

A Framework for Clone Detection in UML Models



By:

Ayesha Irshad

(Registration No: MS-SE-20-328534)

Supervisor

Dr. Farooque Azam

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING,
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING,
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,
ISLAMABAD

July 23, 2024

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis written by **NS Ayesha Irshad** Registration No. 00000328534, of College of E&ME has been vetted by undersigned, found complete in all respects as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS/MPhil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the thesis.

Signature : _____

Name of Supervisor: **Dr Farooque Azam**

Date: 23-07-2024

Signature of HOD: _____

(Dr Usman Qamar)

Date: 23-07-2024

Signature of Dean: _____

(Brig Dr Nasir Rashid)

Date: 23 JUL 2024

A Framework for Clone Detection in UML Models

By

Ayesha Irshad

(Registration No:0000328534)

A thesis submitted to the National University of Science and Technology,

Islamabad

in partial fulfillment of the requirements for the degree of

Master of Sciences in Software Engineering

Thesis Supervisor:

Dr. Farooque Azam

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING,

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING,

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,

ISLAMABAD

July 23, 2024

Dedicated to my beloved Parents for their endless support and prayers, and to my Husband who always motivated and inspired me with unwavering support.

ACKNOWLEDGEMENTS

First, I would like to praise and honor the Almighty Allah, the most beneficent and the most merciful, for granting me the ability, courage, knowledge, and skills required to undertake and accomplish this task successfully. No doubt, He has facilitated my journey, and I am unable to achieve anything without His blessings.

I would also like to appreciate the sincere efforts of my supervisor **Dr. Farooque Azam** for guiding me throughout the journey of my MS thesis. Furthermore, I am also grateful to him for teaching us the courses of Model Driven Software Engineering (MDSE) and Software Development and Architecture (SDA). He professionally taught us both subjects in depth and from that; I developed my interest in continuing my research in the field of model-driven software engineering.

I would also like to express my gratitude to my Guidance Committee Members **Dr. Wasi Haider Butt**, and **Dr. Mehwish Naseer** for providing guidance and assistance to further improve my work with their valuable recommendations. I would like to express my sincere gratitude to **Muhammad Waseem Anwar** for his assistance and cooperation throughout the journey of my thesis in achieving my research objectives. Without his unwavering support, the completion of this dissertation would not have been possible. I appreciate his patience and support throughout the whole thesis.

I am deeply grateful to my beloved parents for raising me and being available whenever I needed them and for their unwavering support throughout every aspect of my life.

Finally, I would also like to extend my heartfelt gratitude to my husband and my family for their steadfast support and cooperation through the research journey.

ABSTRACT

Clone detection in software engineering has a fundamental role in ensuring the quality and maintainability of software systems. Developers often reuse several components of code in their software and code review to identify clones or refactoring of copied code is often neglected resulting in code clones. These cloned components can cause several consistency, bug propagation, maintainability, and quality issues. UML models are the essential artifacts usually in the initial phases of the process of software development, to specify and visualize the software design. These models serve as a blueprint to guide throughout all the phases of software development. Therefore, if there are clones in these UML models they will induce clones in further stages of software development as well. Therefore, these clones will propagate and amplify the clone-related issues from the basic to the final stages of software development. For this reason, it is equally essential to identify, track, and remove the duplicates in UML models as in code. Furthermore, a key goal of Model Driven Software Engineering (MDSE) is to generate code from models such as UML models. Consequently, increasing the importance of Model clone detection.

This study focuses on the application of Natural Language Processing (NLP) to detect clones within UML models. Initially, a UML model is created and clones are induced in the diagram. The model is exported in Extensible Markup Language (XML) format to represent the model in textual form. In the next step, the XML code is parsed to extract the relevant features of the model for clone detection purposes. Since the XML code of UML diagrams carries a lot of structural information that is irrelevant for clone detection and is also not balanced. Therefore, the extracted features are further preprocessed to represent them in a suitable format. Furthermore, the extracted data is labeled to represent clone and nonclone pairs. Moreover, for the detection of clones Natural Language processing techniques are used since the naming and representation of properties of elements of UML models are mostly in textual format. Therefore, NLP techniques can efficiently detect clones in UML Models. The proposed framework is applied to several case studies. These case studies validate the effectiveness of our approach in model clone detection.

Keywords: MDSE (Model Driven Software Engineering), UML (Unified modeling language), State Machine (SM), NLP (Natural Language Processing), Extensible Markup Language (XML)

TABLE OF CONTENTS

DECLARATION	i
ACKNOWLEDGEMENTS	ii
ABSTRACT	iii
TABLE OF CONTENTS	iv
LIST OF FIGURES	vi
LIST OF TABLES	vii
CHAPTER 1: INTRODUCTION	1
1.1 BACKGROUND STUDY	1
1.1.1 Clones in UML Models:	1
1.1.2 Unified Modeling Language (UML):.....	1
1.1.3 Model-Driven Software Engineering (MDSE):2	
1.1.4 Clone Detection in UML Models:.....	3
1.2 PROPOSED METHODOLOGY:	3
1.3 RESEARCH CONTRIBUTION:	4
1.4 THESIS ORGANIZATION:.....	4
CHAPTER 2: LITERATURE REVIEW	6
2.1 SOFTWARE CLONES:.....	6
2.2 CLONE DETECTION TECHNIQUES AND APPROACHES:.....	8
2.3 RESEARCH GAP:.....	9
CHAPTER 3: PROPOSED METHODOLOGY	11
3.1 CLONES IN UML STATE MACHINE	11
3.1.1 Clone sources in UML State Machine:	11
3.1.2 UML Model Clone Definitions.....	12
3.2 PROPOSED METHODOLOGY:	13
CHAPTER 4: IMPLEMENTATION	17
4.1. CASE STUDY 1: ATM SYSTEM	18
4.2. CASE STUDY 2: ELEVATOR.....	20
4.3. CASE STUDY 3: TELEPHONE LINE.....	22
4.4. CASE STUDY 4: ROOM RESERVATION SYSTEM	23
4.5. CASE STUDY 5: STUDENT PORTAL	24
4.6. CASE STUDY 6: ONLINE SHOPPING	25
4.7. CASE STUDY 7: AUTO CRASH PREVENTION SYSTEM (ACPS)	26
4.8. CASE STUDY 8: ARBITER	28
4.9. CASE STUDY 9: HEATING SYSTEM	29
4.10. CASE STUDY 10: REMOTELY PILOTED AIRCRAFT (RPA).....	30
4.11. CASE STUDY 11: POPCORN MACHINE	32

4.12. CASE STUDY 12: TRAFFIC LIGHTS	33
4.13. IMPLEMENTATION OF UMCD FOR CLONE DETECTION IN THE ATM SYSTEM CASE STUDY.	35
CHAPTER 5.....	41
VALIDATION.....	41
5.1 CASE STUDY	41
5.1.1 Application of UML Model Clone Detection (UMCD) Framework:	41
CHAPTER 6.....	44
6.1 THREATS TO VALIDITY	41
DISCUSSION	44
CHAPTER 7.....	46
CONCLUSION AND FUTURE WORK	46
REFERENCES	

TABLE OF FIGURES

Figure 1. Flow of research	4
Figure 2. Thesis organization.....	5
Figure 3. Simplified Workflow of UMCD.....	14
Figure 4. Proposed Workflow.....	15
Figure 5. ATM System State Machine.....	19
Figure 6. Elevator.....	21
Figure 7. Telephone line	22
Figure 8. Room Reservation System.....	23
Figure 9. Student Portal	24
Figure 10. Online shopping.....	26
Figure 11. Automatic Crash Prevention System.....	28
Figure 12. Arbiter.....	29
Figure 13. Heating System.....	30
Figure 14. Remotely Piloted Aircraft (RPA)	32
Figure 15. Popcorn machine	33
Figure 16. Traffic Lights.....	34
Figure 17. Tree Representation of ATM System	35
Figure 18. XMI representation of ATM System State-Machine.....	36
Figure 19. Parsed XMI code of ATM Case Study.....	36
Figure 20. Parsed XMI code after further pre-processing.....	37
Figure 21. Labeled Data.....	38
Figure 22. Highlighting clones in the Data	38
Figure 23. Clone Type 1 Results.....	39
Figure 24. Clone Type 2 Results.....	39
Figure 25. Clone Type 3 Results.....	40
Figure 26. UMCD Results from ATM case study	42
Figure 27. Performance results of UMCD on Other case studies	43

LIST OF TABLES

Table 1. UML diagram Categories	2
Table 2. Overview of Model Clone Detection Techniques.....	9
Table 3. UML Model clone definition.....	12
Table 4. List of Case Studies	17
Table 5. Results of UMCD on ATM Case Study.....	42

CHAPTER 1

INTRODUCTION

This chapter presents an introduction to the research work. It emphasizes the background study, research technique, problem definition, research contribution, and thesis organization.

6.1 Background study

1.1.1 Clones in UML Models:

In the software engineering domain, the identification of duplicate fragments of code is fundamental to ensure the quality and maintainability of software. Due to the heavy workload and short deadlines for software projects developers often reuse several software components. These reused components result in clones. It is important to review and refactor reused software components to avoid duplication and maintenance issues in later steps [1]. Before developing a software system its functionality is modeled to get a clear overview of the system requirements and expected functionality. Therefore, UML models are the key artifacts in the software development life cycle to specify and visualize the software design [2]. These models serve as a blueprint to guide through all the software development phases. Therefore, it becomes an essential requirement to identify and remove clones from UML models, as these clones will propagate to further stages of software development making it more difficult to remove or refactor them. Therefore, these clones will propagate and amplify the clone-related issues throughout the software development lifecycle. For this reason, it becomes equally important to detect, track, and remove the clones in UML models as in code [3].

1.1.2 Unified Modeling Language (UML):

Before actual coding and development of software, modeling is a crucial component of software projects and assists in the development of software projects of large, medium as well as of small scale. Software modeling plays a significant part in guiding the development process of software, therefore it acts as a blueprint and a roadmap just like the role played by models in construction projects or the development of some mechanical equipment. Using a model can ensure the successful development of a software project by ensuring that the design matches the requirements and that the required functionality is complete and correct. Models also help to determine if the model depicts requirements for resilience, flexibility and other non-functional attributes. Several studies reveal that complex software projects have a substantial probability of failure and are more likely to meet the cost and budget criteria. Modeling the

requirements prior to coding helps to visualize the design and if it will be able to meet the required criteria of cost, time, and other resources.

UML models support us in working at a higher abstraction level. UML models allow software developers to focus on different aspects of a prototype by hiding details and allowing the developers to view the big picture and analyze the behavior and requirements of a software system more efficiently. UML 2.0 provides a facility to get comprehensive insights into software in its practical environment, and can easily view links to other apps or, to other sites. Therefore, OMG's Unified Modeling Language (UML) aids the software developers to visually analyze the design and structure of software systems, to identify if it meets all of the requirements of the software.

Thirteen different kinds of diagrams, classified into three groups are defined in UML 2.0: out of which, 6 diagram categories characterize, static application structure; 3 represent general behavior types; and 4 represent possible interactions among elements of application [33]. UML diagram types are given in the following table 1.

Table 1. UML diagram Categories

Diagram Category	Diagram types
Structure Diagrams	Class Diagram, Object Diagram, Component Diagram, Composite Structure Diagram, Package Diagram, Deployment Diagram
Behavior Diagrams	Use Case Diagram, Activity Diagram, State Machine Diagram
Interaction Diagrams	Sequence Diagram, Communication Diagram, Timing Diagram, Interaction Overview Diagram

1.1.3 Model-Driven Software Engineering (MDSE):

It is a specific branch of software engineering, which emphasizes the use of models as important components throughout the entire development process. In MDSE, models are employed to specify the software system being developed. Automatic model transformations are then used to do various operations on the models, including code generation, model integration, and deconstruction [3]. MDSE also aims to automatically generate executable software code from models directly.

UML serves as the basis for Model Driven Architecture (MDA) of Object Management Group (OMG). Depending on the preference of the developer, they can either create platform-specific or platform-independent models. The MDA approach effectively utilizes both variations. Initially, Each MDA model or application is fundamentally grounded on a Platform-Independent Model (PIM). PIM accurately depicts its behavioral and functional aspects but does not encompass practical elements. The PIM is then used to generate Platform-Specific Models (PSMs) in UML by utilizing MDA tools for development that adhere to the standardized mapping guidelines of OMG. Each PSM corresponds to a specific target platform selected by the developer. The conversion process is largely automated, but it is not a magical process. The developer is required to annotate the initial Platform-Independent Model (PIM) before

generating a PSM, to create a more specific PIM that is still independent of any particular platform. This annotated PIM includes more data about the desired semantics and guides the tool to make appropriate choices during the conversion process. Since there is resemblance between middleware systems of some certain genre component-based, or messaging-based, therefore, this information from PSM can be incorporated into a PIM without any modifications. However, the developers will have to precisely adjust the generated PSMs, especially in the early phases of MDA and as the tools and the algorithms evolve this requirement of adjusting and fine-tuning the PSMs decreases [33].

