satisfiability. In *Proceedings of the Conference on Design, Automation and Test in Europe* (pp. 1341-1344). European Design and Automation Association. March 2010

[8] Snook, C., & Butler, M. UML-B: Formal modeling and design aided by UML. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *15*(1), 92-122. 2006

[9] Marcano, R., and N. Levy. "Using B formal specifications for analysis and verification of UML/OCL models." In *In Workshop on consistency problems in UML-based software development. 5th*

International Conference on the Unified Modeling Language, pp. 91-105. 2002.

[10] Cabot, J., Claris, R., & Riera, D. Verification of UML/OCL class diagrams using constraint programming. In *Software Testing Verification and Validation Workshop, 2008. ICSTW'08. IEEE International Conference on* (pp. 73-80). IEEE.

[11] Daniel Varro, Mark Asztalos, ..el. Transformation of UML Models to CSP: A Case Study for Graph Transformation Tools. In Andy Schürr, Manfred Nagl, and Albert Zündorf, editors, *Proc. 3rd Int. Workshop Applications of Graph Transformation with Industrial Relevance (AGTIVE '07)*, pages 540-565. Springer, Berlin, LNCS 5088, 2008.

[12] Beato, M.E., Barrio-Solórzano, M., Cuesta, C.E. and de la Fuente, P., UML automatic verification tool with formal methods. *Electronic Notes in Theoretical Computer Science*, *127*(4), pp.3-16.2005.

[13] Karsai, G., Agrawal, A., Shi, F. and Sprinkle, J., On the use of graph transformation in the formal specification of model interpreters. *J. UCS*, *9*(11), pp.1296-1321.2003

[14] Hazra, Rumpa, Shouvik Dey, and Jayashree Singha. "Modeling, analysis and verification of real-time resource access control protocols: a formal approach." *International Journal of Computers and Applications* (2017): 1-10.

[15] Ober, I., Graf, S., & Ober, I. Validating timed UML models by simulation and verification. *International Journal on Software Tools for Technology Transfer*, *8*(2), 128-145.2006

[16] Evans, A., France, R., Lano, K., & Rumpe, B.. The UML as a formal modeling notation. In *International Conference on the Unified Modeling Language* (pp. 336-348). 1998 Springer, Berlin, Heidelberg.

[17] D. Karlsson, P. Eles and Z. Peng. Formal verification of component-based design. *In Lecture Notes in Electrical Engineering Volume* 10, 2008, DOI: 10.1007/978-1-4020-8297-9

[18] Lima, V., Talhi, C., Mouheb, D., Debbabi, M., Wang, L., & Pourzandi, M. Formal verification and validation of UML 2.0 sequence diagrams using source and destination of messages. *Electronic Notes in Theoretical Computer Science*, *254*, 143-160. 2009

[19] Sengupta, Souvik, and Ranjan Dasgupta. "Use of semi-formal and formal methods in requirement engineering of ILMS." *ACM SIGSOFT Software Engineering Notes* 40, no. 1 (2015): 1-13.

[20] Dania, C., & Clavel, M. OCL2MSFOL: a mapping to many-sorted first-order logic for efficiently checking the satisfiability of OCL constraints. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems* (pp. 65-75). ACM. (2016, October).

[21] Sengupta, Souvik, and Ranjan Dasgupta. "Use of semi-formal and formal methods in requirement engineering of ILMS." *ACM SIGSOFT Software Engineering Notes* 40, no. 1 (2015): 1-13.

[22] Meyers, B., Deshayes, R., Lucio, L., Syriani, E., Vangheluwe, H. and Wimmer, M., September. ProMoBox: A framework for generating domain-specific property languages. In *International Conference on Software Language Engineering* (pp. 1-20). Springer, Cham. 2014

[23] Wu, D. and Schnieder, E., Scenario-based system design with colored petri nets: an application to train control systems. *Software & Systems Modeling*, *17*(1), pp.295-317, 2018.