The OMG establishes the MDA as a new paradigm for designing software systems, focusing on the role of models as significant artifacts in the software development process. MDA models are defined in UML [2]. Furthermore, a fundamental purpose of MDSE is to produce code from models such as UML models [4]. Consequently, enhancing the significance of model clone identification.

1.1.4 Clone Detection in UML Models:

Since UML models are the primary artifacts in the software development life cycle, therefore, it becomes equally essential to detect and remove clones in UML models. If these clones are left unresolved, they can result in bug propagation and if modification is required in the design then all the cloned model fragments will also require modification. If clones are not properly tracked, identified, or refactored it will result in huge maintenance expenses also making the process more time-consuming. Therefore, detecting clones in UML models becomes a necessary part of the software design process.

Several researchers have modified and applied the code clone detection approaches for the detection of model clones. Such as clone detection approaches based on, text such as code, tokens, tree representations, metric representation and comparison, semantic-based and some researchers used hybrid approaches, etc. Still, there is a need for continuous improvement in Model clone detection approaches for more accurate identification of clone pairs since, models have a lot of structural information that is not necessary for the process of clone detection, extraction of relevant features itself is a complex task.

6.2 Proposed Methodology:

The research approach involves several phases. Figure 1 depicts the flow and phases of this research. The process of research starts with a literature review that's a crucial step for a comprehensive overview of the existing studies in model clone detection. This literature review helps in identifying research gaps in the existing studies. This literature study provides a basis for comprehending the amount of information that already exists in these fields, therefore, identifying the research problem. Subsequently, the identified problems are resolved by proposing a solution based on natural language processing. The proposed solution is thoroughly explained, emphasizing its key components and outlining the required steps for implementation. The suggested approach is described, and then the specifics of its execution are given. The next step is the implementation of the proposed solution. Implementation includes all the technical elements, resources, and frameworks that will be applied throughout the implementation procedure, guaranteeing transparency and stability. The proposed solution is validated by using several case studies.

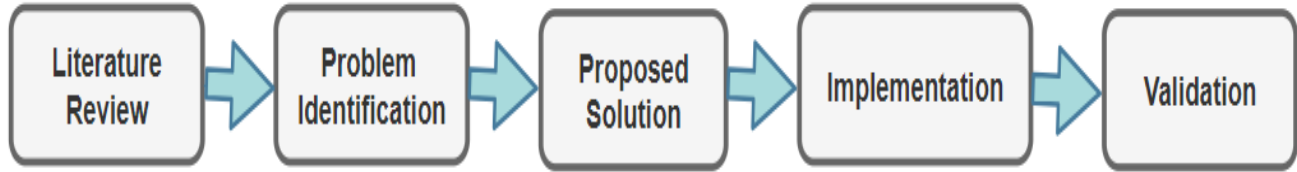


Figure 1. Flow of research

By the use of this approach, this research seeks to close the known research gaps, increase knowledge of NLP techniques, and contribute to the fields of clone detection in UML Models.

6.3 Research contribution:

- For the identification of clones in UML models, specifically in UML state machines, 12 case studies of UML state machines are created using Eclipse Papyrus, which is a robust, open source Model-Based Engineering tool.
- The state machine models of 12 case studies are used as a data set for validation of the proposed approach for clone detection in UML models.
- For extracting required relevant features from complex XML code of UML models, the code is parsed by using a Python IDE, Pycharm. The relevant features are extracted and further processed to represent data extracted data in tabular format.
- Furthermore, the clones are manually labeled to calculate the accuracy of the proposed framework.
- Clones are identified in the diagram by using Techniques of natural language processing (NLP) since a lot of information in UML diagrams is in textual form. Therefore, we can effectively apply clone detection techniques to identify similarities between the elements of the UML diagrams.

6.4 Thesis Organization:

The overview of the UML model clones and the proposed approach are both briefly described in Chapter 1, which serves as an introduction to this research. Figure 2. presents a clear organization of the thesis. In Chapter 2, a thorough literature analysis is conducted to look at the previous work of various scholars in the fields of clone detection in UML models. Chapter 3 provides a detailed explanation of the methodology used in the research and discusses the suggested way to address the identified problem. Chapter 4 details the implementation specifics, including both the practical and technical aspects of the development process. In Chapter 5, validation of the proposed framework is presented. The proposed framework is validated using several case studies. A complete description of the case studies is also presented in the chapter. Discussion, as well as any limitations encountered throughout the study, are covered in detail in Chapter 6. The thesis is concluded in Chapter 7 with general conclusions based on the findings and suggestions for more research in the area.

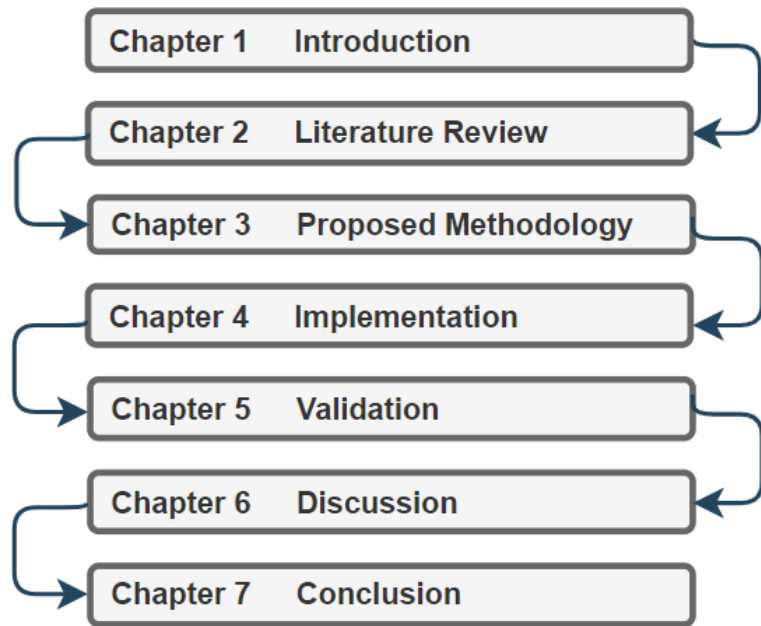


Figure 2. Thesis organization

CHAPTER 2

LITERATURE REVIEW

Chapter 2 conducts an extensive literature review and comprises three major sections. The first section, 2.1, thoroughly explores the field of software clones especially clones in UML models. Section 2.2, presents several clone detection techniques and approaches discussed in the previous research offering a critical evaluation of previous studies' techniques, results, and research methods, while the research gap is discussed in section 2.3.

2.1 Software clones:

In the software engineering domain, the identification of clones is an important research area. Due to heavy workloads and short deadlines, software developers often reuse several code fragments to accelerate software development. The significance of code review and refactoring is often neglected. This induces clones in the software that make the system more vulnerable to security and maintenance issues as these clones can rapidly escalate bug propagation. Since, if there is a bug in one code segment and that code segment is duplicated somewhere else in the program, it becomes crucial to inspect every duplicated code fragment for that bug [5]. Subsequently, the maintenance cost rises and the process is time-consuming as well [6] [7].

In the literature, four different types of code clones are discussed:

- Type-1 code-clones known as exact clones are duplicated components of a program with some minor changes in comments, layout, and whitespaces.
- Type-2 code-clones known as renamed clones are components of the program that are alike but some unique identifiers such as variables, functions, or other program elements are renamed along-with some variations in comments, identifiers, types, literals, and layouts.
- A type-3 code-clones also called gapped clones or near-miss clones include some modification in code such as the addition or removal of code or rearrangement of code segments.
- A type-4 code-clones also called Semantic clones have multiple segments of code with similar functionality, but varying syntactic variants are used for implementation [8] [9].

For the development of software projects using models is the best practice for clear overview and guidance throughout the process of software development. Therefore, models are integral to the development of software systems. Models for larger projects can be complex and may contain duplicated model fragments [3]. Since models are developed during the initial stages to guide through the process

of software development. If the duplicated model fragments are not removed these model clones will propagate throughout the software development phases and identifying and removing bugs will become more costly and time-consuming. Therefore, it is equally important to identify duplications in models as in code [18]. Since there is no explicit or standard definition of model clones, several researchers have provided their own definitions of model clones as code clones [19] [20]. Four model clone types have been discussed in the literature.

- Type 1 Model-clones: also referred to as exact model clones or identical model clones are exact duplicates ignoring the layout and visual aspects.
- Type 2 Model-clones: also called renamed or modified model clones have changes in elements or attribute names and changes in layout and visual aspects are ignored.
- Type 3 Model-clones: also named near-miss model clones allow renaming, additions, or removals of parts while ignoring the changes in layout and visual aspects.
- Type 4 Model-clones: also named semantic model clones are significantly different in structure but are semantically similar.

The researcher in [21], defined model clones as, Type-1 or exact model clones as duplicate model features with only differences in visual aspects such as presentation, and layout. For example, in UML sequence diagrams interaction elements may share the same "name," "receive Event," "send Event," and "message Sort" attributes but differ in presentation elements like fonts, sizes, positions, or colors. Type-2 also named as renamed model clones, model fragments might display variations in element or attribute names and also differ in, layout, format, or visual aspects. E.g., in sequence diagrams, two lifeline elements involved in alike conversations may have distinct names, xmi-id attributes, and changes in font, size, position, etc. These lifelines are classified as renamed clones if they engage in a set of messages that are similar within a particular conversation. Type-3 known as near-miss model clones are model elements that exhibit minor discrepancies or alterations such as the addition or deletion of elements such as variations of interaction elements in sequence diagrams, beyond differences in attribute names and visual variations already noted in Types 1 and 2. The acceptable degree of variance is adjustable based on a configurable threshold.

Another study [24] presented the following definition of model clones: Sort 1 (exact clones): Application segments that are alike apart from differences in whitespace and comments. Sort 2 (parameterized clones or renamed clones): Software segments that are similar in structure and syntactic methods, apart from variations in literals, identifiers, types, formatting, and comments. Sort 3 (gapped clones or near-miss clones): Copied software fragments with additional variations such as additions or deletions of declarations, along with alterations in types, literals, identifiers, and formatting. Sort 4 (semantic clones): Fragments of a software code/model that perform the same functions but do not share a textual similarity.

Similarly, several other studies provided their own definitions of model clones and modified them according to the type of models such as UML models or Simulink models; furthermore, they also altered the definitions as per the attributes of their specific models used in the research such as sequence diagram or class diagram or any other type of model. All the different types of models have varying characteristics. Therefore, the definition of clones in those models differs accordingly.

2.2 Clone detection techniques and approaches:

Several studies have proposed different approaches for uncovering different types of code clones [8] [10]. Some studies used text-based approaches that can efficiently identify exact clones [11] [12]. Some Researchers used Token-based approaches that can identify renamed clones more efficiently [13]. Tree-based approaches are proposed by some studies that have high efficiency in detecting Near Miss clones [14]. Metric-based approaches are used for clone detection in several studies and these approaches can identify near-miss clones with more precision [15]. Some studies proposed semantic, Hybrid approaches for the clone detection process to detect type-4 clones efficiently with better precision along with other categories of code clones [16] [17].

An approach to identify type-3 clones in UML sequence diagrams is proposed in [21]. It addresses the lack of research on duplication or clone detection in the dynamic functionality of interactive models and systems. The paper in [22] describes static dependencies between entities in the process of manufacturing by using UML class diagrams, emphasizing the significance of fault detection and consistency checks. A suffix tree-based approach is proposed in [23] to identify duplication in UML sequence diagrams. They verified their approach on six industrial case studies with 100% precision and 92% recall. An approach for clone detection in behavioral models is proposed in [24]. They use the Nicad tool for identifying duplicates in sequence diagrams. A similarity detection algorithm is proposed in [25] based on XML parsing by using a DOM parser. They verified their proposed approach to UML class diagrams by using the case study of the library management system. In [26] a technique for identifying clones based on tree comparison is proposed. After parsing the XML code of UML class diagrams, subtrees are compared to report similarity as model clones. A case study of Enrollment and teaching packages is used to verify the approach.

Another research proposes Similacode which utilizes Natural Language Processing (NLP) techniques, vector space models, and similarity metrics to detect the similarity in code. They have demonstrated the successful detection of duplicated code in the Python programming language.[27] Another study proposes a text similarity algorithm that is based on matching semantics and discusses the significance of NLP techniques such as algorithms for textual similarity. It has applications in an extensive variety of fields such as in optimizing search engines and detecting plagiarism in textual data etc. [28]. A BERT model is introduced in another study that can efficiently identify semantic sentence similarity. The study demonstrates the application of a fine-tuned model to reduce neuron count in neural networks. Therefore, it decreases the time and storage required for creating training data for deep learning models [29].