[24] Guerson, John, Tiago Prince Sales, Giancarlo Guizzardi, and João Paulo A. Almeida. "OntoUML lightweight editor: a model-based environment to build, evaluate and implement reference ontologies." In *Enterprise Distributed Object Computing Workshop (EDOCW), 2015 IEEE 19th International*, pp. 144-147. IEEE, 2015.

[25] Gu-Beom Jeong and Guk-Boh Kim - A Framework for Security Assurance in Component Based Development. Computational Science and Its Applications – *ICCSA 2005Lecture Notes in Computer Science* 2005.

[26] Heidenreich F, Johannes J, Karol S, Seifert M, Thiele M, Wende C, Wilke C. Integrating OCL and textual modeling languages. *InInternational Conference on Model Driven Engineering Languages and Systems* 2010 Oct 3 (pp. 349363). Springer, Berlin, Heidelberg.

[27] Latella, Diego, Istvan Majzik, and Mieke Massink. "Automatic verification of a behavioural subset of UML statechart diagrams using the SPIN model-checker." *Formal aspects of computing* 11, no. 6 (1999): 637-664.

[28] Hilken, F. and Gogolla, M., Verifying Linear Temporal Logic Properties in UML/OCL Class Diagrams Using Filmstripping. In *Digital System Design (DSD), 2016 Euromicro Conference on* (pp. 708-713). IEEE. 2016

[29] Gabmeyer, S., Kaufmann, P., Seidl, M., Gogolla, M. and Kappel, G., A feature-based classification of formal verification techniques for software models. *Software & Systems Modeling*, pp.1-26. 2017

[30] Shaikh, Asadullah, and Uffe Kock Wiil. "Overview of Slicing and Feedback Techniques for Efficient Verification of UML/OCL Class Diagrams." *IEEE Access* (2018).

[31] Przigoda, N., Soeken, M., Wille, R., & Drechsler, R. Verifying the structure and behavior in UML/OCL models using satisfiability solvers. *IET Cyber-Physical Systems: Theory & Applications*, *1*(1), 49-59. 2016

[32] Elsayed, E.K. and El-Sharawy, E.E., Detecting Design Level Anti-patterns; Structure and Semantics in UML Class Diagrams. *Journal of Computers*, *13*(6), pp.638-655. 2018

[33] Cunha, Alcino, A. Garis, and D. Riesco. "Translating between Alloy specifications and UML class diagrams annotated with OCL." *Journal of Software and System Modeling (SOSYM), Vol* 14(1), 5-25,2015.

[34] Dang, D.H. and Gogolla, M., An OCL-Based Framework for Model Transformations. *VNU Journal of Science: Computer Science and Communication Engineering*, *32*(1).2016

[35] Khanh-Hoang Doan and Martin Gogolla. Extending a UML and OCL Tool for Multi-levels: Applications Towards Model Quality Assessment. In Loli Burgueno and et al., editors, *Proc. Workshop Multi-Level Modelling (MULTI 2017)*, pages 251-251. CEUR Proceedings 2018, 2017.

[36] Salem, Mohamed Oussama Ben, Olfa Mosbahi, Mohamed Khalgui, and Georg Frey. "R-UML: An UML Profile for Verification of Flexible Control Systems." In *International Conference on Software Technologies*, pp. 118-136. Springer, Cham, 2015.

[37] Belghiat, Aissam, Allaoua Chaoui, and Mokhtar Beldjehem. "Capturing and verifying dynamic program behaviour using UML communication diagrams and pi-calculus." In *Information Reuse and Integration (IRI), 2015 IEEE International Conference on*, pp. 318-325. IEEE, 2015.

[38] Bennama M, Bouabana-Tebibel T. A CTL-based OCL extension using CPN ML for UML validation. *International Journal of Critical Computer-Based Systems.* 2016;6(4):30221.

[39] Rahim, M., Hammad, A., & Ioualalen, M. A methodology for verifying SysML requirements using activity diagrams. *Innovations in Systems and Software Engineering*, *13*(1), 19-33. 2017

[40] Soeken, Mathias, Robert Wille, Mirco Kuhlmann, Martin Gogolla, and Rolf Drechsler. "Verifying UML/OCL models using Boolean satisfiability." In *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 1341-1344. European Design and Automation Association, 2010.