The SSCD, a clone detection technique that employs a nearest-neighbor strategy based on BERT, is introduced in another study; it addresses the inadequacy of performing pairwise comparisons in larger datasets of code. Additionally, SSCD aims to optimize recall of near-miss and clones and semantic clones [30]. Another research applied CodeBERT on multiple datasets to assess its performance in the area of clone detection. The study demonstrates that this approach can efficiently identify exact clones and semantic clones with high recall [31]. Therefore, model clone detection is a continuously evolving field requiring innovative approaches to detect duplicated model fragments more precisely.

Table 2. Overview of Model Clone Detection Techniques

Paper	Database	Year	Technique	Target
[3]	ACM	2010	Matching Algorithm	UML Models/ Class diagram, Activity diagram
[23]	IET Software	2010	Suffix Tree	UML Models/Sequence Diagrams
[43]	Springer	2011	Pattern-based	Simulink Models
[19]	IEEE	2012	Nicad	Simulink Models
[26]	IEEE	2012	Tree Pruning and tree Matching	UML Class diagram
[39]	ACM	2014	Text-Based Similarity	Model Clone/Class diagram
[20]	Springer	2015	Matching Algorithm	UML Models/ Class Diagram, Activity diagram
[21]	IEEE	2016	NiCad	UML Models/Sequence Diagrams
[40]	IEEE	2017	Reachability graphs	UML Activity Diagrams
[41]	IEEE	2018	Control-Flow based modeling	UML Models/ Class diagram
[42]	Springer	2019	eScan and conQAT	UML Models/class diagram
[8]	ACM	2019	Token Based Similarity	UML Models/class diagram

Researchers continue to discover advanced tools and techniques to boost clone detection capabilities in various domains.

A brief overview of model clone detection techniques from literature is shown in table 2. It shows the different approaches used for model clone detection and their targeted models such as Simulink or UML models etc.

2.3 Research Gap:

Although comprehensive research has been conducted in the domain of model clone detection, there remains a significant gap in the literature concerning clone detection specifically within UML state machines. The existing methodologies and tools designed for other forms of model clone detection are often not directly applicable to the semantic and structural specifics of UML state machines, emphasizing the need for targeted research in this field.

Clones in UML State Machine diagrams can lead to redundancy, maintenance challenges, and difficulties in understanding the intended behavior of the system. Detecting and addressing clones in State Machine diagrams is important for maintaining the clarity, efficiency, and correctness of the diagram. Therefore, addressing this gap will also enhance the robustness and effectiveness of model-driven engineering practices, especially in complex software systems where behavioral duplications are frequent yet challenging to detect.

CHAPTER 3

PROPOSED METHODOLOGY

In this project, the emphasis is on the identification of Model clones in UML state machine diagrams. This section defines clones in UML state machines, discusses possible clone sources, and the proposed methodology to identify the model clones of type-1, type-2, and type-3 clones.

3.1 Clones in UML State Machine

UML state machines are one of the important primary artifacts in the software development process and are used to model the dynamic behavior of a system, especially how an object in a system transitions from one state to another in response to some event or trigger. They help in understanding system functionalities and play a crucial role in the development process, especially in complex systems such as Automated Teller Machines (ATM), business process management and workflow systems, Interactive applications, including mobile and web apps, and many others. Identifying clones in UML state machines will improve maintainability, reduce redundancy, and help maintain consistency across different parts of the UML state machine model.

3.1.1 Clone sources in UML State Machine:

A state machine is composed of several components such as states, transitions, events, triggers, actions, initial states (starting point of a system), final states (indicate the completion or termination of a process), constraints, and conditions. Where a state represents a specific mode or behavior of a system that can change based on some external or internal event or trigger. Events are external stimuli or inputs that trigger state transitions that cause the system to change its state. Transitions represent the change from one state to another in response to an event or condition. Actions represent behavior or activities performed when a state transition occurs.

Pertaining to UML State Machine diagrams, model clones refer to similar or identical segments of the diagram that represent the same behavior or state transitions. Clones in UML state machines can occur in several possible ways such as in

- **States:** If two or more states of a state machine are identical that is have identical or nearly similar names, initial activities, and outgoing transitions therefore representing similar behavior.
- **Transitions:** Transitions between the states that perform similar action and have similar names and triggers or constraints associated with them.
- **Constraints:** Similar constraints and constraint values associated with the transition.
- **Actions:** Actions or activities associated with states or transitions that are repeated across different portions of the diagram.

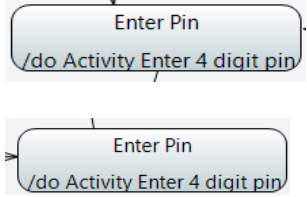
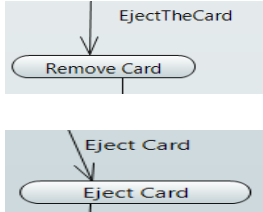
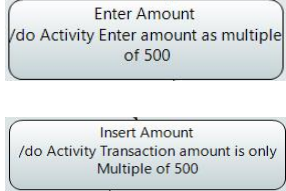
- **Parallel Structures:** Sections of the diagram that have similar structures, representing the same sequence of states and transitions.
- **Similar Triggers:** Transitions that have similar triggering events but lead to different states.

Detecting these clones' state machine diagrams can assist in simplifying the design, reducing redundancy, and ensuring that the diagram represents the intended system behavior more precisely. This is particularly essential for complex systems where state machine diagrams are used to model intricate state transitions and interactions.

3.1.2 UML Model Clone Definitions

As in other UML models, there are four possible types of Model clones in UML state Machines. The definitions of Clones in UML state machines are given in Table 1.

Table 3. UML Model clone definition

Clone Type	Definition	Example
Type 1 (Exact Clones)	For the same element type i.e., Region, State, Transition, or Choice if they have identical names and other parameters of the element i.e., they are identical model fragments then they will be classified as Model clone type 1.	
Type 2 (Renamed Clones)	If the element type is the same i.e., State, Transition choice, etc., but there are minor changes in names or representation of the element such that names are different but have the same meaning. While other parameters of the element remain unchanged. Such clone pairs will be classified as model clone type 2.	
Type 3 (Near Miss Clone)	If the element type is the same i.e., State, Transition choice, etc., and there are modifications in the name or attributes of the elements, such as the addition or removal of some statement then it will be near miss model clone then such similarity in the model elements will be classified as model clone 3.	

Type 4 (Semantic Clones)	For the same element types, model elements or model fragments that are functionally similar but are significantly structurally different and are implemented using different syntactic variants. Such clone pairs will be classified as model clone type 4.	
---------------------------------	---	--

3.2 Proposed methodology:

In this thesis, a framework for clone detection in UML models is proposed, UMCD (UML Model clone detection). This framework aims to detect clones of type1, type2, and type3 in UML state machine diagrams. The simplified flow of work is presented in Figure 3. Initially, a State machine model is created in Eclipse Papyrus. State machine diagrams are used for behavioral representation of the software systems. By representing different stages of the software system using states and transitions based on events. The state machine model is then saved in standardized XMI format by using the built-in services of the Eclipse papyrus. XML (eXtensible Markup Language) is a multipurpose, flexible, and extensively supported markup language that allows information to be structured in a format that is readable by both humans and machines. When applied to UML diagrams, XML provides several benefits such as a structured representation of complex UML diagrams, and XML format can be easily parsed and processed. It also provides a platform-independent and human-readable format for storing UML diagrams.

In the next step, XML code is parsed to remove the irrelevant code i.e., XML-specific code and other structural details generated by the tool along with diagram details. Parsing the XML code of UML models serves as the foundational step in preparing the model for further processing and utilization in a software development environment. By parsing the XML code, the required features of the state machine diagram that are relevant to the clone detection process are extracted and stored in a .csv file. Features are stored in tabular format for better representation and understanding. The extracted data is further preprocessed to make the data more suitable for clone detection. Preprocessing of data includes removing null values, converting data in lowercase, removing unnecessary data columns that do not play any significant role in the clone detection process, replacing complex IDs with simpler IDs, Since, UML assigns complex IDs to model elements that make them difficult to comprehend and manipulate. The extracted data is then labeled manually. Our approach aim to detect clones of type-1, type-2, and type-3 only so the data is labeled as

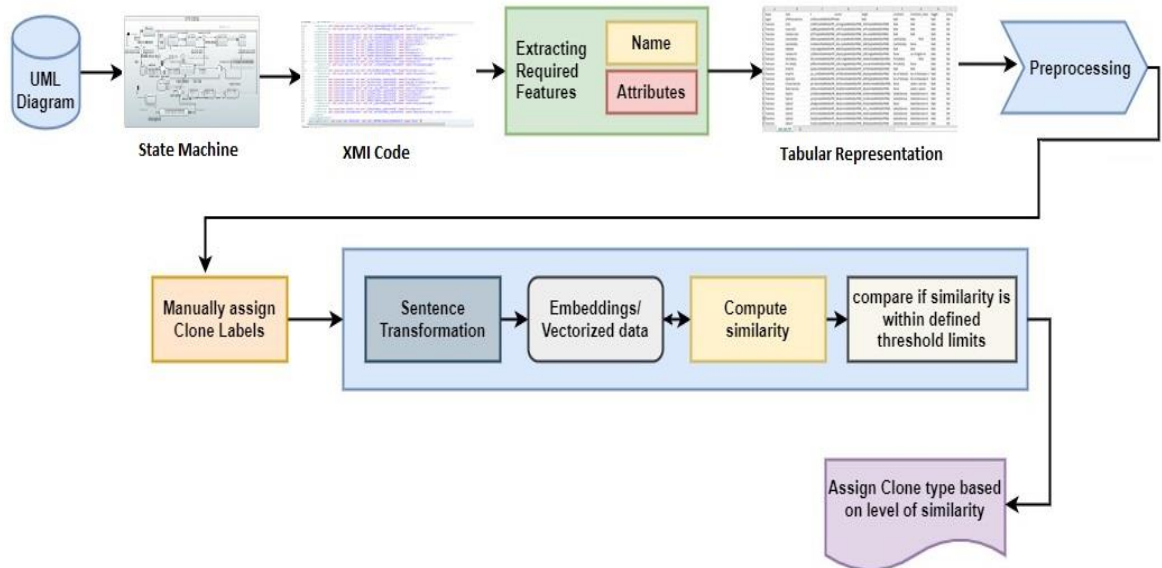


Figure 3. Simplified Workflow of UMCD

Clone-1, Clone-2, Clone-3, and nonclone accordingly. Furthermore, our approach makes use of natural language (NLP) techniques to identify clones. We choose to use NLP techniques for model clone detection in UML state machines since UML models comprise of numerous textual data such as in the naming of model elements, comments, constraints, and several other attributes. NLP can help identify not only exact duplicates but also semantically similar elements across different parts of the UML diagrams. Furthermore, NLP provides several effective techniques to transform text in a format where similarity between different text strings can be easily computed. The detailed workflow of the proposed framework is presented in Figure 4. After extracting features from the XMI code of the state machine, similarity is calculated between the model elements such that elements of the same type are extracted. For elements of the same type such as a state is compared to other states for similarity and a transition is compared with another transition to compute similarity. If two or more states, transitions, or any other model elements are exactly duplicated then the element is labeled as clone type 1 since these model elements are exact replicas of each other. Use the sentence transformer model to build embeddings of extracted data.

For the detection of clones of Type 2 Model clones, we used a TF-IDF vectorizer. The Term Frequency-Inverse Document Frequency (TF-IDF) vectorizer assesses the significance of a word within a document in relation to a collection or corpus. This significance scales with the frequency of the word in the document, adjusted by its frequency across the corpus. TF-IDF is frequently used in text mining and information retrieval to convert text into numerical representations that can be utilized to train ML algorithms for prediction. Therefore, by using TF-IDF vectorizer the elements/segments of extracted data are converted into vectors. The text in each row is transformed into a TF-IDF vector representing the importance of each term in the context of the diagram. The TF-IDF calculation will consider how

important a term such as a name or an attribute value of a model element is compared to its frequency across other elements of the model. Then similarity between vectors is measured

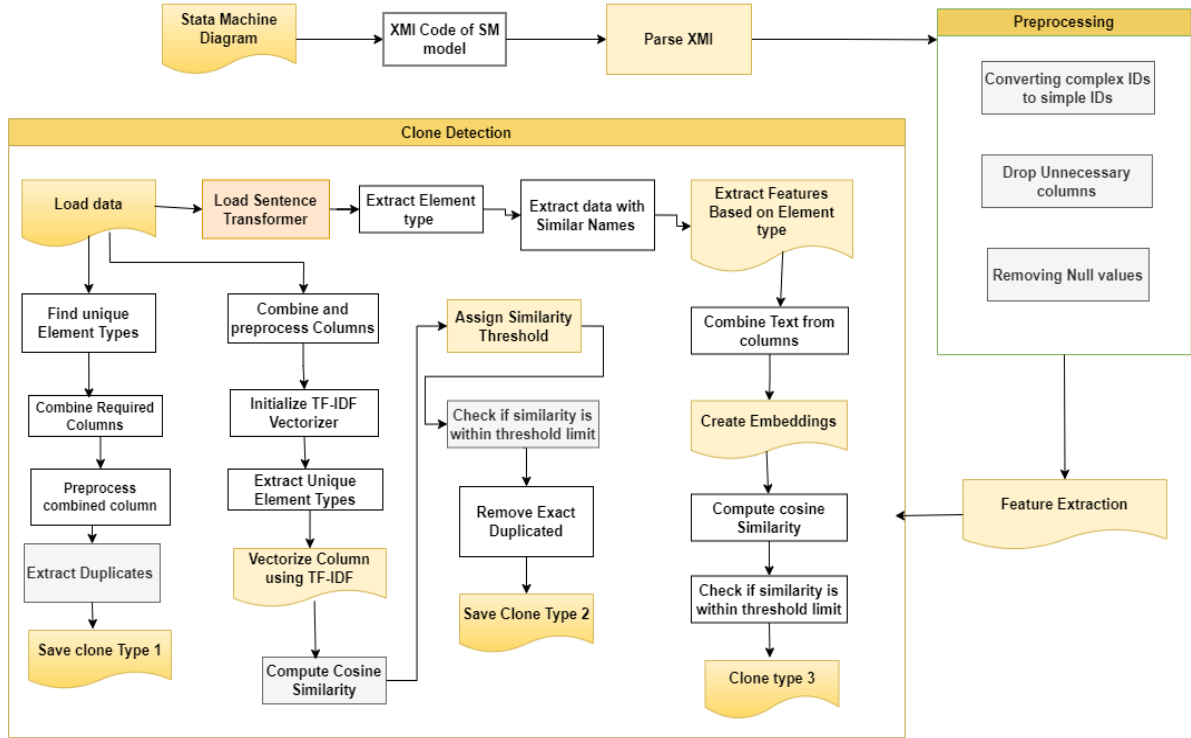


Figure 4. Proposed Workflow

using cosine similarity. It measure is used to determine how similar two statements are, based on the angle between their corresponding vector representations. If the similarity score is high, it indicates potential clones. Furthermore, a threshold is defined, if the similarity score is within the limits of the defined threshold, that pair is categorized as model clone type-2.

For model clones of type-3 another approach of NLP i.e., sentence transformer is used. The detection of type-3 model clones is more complex than type-1 and type-2 model clones. Therefore, sentence transformer is used since they are designed to capture the meaning of the text at a deeper level than simple word counts or term frequencies. They use models pre-trained on large datasets to generate embeddings that reflect the semantic content of text. This ability allows them to detect clones that are not textually identical but are semantically similar. A sentence transformer is a type of model used in natural language processing (NLP) that transforms sentences into meaningful, fixed-size numerical representations or vectors. These models are based on transformer architectures, a type of deep learning.

Our framework uses *sentence-transformers/all-MiniLM-L6-v2* model to create embeddings of the data. It is a compact and efficient transformer-based model developed by the Sentence Transformers library, which is based on the original MiniLM architecture. This model is designed specifically for generating semantically meaningful embeddings for sentences and paragraphs in a variety of languages. After

creating the embeddings of the extracted features of the state-machine model, cosine similarity between the embeddings is calculated to extract the similar statements. A threshold limit is applied to check the similarity between statements based on how closely two statements are linked i.e., check semantic similarity. If the embeddings fall into the defined threshold limit they are categorized as model clone type-3. After extracting the clones by using UMCD, the results are then compared to the manually assigned labels, and the accuracy of the proposed framework is calculated.

CHAPTER 4

IMPLEMENTATION

For the identification of clones in UML diagrams, our approach considers the detection of clones in UML state machines. As discussed in Chapter 3. State machine diagrams are of significant importance in the domain of software engineering, providing a clear, structured way to model and manage the dynamic behavior of systems. The diagram is created in the *papyrus tool*. Any other modeling tool can be used for creating a state machine diagram. A date set of state machines is created for clone detection in UML state machines. XMI code of state machines is used for extracting diagram features. Most of the tools used for creating UML models automatically generate XMI code for the diagram. The XMI code is then parsed to remove irrelevant data i.e., the XML-specific data or structural details of the model and to extract only relevant data and features essential for clone detection. Furthermore, relevant features are extracted from XMI code such as states, transitions, activities, constraints, etc. The extracted features are then represented in tabular format. Extracted data is further preprocessed to remove unnecessary columns, replace complex IDs with simpler ones, represent data in the same format such as in lowercase, and remove null values.

Table 4. List of Case Studies

Sr.	Case Study Title
1.	ATM System
2.	Elevator
3.	Telephone Line System
4.	Room reservation System
5.	Student Portal
6.	Online System
7.	Auto Crash Prevention System (ACPS)
8.	Arbiter
9.	Heating System
10.	Remotely Piloted Aircraft (RPA)
11.	Popcorn Machine
12.	Traffic lights System

Extracted data is then manually labeled to represent clone1, clone2, clone3, and non-clones as per definitions given in Table 2. Since our approach only focuses on the identification of model clones of types 1, 2, and 3. Our framework proposes an approach based on NLP for the identification of these clones. State machine diagrams of 12 case studies are used for model clone detection. Table 3. Presents the list of case studies used in this research for validation of the proposed framework.

4.1. Case Study 1:

ATM System:

The ATM case study characterizes the detailed working of the Automated Teller Machine (ATM) machine.

The behavior of the ATM case study is as follows:

- ATM remains idle initially, when the user inserts his card (insert card state) the system checks if the card is valid or not.
- If the card is valid, the system asks the user to enter the PIN (Give PIN). If the pin is valid a menu “Select-Service” appears on the screen and the user selects the services represented as states in the state machine i.e., Cash-Withdraw, Balance-Inquiry, Transfer-Funds, Generate ATM card, Bill payment, change pin or quit.
- If the user selects Cash-Withdraw system asks the user to insert the amount, if the amount is valid cash is ejected (eject cash), a receipt is printed (Print-receipt) and then the card is ejected (eject card). If the user selects ‘Balance-Inquiry’ the user's account balance is displayed as ‘View-Balance’ and a receipt is printed (Print-Receipt), if the user has no more functions card is ejected (eject card).
- If the user selects ‘Transfer-Funds’ system asks the user to ‘Enter-Receiver-Account’ and then insert the amount, if the amount is valid cash is transferred (transfer cash) a receipt is printed (Print-receipt) and then the card is ejected (eject card).
- If the user selects ‘Generate ATM card’ the system asks the user to input user credentials and Account number. If provided data is valid card is generated, otherwise the request is denied.
- If the user selects ‘change pin’ the system asks the user to enter ‘previous pin’ and ‘Enter New pin’ if both meet the system requirements new pin is generated otherwise the request is denied.
- If the user selects the ‘Bill Payment’ option, the system asks the user to select bill type, and mode of payment and enter the bill amount if the data is valid, and the user has a sufficient account balance the transaction is completed otherwise the transaction is terminated.

The UML state machine of the ATM system is given in Figure 5.

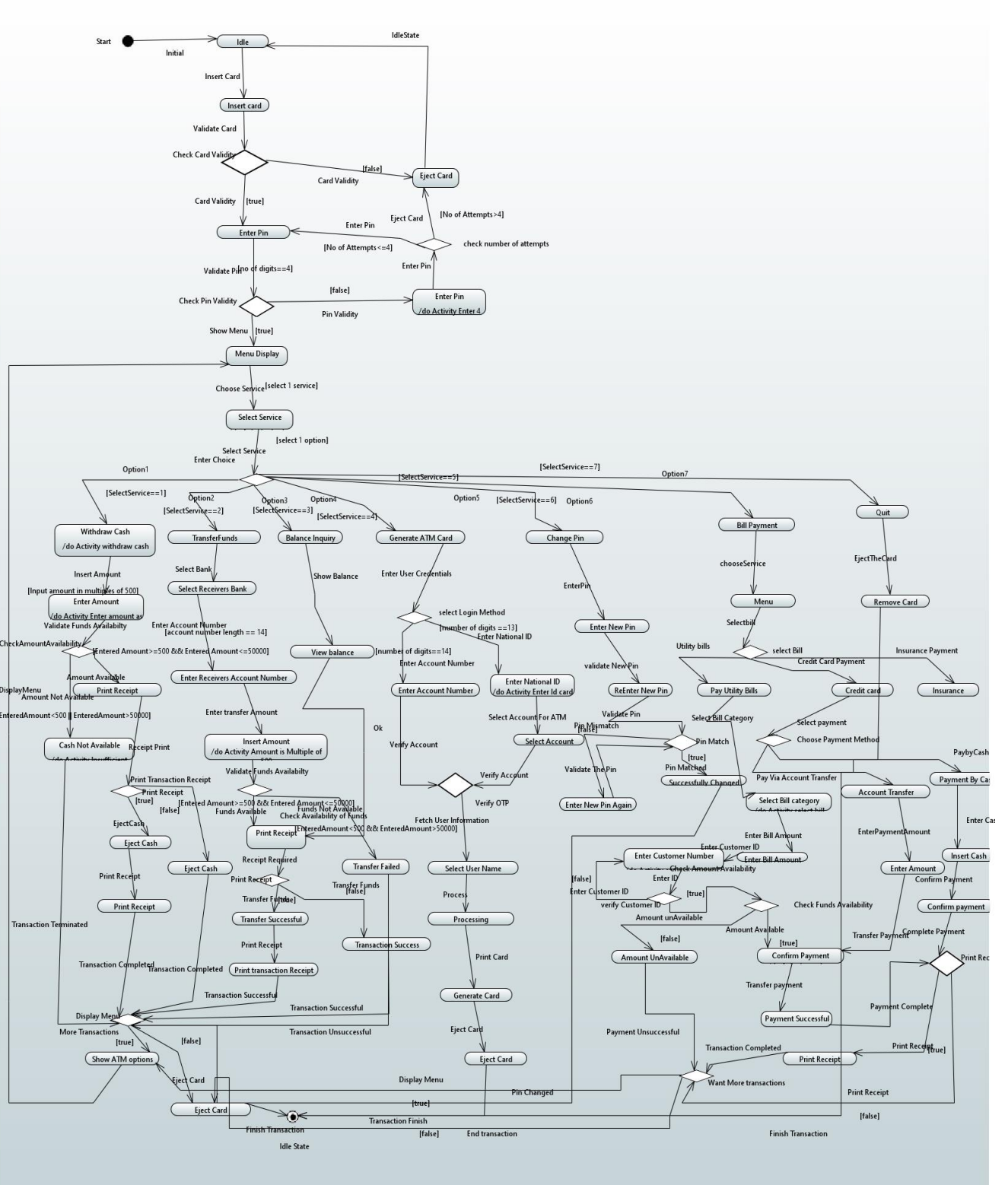


Figure 5. ATM System State Machine

4.2. Case Study 2:

Elevator:

The implementation of a modern elevator system characterizes the elevator design used for moving people and possessions across various levels of a building. The elevator has a load limit. Therefore, to determine the total load use a weight sensor.

The behavioral requirements of an elevator are as follows:

- The initial State is an “IDLE” state.
- The person presses the button to open the elevator. The weight is calculated as someone enters the elevator.
- If the weight is greater than the maximum allowed weight then it returns to the “IDLE” state and the alarm rings.
- Otherwise, the elevator will move toward checking the chosen floor.
- The elevator selects to move up or down as per the target floor choice by the person.
- Subsequently, the elevator keeps checking if it is the desired floor until it, reaches it after that it moves back to the initial state (IDLE).

The UML state Machine diagram of the Elevator case study is given in Figure 6.