[41] Ben Abdessalem Karaa, Wahiba, Zeineb Ben Azzouz, Aarti Singh, Nilanjan Dey, Amira S Ashour, and Henda Ben Ghazala. "Automatic builder of class diagram (ABCD): an application of UML generation from functional requirements." *Software: Practice and Experience* 46, no. 11 (2016): 1443-1458.

[42] Hu, S. and Zhang, H., Modeling of aircraft brake system in UML-B. In *Reliability Systems Engineering (ICRSE), 2015 First International Conference on* (pp. 1-7). IEEE.

[43] Mayvan, Bahareh Bafandeh, and Abbas Rasoolzadegan. "Design pattern detection based on the graph theory." *Knowledge-Based Systems* 120 : 211-225. 2017

[44] Babkin, E. A., and N. O. Ponomarev. "Analysis of the consistency of enterprise architecture models using formal verification methods." *Business Informatics* 3 (2017): 41.

[45] Alagar, V. and M. Mohammad, Specification and verification of trustworthy component-based real-time reactive system, *In Procedings of SAVCBS'07, Dubrovnik, Croatia,* 2007.

[46] M. Friedl , A. Kellner, L. Weingartner, "Integration of domain specific simulation models into descriptive system models by using SysML", *Systems Engineering Symposium (ISSE) 2017 IEEE International*, pp. 1-5, 2017.

[47] Lano, K., Clark, D. and Androutsopoulos, K., UML to B: Formal verification of object-oriented models. In *International Conference on Integrated Formal Methods* (pp. 187-206). Springer, Berlin, Heidelberg. 2007

[48] Mukherjee, D., Shekhar, D., & Mall, R. Proposal for A Structural Integration Test Coverage Metric for ObjectOriented Programs. *ACM SIGSOFT Software Engineering Notes*, *43*(1), 1-4. 2018.

[49] Kuhlmann, M., Hamann, L. and Gogolla, M., Extensive validation of OCL models by integrating SAT solving into USE. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*(pp. 290-306). 2011.Springer, Berlin, Heidelberg.

[50] Chimiak-Opoka, J.D., Demuth, B., Awenius, A., Chiorean, D., Gabel, S., Hamann, L. and Willink, E., OCL tools report based on the ide4OCL feature model. *Electronic Communications of the EASST*, *44*. 2011

[51] Csertán, György, Gábor Huszerl, István Majzik, Zsigmond Pap, András Pataricza, and Dániel Varró. "VIATRA-visual automated transformations for formal verification and validation of UML models." In *Automated Software Engineering, 2002. Proceedings. ASE 2002. 17th IEEE International Conference on*, pp. 267-270. IEEE, 2002.

[52] Loli Burgueno, Antonio Vallecillo, and Martin Gogolla. Teaching UML and OCL Models and their Validation to Software Engineering Students: An Experience Report. *Journal on Computer Science Education, Taylor and Francis, UK*, 2018.

[53] Frank Hilken, Philipp Niemann, Robert Wille, and Martin Gogolla. Towards a Base Model for UML and OCL Verification. In Frédéric Boulanger, Michalis Famelis, and Daniel Ratiu, editors, *Proc. 11th Int. Workshop on Model Driven Engineering, Verification and Validation Workshop (MODEVVA'2014)*, pages 59-68, http://ceurws.org/Vol-1235/, 2014. CEUR Proceedings, Vol. 1235

[54] Loli Burgueno, Frank Hilken, Antonio Vallecillo, and Martin Gogolla. Generating Effective Test Suites for Model Transformations using Classifying Terms. In Huseyin Ergin, Richard F. Paige, Eugene Syriani, Steffen Zschaler, and Moussa Amrani, editors, *Proc. Workshop Verification of Model Transformation (VOLT 2016)*, pages 48-57. CEUR WS Proceedings 1693, 2016.

[55] Mira Balaban, Phillipa Bennett, Khanh-Hoang Doan, Geri Georg, Martin Gogolla, Igal Khitron, and Michael Kifer. A Comparison of Textual Modeling Languages: OCL, Alloy, FOML. In Achim D. Brucker, Jordi Cabot, and Adolfo Sanchez-Barbudo Herrera, editors, *Proc. Workshop OCL and Textual Modelling (2016)*, pages 57-72. CEUR WS Proceedings 1756, 2016.