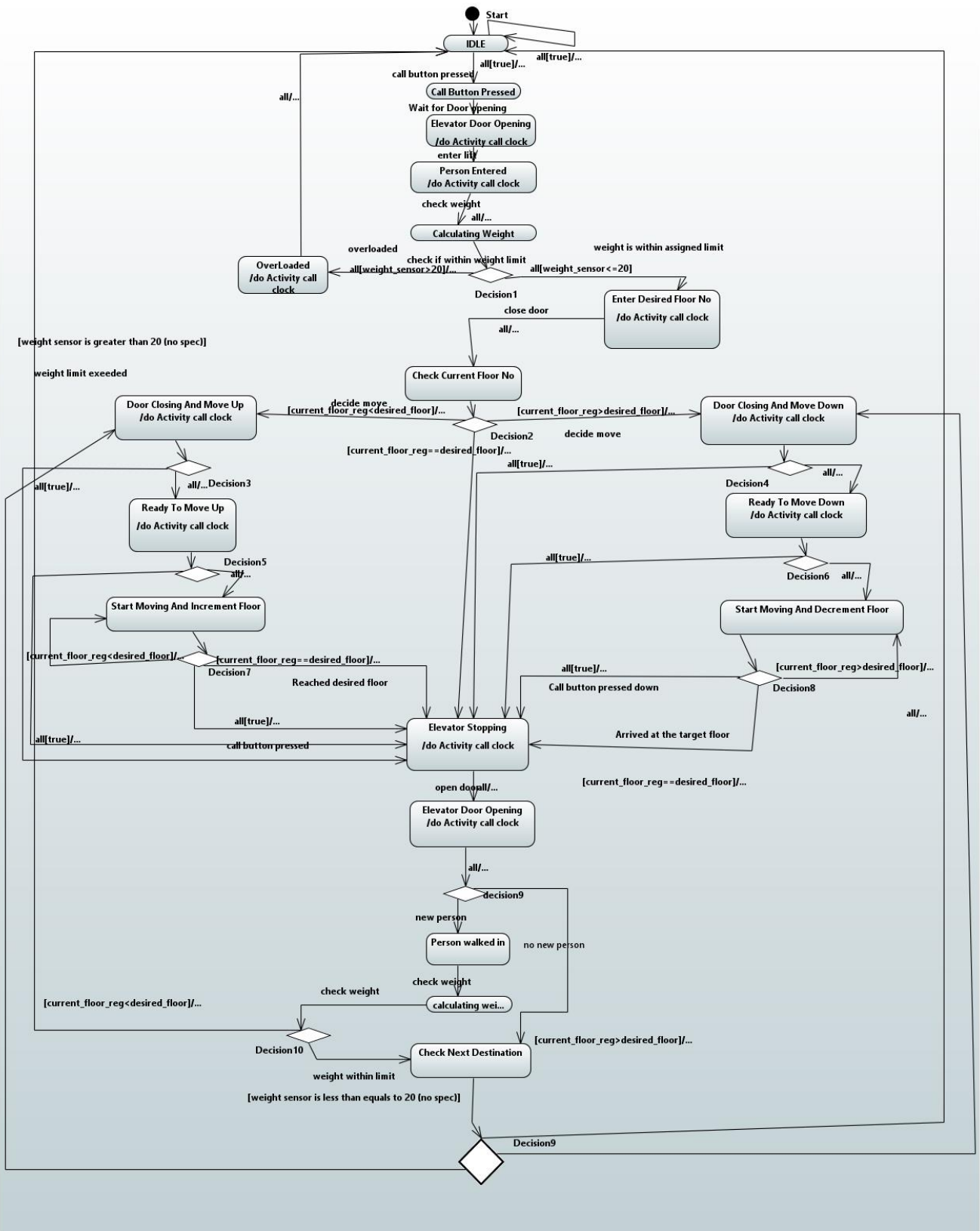


Figure 6. Elevator

4.3. Case Study 3:

Telephone line:

The Telephone line case study characterizes the behavior of a telephone system representing all stages of a call connection.

The behavior of the telephone line case study is as follows:

- The initial state of the system is idle.
- As soon as the user picks up the receiver from the hook, a dial tone rings and allows the entering of digits of the phone number.
- If the number is not dialed in the given time interval it moves to the time out state and than to the idle state else it checks for connection.
- If call is connected it starts ringing, it user is busy, the system moves to busy state and a busy message is played. Else if call is rejected it again shows busy status.
- If the dialed number is not valid a recorded message is played else system tries to connect a call & routes it to its destination.
- If the call is answered, the conversation begins and continues until the call is hanged up.
- Finally, phone call is disconnected and it returns to idle state.

The state machine diagram of the Telephone line vase study is given in Figure 7.

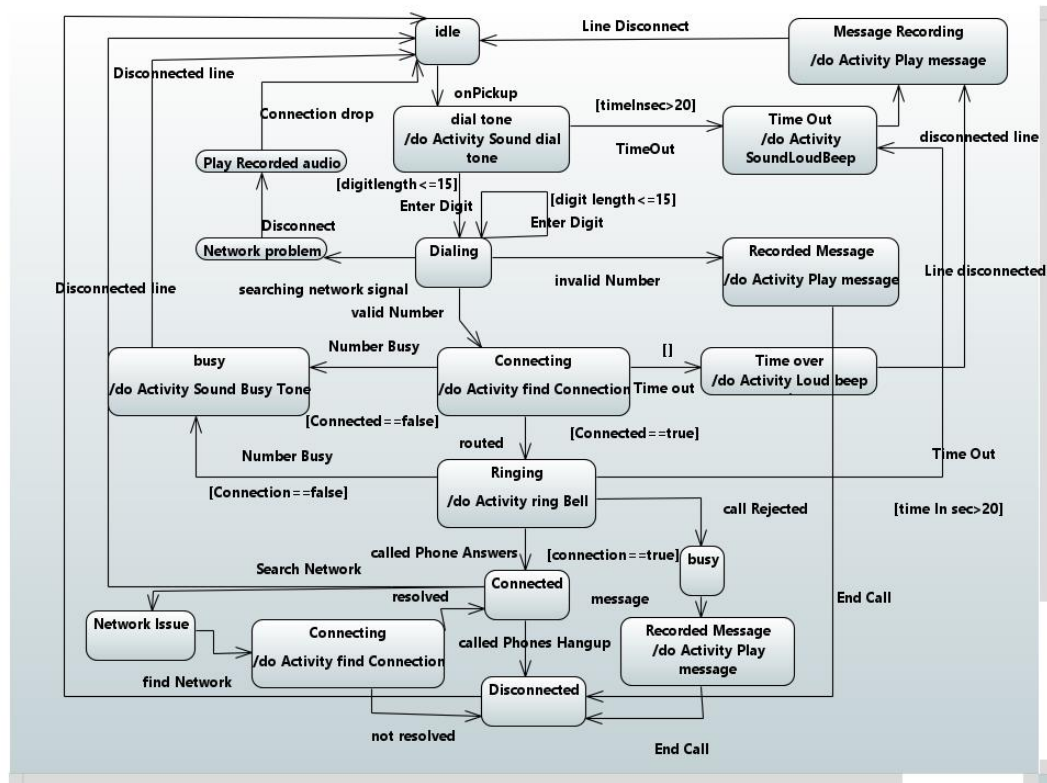


Figure 7. Telephone line

4.4. Case Study 4:

Room Reservation System:

A room reservation system is used for booking a room in a hotel. A person can reserve a room in the hotel based on its availability and if it matches the customer's requirements.

The case study has the following behavior:

- The user creates an account and logs in to the system.
- A list of available rooms is displayed in “Display-Rooms”.
- A person chooses the room as per his requirement “Choose-Room”. After choosing the room availability of the desired room is checked if not available he is redirected to the “Choose-Room” state. If available he is asked to enter the required reservation details “Input-Reservation-Details”.
- Further payment code ‘Generate Payment Code’ is displayed for the user to make payment ‘Make-Payment’ after the payment is validated an ‘invoice’ is generated.
- After successful payment, booking confirmation is sent to the user ‘Confirm-Reservation’.

The State Machine of the Room Reservation system is given in Figure 8.

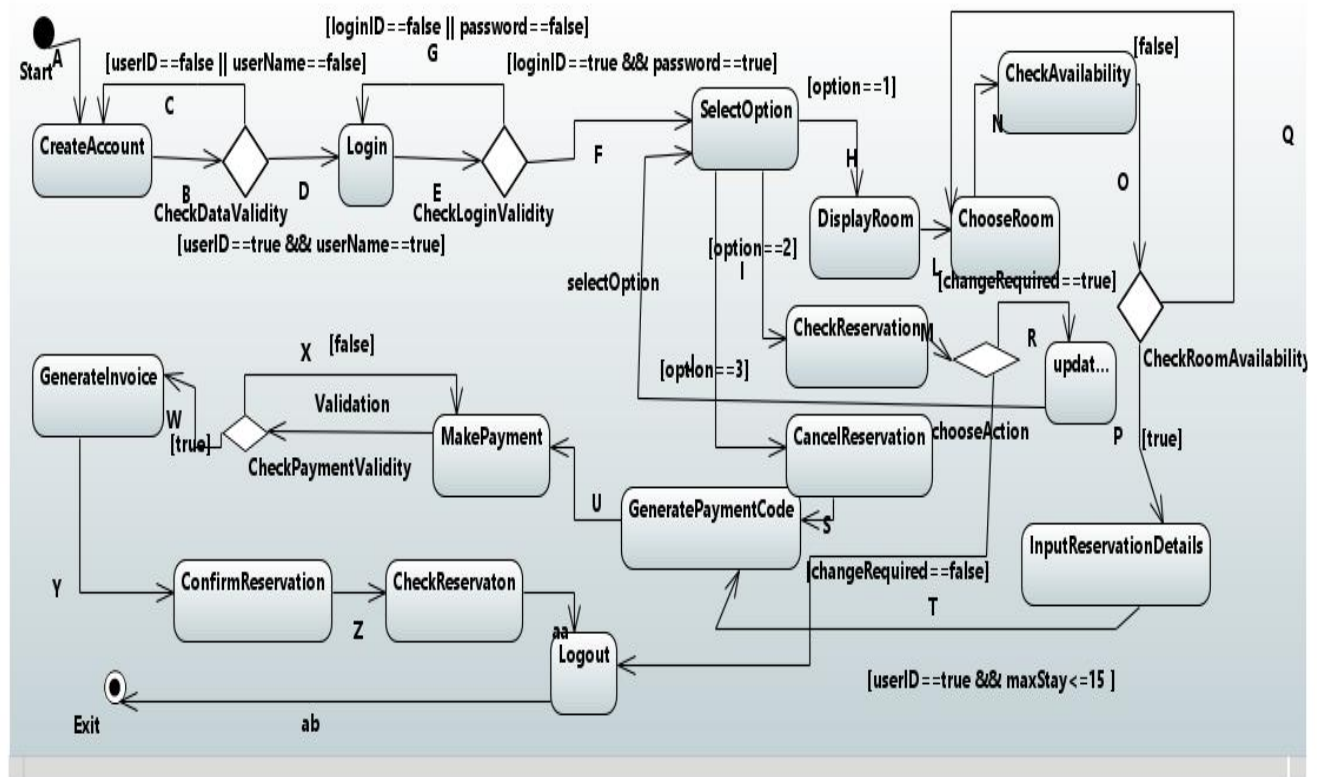


Figure 8. Room Reservation System

4.5. Case Study 5:

Student Portal:

A student portal is an online platform that provides students with access to a variety of academic and administrative activities such as viewing timetables, test assessments, new assignments, etc.

Following are the behavioral specifications of the student interface of the student portal.

- As the system starts login page displays (Login Dialog).
- After login credentials are entered their validity is tested if they are incorrect user is returned to the login page to enter the credentials again. Else if, the password is correct password expiry is checked if it is expired user is again returned to the login page to update the password.
- If the password is valid, the system enters the 'authorized' state and the 'Student profile' is displayed.
- From the profile students can user to edit data by entering 'User Details' and return to student profile for other operations.
- Student can take the test by entering 'Test-Assessment-Window' and enter 'Test-Results-Window' after he/she finishes the test or time is over 'timeout'.
- Students can also choose to get test results by entering 'Test-Results-Window' directly from 'Student-Profile'.
- After the session expires system will redirect to 'Login-Dialog' System will exit by entering the finish state.

Figure 9. Presents the state machine diagram of the student portal case study.

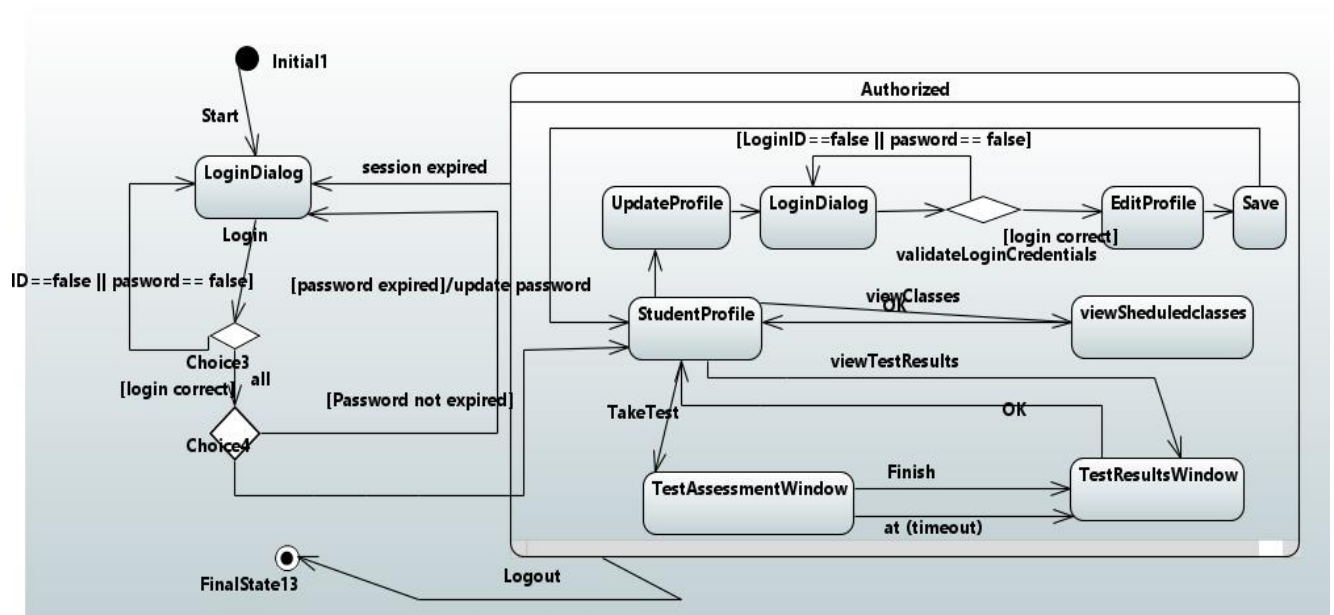


Figure 9. Student Portal

4.6. Case Study 6:

Online shopping

An online shopping website provides their users an easy access to view, choose and purchase items of their choice from comfort of their home.

The behavioral features of online-shopping case study are as follows:

- Initially after opening, the app/website one logs in to the system (Login) if already registered.
- If not registered then chose to register to the system first (Register) and then login to the system.
- After successful login one may want to search new products or view cart.
- If one choses to search some product (Search Products) he either finds the required product or not if the product is not found one can chose to search again or exit the system else if product is found net state is view product (View Item).
- If it is the desired product one can decide to add it to the cart (Add to cart) else search again by returning to the 'search products' state. After adding the product to the cart one can search for more products or choose to view the cart.
- After that one can search for more products or choose to remove items from the cart and can again decide to 'search products', 'view cart', or checkout from the system.

The following state machine diagram represents the case study of the online shopping system.

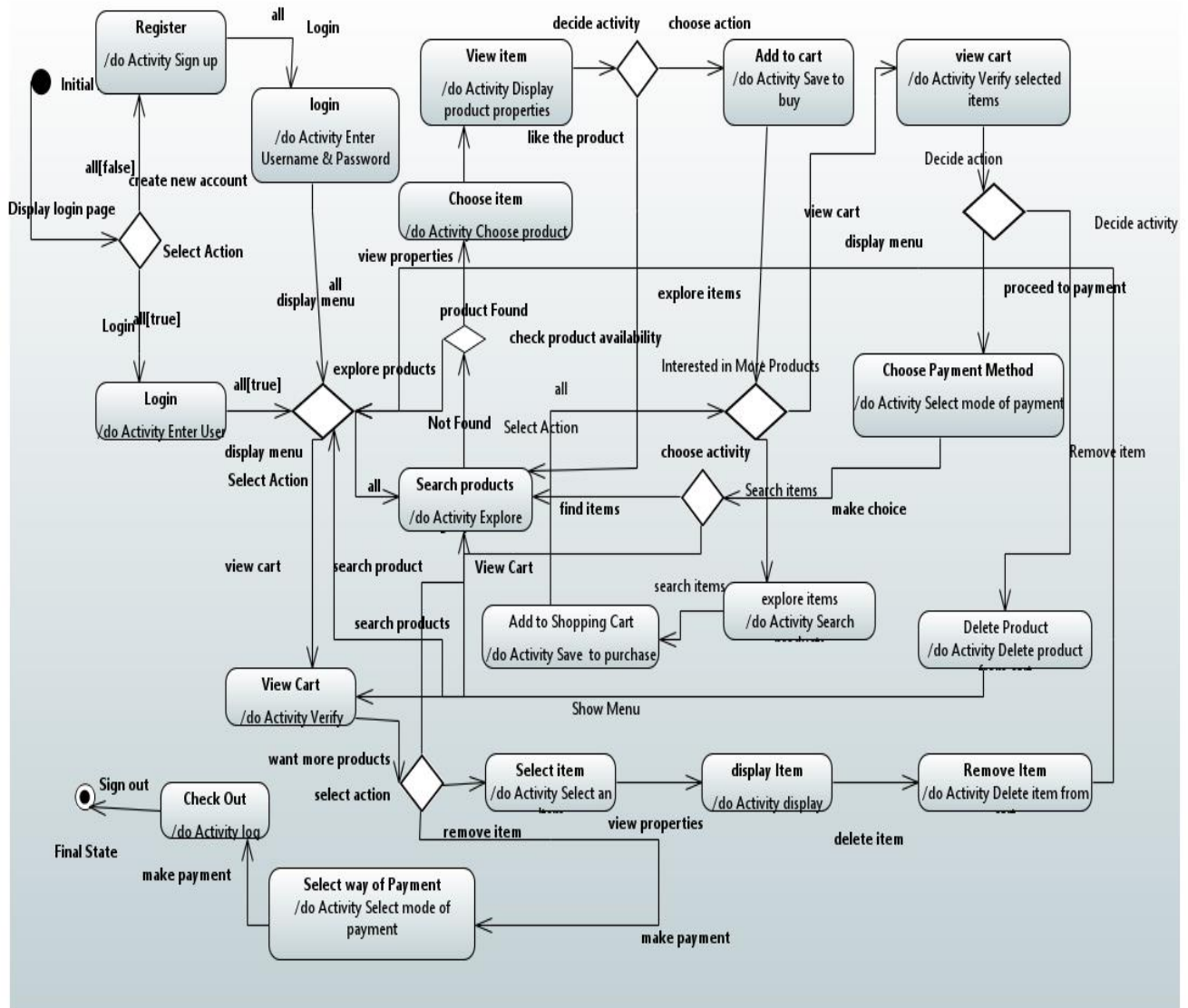


Figure 10. Online shopping

4.7. Case Study 7:

Auto Crash Prevention System (ACPS):

An Auto-crash prevention system is an intelligent technology that is designed to avoid or minimize the severity of a vehicle accident.

The behavioral requirements of an auto-crash prevention system are given below:

- Initially, the system is in an “IDLE” state.
- As the vehicle is moving or activated system ACP system goes to “Start” state.

- On the positive edge of the clock the ACPS transitions to a “Driving-Safely” state. In the next clock cycle, the system checks the activation of the obstacle detection modules i.e., camera and radar system.
- As the radar activates, the system transitions to an “Emitting-waves” state. The received signals from the radar are sent to the main controller continuously as per defined clock cycles.
- If the radar module detects the presence of some obstacle, the system changes to the “Obstacle-Detected” state.
- As the camera module is activated, the system goes to the “Take-Pictures” state. And the data is transferred to the controller.
- From the “Obstacle-Detected” state, the system then enters the “Obstacle-Detection-Module” state for estimating the distance between the vehicle and the obstacle.
- The system enters the “Imminent-Collision-Strategy-composite” state, in case of a distance less than 5 meters. In this case, the system transfers to the “Notification” state, and generates an alarm then goes to the “Warning” state. Subsequently, moves to an “Emergency-Stop” state by applying the emergency breaks.
- The ACPS goes to “Near-Collision-Avoidance-Strategy-composite-state” if the estimated distance is above 5 meters and less than 10 meters. In this case, the system goes to the “warning” state and notifies the system to assert a “normal-brakes” signal.
- The ACPS transfers to the “Changing-Lane-Strategy-composite-state”, in case of distance above 10 meters and less than 20 meters. Then the system chooses from two available options “auto” or “manual” turn. On activation of the “deviation-sensor,” the system goes to the “Auto-Turn” state, and “lane-change-alarm” is activated to inform the driver about auto lane change and then goes to the “Lane-Changed-Successfully” state. In the second case, the driver manually chooses to switch lanes and then the system goes to the “Lane-Changed-Successfully” state.

State-Machine of the ACPS system [38] is given in Figure 11:

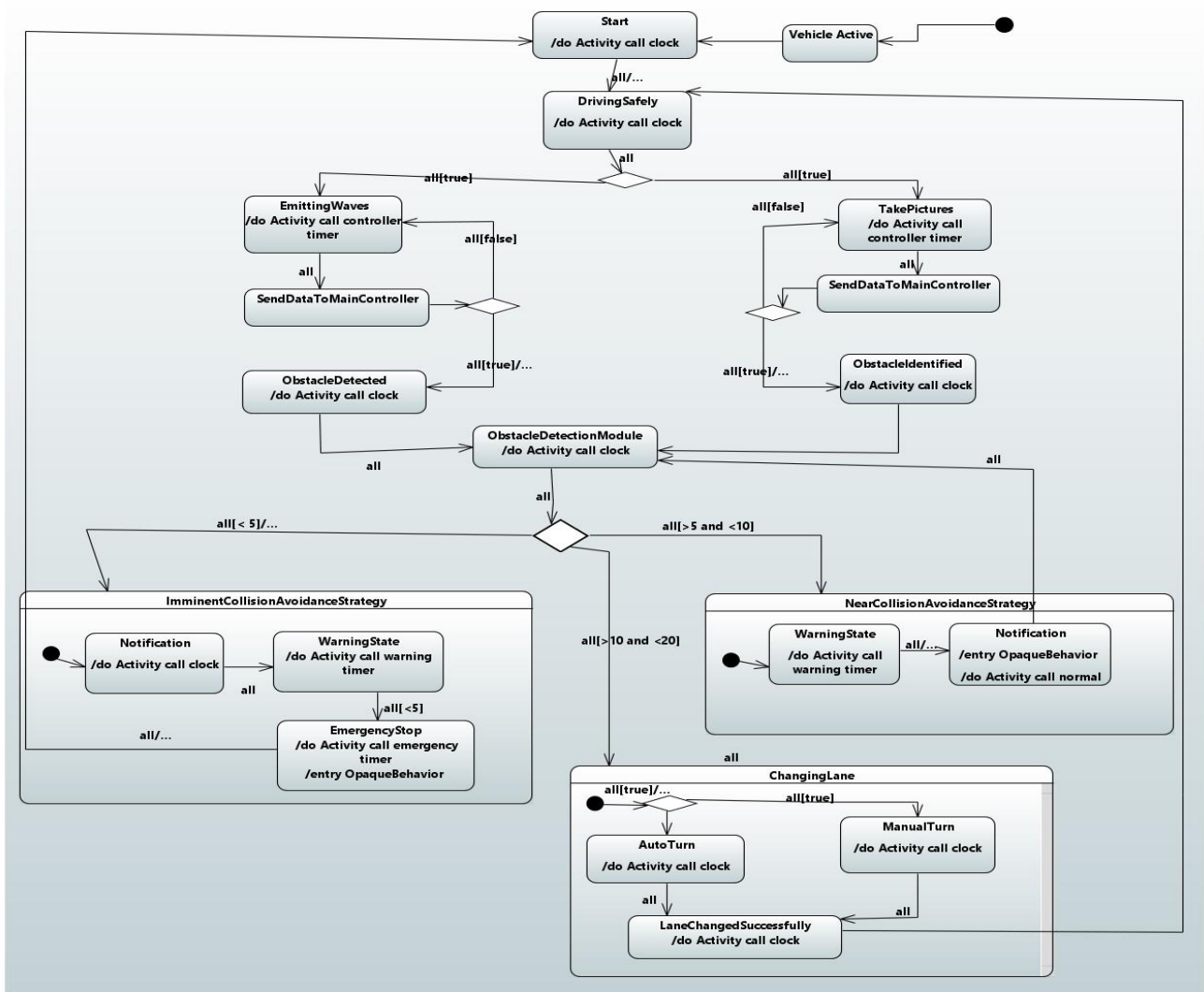


Figure 11. Automatic Crash Prevention System

4.8. Case Study 8:

Arbiter:

The arbiter is the control unit that regulates access to the shared resource such as bus, memory, or any other resource. It determines which master gets access at any given time based on a predefined protocol or priority scheme. In this case study the design of an arbiter, as one-hot coding style state machine with 7 possible states is presented [38]. The states of the arbiter include “Master-1”, “Master-2”, “Master-3”, “Idle”, “Idle-1”, “Idle-2” and “Idle-3”. Each of the master devices can request for the assignment of resources such as for bus grant. All the masters can make the request simultaneously as well. The arbiter determines the priority for the bus grant by using the Round-Robin policy. Once a “Master” gains access to the bus, it can carry out specific transactions for a specified time limit and then release the bus for processing requests from other master devices.

The following are the behavioral requirements of an arbiter state machine:

- Initially, the state machine is in an “Idle” state. The system waits for the master devices to request for bus grant.
- The system goes to the “Idle” state on “reset”.
- The arbiter used a Round-robin policy to assign buses to the master devices. This technique maintains a pointer or counter that cycles through the list of masters sequentially. If all the master devices request bus grants simultaneously, then the arbiter will assign the arbiter to the “Master-1” device.
- When “Master-1” completes its task, it transitions to the “Idle-1” state signaling the completion status.
- From the “Idle-1” state, the arbiter then assigns the bus to “Master-2” and the process continues sequentially, in the order of “Master-2”, “Master-3”, and then again to “Master-1” if it requests for the bus again.

State-Machine is given in Figure 12.