[56] Przigoda N, Wille R, Przigoda J, Drechsler R. Automated Validation & Verification of UML/OCL Models Using Satisfiability Solvers. Springer; 2018 Jan 22.

[57] Abdul Hafeez, Syed Hyder Abbas Mussavi, Aqeel-Ur Rehman, "Ontology-Based Finite Satisfiability of UML Class Model", IEEE Access, Vol. PP, Issue 99, December 2017.

[58] H. He, Z. Wang, Q. Dong, W. Zhang, and W. Zhu, "Ontology-based semantic verification for UML behavioral models," *Int. J. Softw. Eng. Knowl. Eng., vol.* 23, no. 2, pp. 117–145, 2013.

[59] R. Drechsler, "Verifying the structure and behavior in UML/OCL models using satisfiability solvers," *IET Cyber-Physical Syst. Theory Appl., vol.* 1, no. 1, p. 49–59(10), Dec. 2016.

[60] Straeten, R.V.D, "Inconsistency Management in Model-Driven Engineering", Ph.D. Dissertation, Dept. of Comp. Sci., Vrije Uni. Brussels, September 2005.

[61] M. Balaban and A. Maraee, "Finite satisfiability of UML class diagrams with constrained class hierarchy," ACM Trans. Softw. Eng. Methodol., vol. 22, no. 3, p. 1, 2013.

[62] I. Traor and D. B. Aredo, "Enhancing structured review with model-based verification," IEEE Trans. Softw. Eng., vol. 30, no. 11, pp. 736–753, 2004

[63] M. E. Beato, M. Barrio-Solórzano, C. E. Cuesta, and P. de la Fuente, "UML automatic verification tool with formal methods," Electron. Notes Theor. Comput. Sci., vol. 127, no. 4, pp. 3–16, 2005.

[64] UML role in MDA, (March, 2017), http://www.omg.org/mda/faq_mda.htm#uml role in mda

[65] Dark Matter Systems Ltd., Overview of Unified Modeling Language in MDA, (2017), http://www.ultradark.com/01mda07umlintro.htm

[66] J. Cabot and R. Clarisó, "UML-OCL verification in practice," in Models Workshops, 2008, vol. 5421

PART –II

PAPERS

LIST OF THE PAPERS

A. Arifa Bhutto, D.M. Akbar. "Formal Verification of UML Profile" in Australian Journal of Basic and Applied Sciences, 5(6): 1594-1598, 2011.

B. Arifa Bhutto, D. M. Akber Hussain. Imran Anwar Ujan, Mehran Syed," Formal Approach for UML components based Development Profile" University of Sindh, Journal of Information and Communication Technology, 2(2): 125-129, 2018.

C. Arifa Bhutto, D.M. Akbar Hussain, "Validate UML model and OCL Expressions using USE Tool" Pertanika J.Sci.& Technology, 26(3):1465-1480,2018.

http://www.pertanika.upm.edu.my/Pertanika%20PAPERS/JST%20Vol.%2026%20(3)%20Jul.%202018/39%20JST(S)-0444-2018-3rdProof.pdf

D. Arifa Bhutto, D.M. Akber Hussain "An Android Based Cooperative Knowledge Acquiring Application" Mehran University Research Journal of Engineering and Technology, 37(3): 453-460, July 2018
DOI: 10.22581/muet1982
publications.muet.edu.pk/index.php/muetrj/article/download/486/211/

E. Sobia Mansoor, Arifa Bhutto**, "**Improvement of Students Abilities for Quality of Software Through Personal Software Process" Abilities for Quality of Software Through Personal Software Process", International Conference on Innovation in Electrical Engineering and Computational Technologies (ICIEECT), 2017, IEEE
DOI: 1109/ICIEECT/2017.7916550

ISSN (online): 2446-1636
ISBN (online): 978-87-7210-194-1

AALBORG UNIVERSITY PRESS