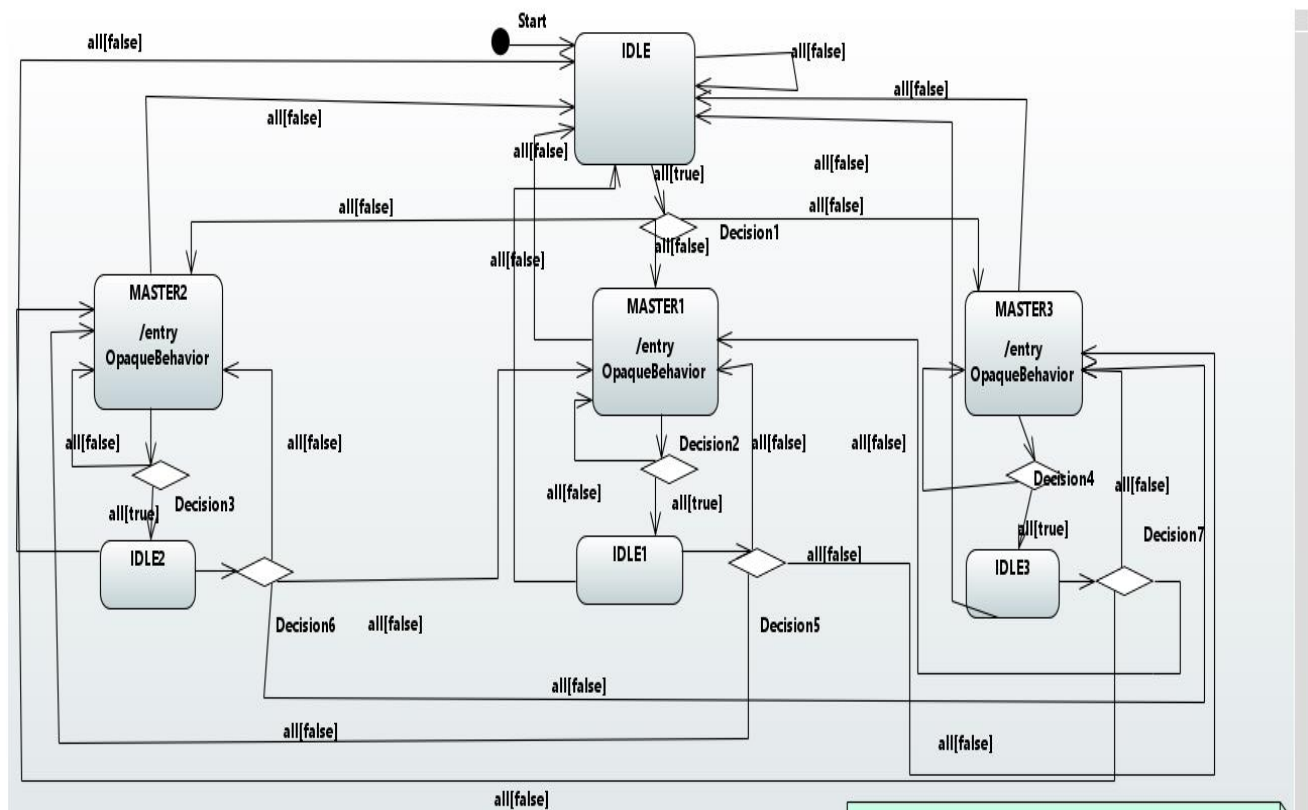


Figure 12. Arbiter

4.9. Case Study 9:

Heating System:

A heating system maintains the temperature at a specific temperature b heating or cooling operations as per requirements.

Behavioral requirements of the heating system case study are as follows.

- Initially when turned on the system is in idle state.
- If the temperature is too cold i.e., the temperature is lower than the desired temperature.
- System enters the ‘Heating’ state that composes of ‘initializing’ state that transitions to ‘Active’ state when ‘ready’.
- When temperature is as per desired system is set to idle state. If the temperature becomes too hot, the system transitions to ‘Cooling’ state or once can chose to shut down the system.
- Else if the temperature is too hot i.e., the temperature is higher than the desired temperature.
- System enters the ‘Cooling’ state. When the temperature is as per the desired temperature, the system is set to idle state. If the temperature becomes too cold then the system transitions to a ‘Heating’ state or one can choose to shut down the system.
- System can also transition to final state i.e., ‘Shutdown’ from ‘idle’ state.

The state machine diagram of the heating system is given in Figure 13.

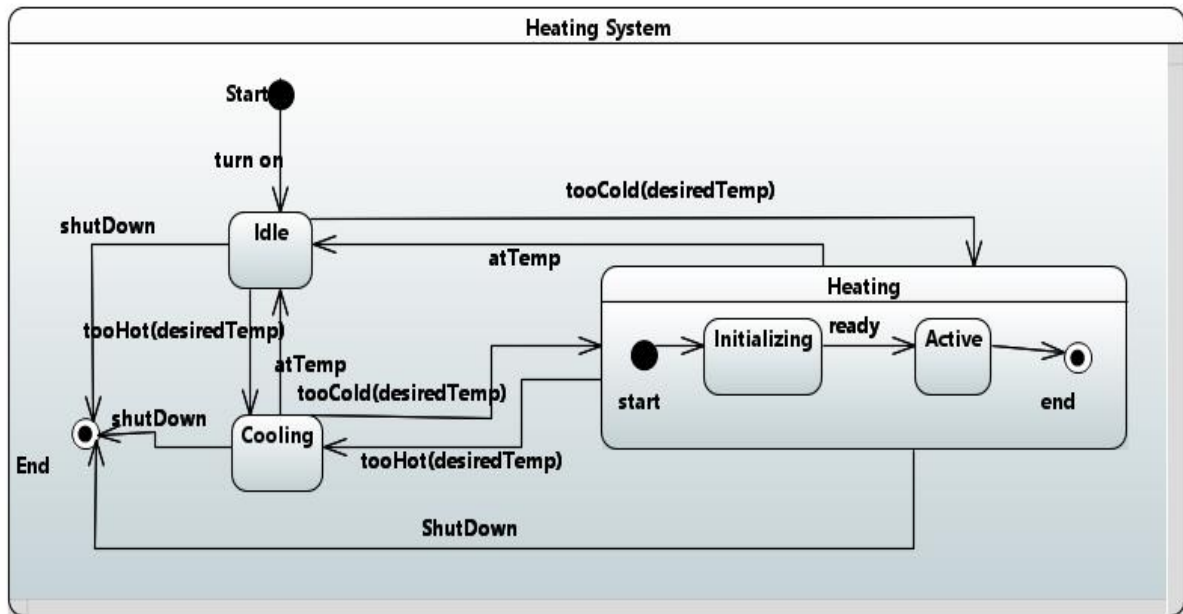


Figure 13. Heating System

4.10. Case Study 10:

Remotely Piloted Aircraft (RPA):

The RPV also called a drone, is a remotely controlled aircraft. RPA system can be preprogrammed to fly autonomously as per flight plans. It is equipped with intricate dynamic automation systems. This model of the RPA system while considering different safety restrictions is showcased in this case study [38].

The simplified behavioral requirements of an RPA system are given below:

- In the beginning, the initial state of the drone is “Flying”. The fault detection system comprises of several sensors and continuously monitors the Engine and GPS systems for possible failures.
- The fault monitor continuously receives values from sensors and identifies if there is any fault. If the sensors detects any failure then the system moves to either “Engine-Failure” state or “GPS-failure” state based on values from the sensors).
- If the system encounters engine malfunctioning and moves to “Engine-Failure” state, the clock cycle is activated and the RPA is directed to emergency.
- If the system sends a termination command i.e., “Termination-Command-Received” state, the system enters the “Flight-Termination-Initiated” state and then transitions to the “Manually-Land-Aircraft” state.
- If the system encounters GPS failure, if GPS auto restores successfully restores GPS, normal flight of the RPA system continues. Otherwise, it is directed to move to the “Flight-Back-To-Station” state and then to the “Reached-Back-To-Station” state. In case the aircraft does not reach the station in the estimated time, the system transitions to the “Aircraft-Lost” state.
- Fault monitoring system of the RPA continuously monitors other faults factor as well such as: “Termination-Command-Received”, “2.4 GHz-Link-Failure”, “Soft-Geofence-Breach” and “Data-link-failure” states via associated sensors.

The state machine of the RPA system case study is given in Figure 14.

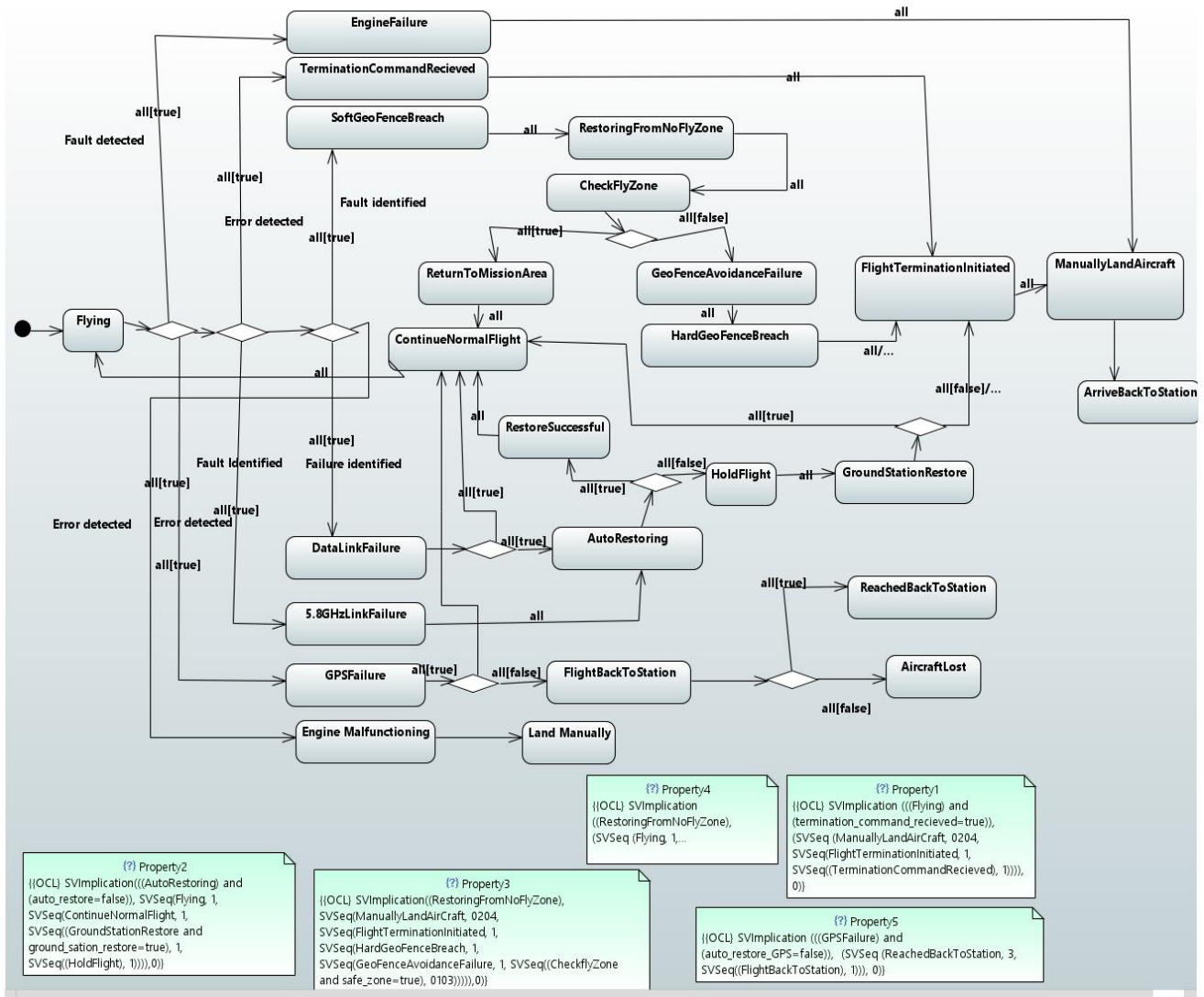


Figure 14. Remotely Piloted Aircraft (RPA)

4.11. Case Study 11:

Popcorn machine:

A popcorn maker is a kitchen appliance designed to pop popcorn kernels into fluffy, edible popcorn. The popcorn case study has the following behavior:

- At first, the popcorn maker is in an idle mode.
- Once the popcorn maker is switched on it enters the Running mode state machine switches to the “ready” state then it transitions to the ‘On’ state as it starts (making popcorn).
- When the time is up it enters the ‘Done state’.
- After it transitions to the off switch, the running state is exited and the machine enters the ‘Off’ state.

The state machine of the popcorn machine is given in Figure 15.

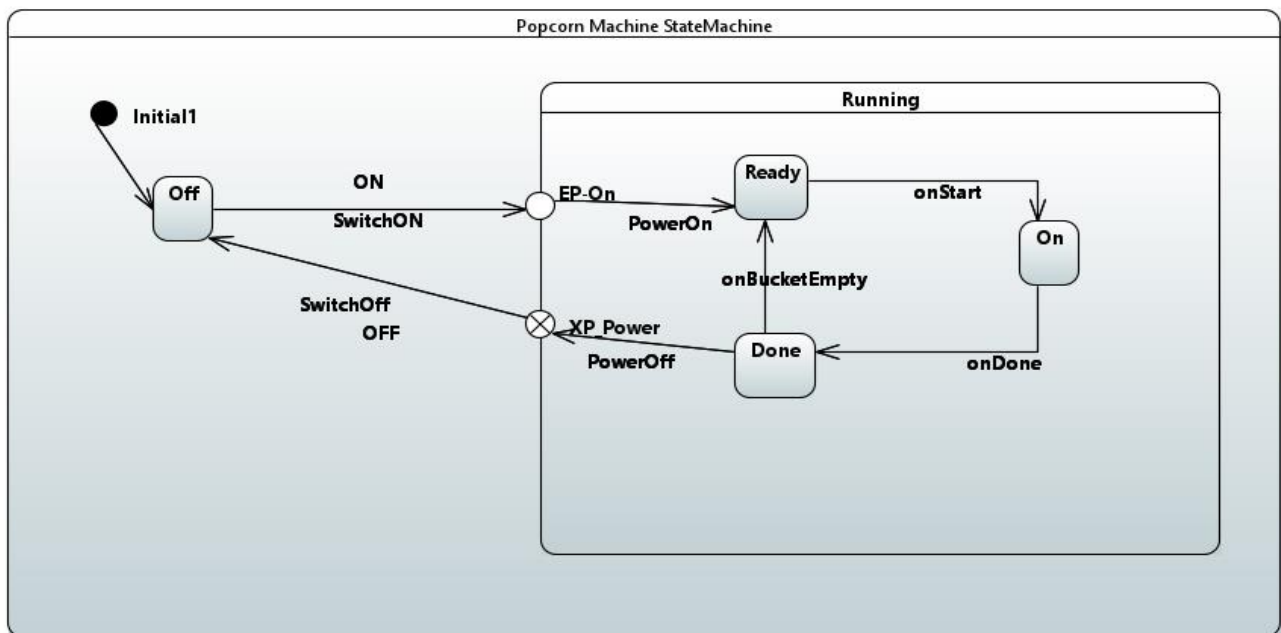


Figure 15. Popcorn machine

4.12. Case Study 12:

Traffic Lights:

The case study of Traffic lights characterizes the control unit activities of a traffic light system for maintaining the proper flow of traffic on North-South Road (N-S) and East-West Road (E-W). The N-S road is the central road with a high traffic rate, therefore more time is allotted to the N-S road that is green light stays 'ON' for a greater time-period. For E-W traffic, an electromagnetic sensor is installed in the road surface to sense the vehicle's presence. If the E-W sensor detects any vehicle it checks for an N-S traffic light. As soon as the time allowed for the green light reaches its limit. The yellow light for both N-S and E-W is turned on and then the green light of the E-W road is turned "ON". In the case of an emergency vehicle, as soon as the emergency sensor is activated the traffic on both N-S and E-W roads is stopped by turning the Red light "ON" to allow the emergency vehicle to pass. Sensors and cameras are also installed at intersections to monitor traffic light violations [38].

State machine diagram, models following behavioral features of traffic lights control unit:

- Initially, the green light on N-S road stays "ON".
- IF the sensor on E-W detects the presence of a vehicle, a signal is sent to the controller and the time limit for the green light on N-S roads is checked. If its limit is reached yellow light is turned on for both roads followed by a Red light on the N-S road and a Green light on the E-W road to allow the traffic on the E-W road to pass.
- After the sensor on the E-W road is deactivated, the Green light on the N-S road is turned "ON".

- Upon controller reset, the Greenlight time for N-S will be set to zero.
- If there is an emergency vehicle, the sensor is activated, and “Greenlight” will turn “Yellow” and then to “Red” to allow the emergency vehicle to pass.
- After the emergency vehicle has passed the signal from the E-W sensor is monitored if the vehicle is present the Green light is turned “ON” for the E-W road and the N-S light stays red until the time limit for E-W is reached.
- A camera continuously monitors traffic light violations. It captures a picture of the vehicle in case of a violation and saves the record for further action for traffic rule violation at the traffic signal.

The state machine of the case study of the traffic light control unit is given in Figure 16.

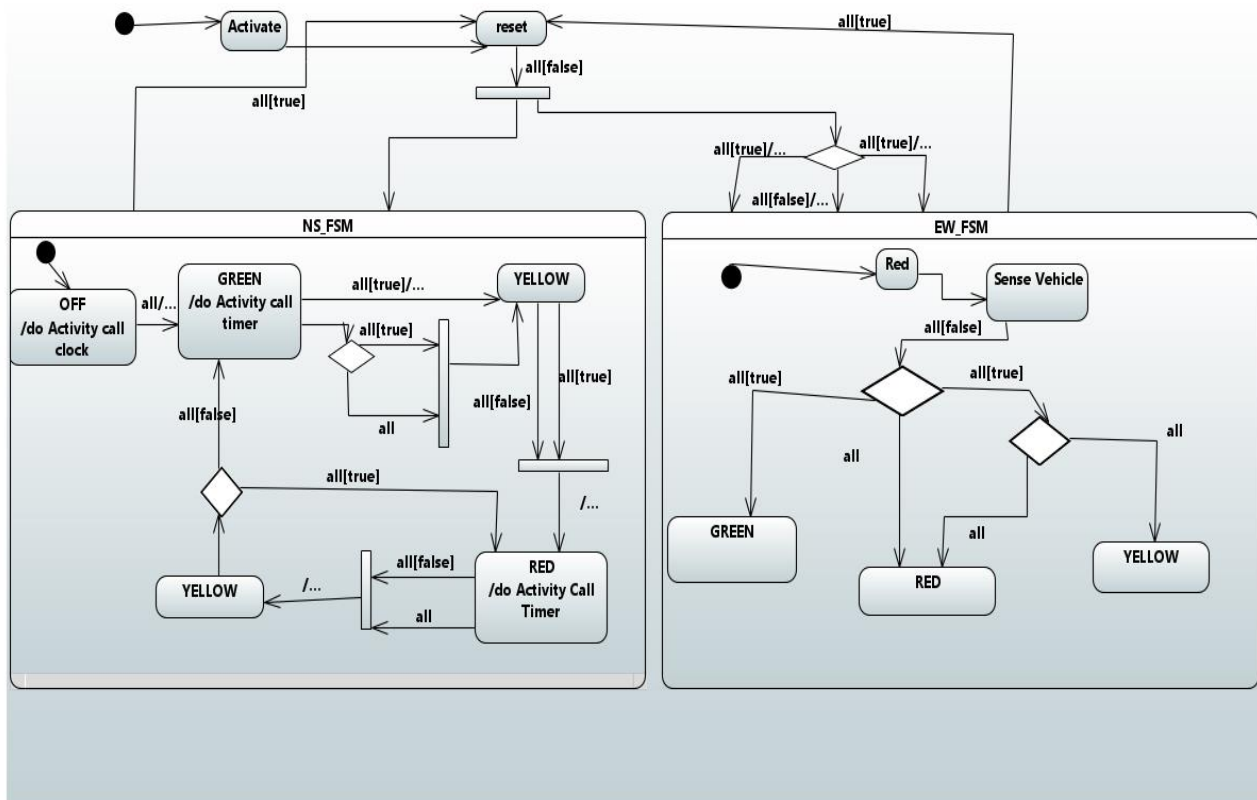


Figure 16. Traffic Lights

4.13. Implementation of UMCD for Clone detection in the ATM System case study.

The case study comprises a UML state machine of an ATM that describes the behavior of an ATM machine as shown in Figure 6. The state machine is developed in the *Eclipse Papyrus* tool and is exported in XMI format using the built-in facilities of the *Papyrus tool*. The tree representation to visualize the hierarchy of the ATM state machine showing states, transitions, and their relation and dependency is shown below in Figure 17. and a section of the XMI code of the ATM System State-machine is given in Figure 18.

Since the XMI code of the state machine textually represents the diagram. Therefore, we can easily access the relevant features of the UML model. Since the XMI code of the ATM system contains a lot of XML-specific information that is irrelevant for identifying model clones. It also contains a lot of structural details about the state-machine diagram that plays no significant role in the clone detection process.

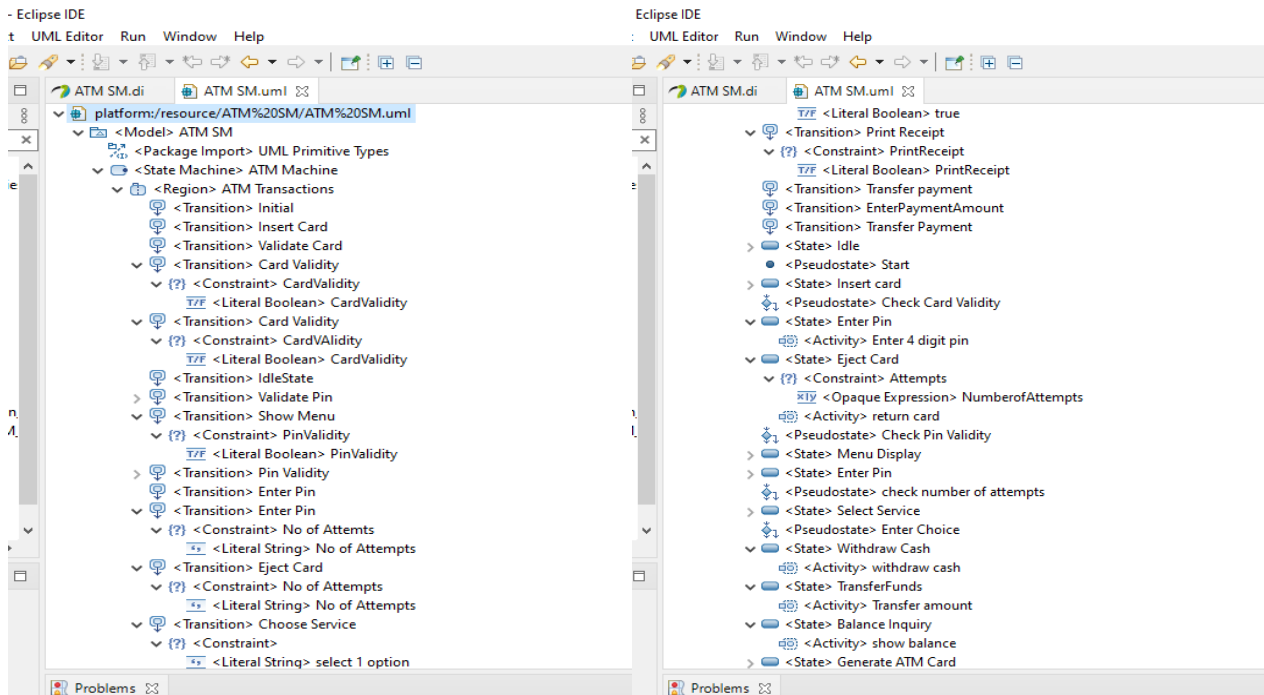


Figure 17. Tree Representation of ATM System

Therefore, it is important to extract only relevant features and information from the XMI code of the model. For this purpose, we used 'xml.etree.ElementTree' library for parsing XML documents in Python. It is a Python module used for XML data processing. It is a simple and effective way of manipulating XML documents such as parsing to extract relevant features by navigating through the XML tree structure.

```

<?xml version="1.0" encoding="UTF-8"?>
<uml:Model xmi:version="20131001" xmlns:xmi="http://www.omg.org/spec/XMI/20131001" xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" xmlns:uml="http://www.eclipse.org/uml/2.0.0" xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" xmlns:uml="http://www.eclipse.org/uml/2.0.0">
  <packageImport xmi:type="uml:PackageImport" xmi:id="oVv2UJjEe6NHdZbcPR96Q">
    <importPackage xmi:type="uml:Model" href="pathmap://UML_LIBRARIES/UMLPrimitiveTypes.library.uml#_0"/>
  </packageImport>
  <packagedElement xmi:type="uml:StateMachine" xmi:id="oUUS8JjnEe6NHdZbcPR96Q" name="ATM Machine">
    <region xmi:type="uml:Region" xmi:id="oVDSwJjnEe6NHdZbcPR96Q" name="ATM Transactions">
      <transition xmi:type="uml:Transition" xmi:id="x2WNYJjoEe6NHdZbcPR96Q" name="Initial" source="_pVUngJjoEe6NHdZbcPR96Q" target="_mSS7YJjoEe6NHdZbcPR96Q"/>
      <transition xmi:type="uml:Transition" xmi:id="ho88QJjpEe6NHdZbcPR96Q" name="Insert Card" source="_mSS7YJjoEe6NHdZbcPR96Q" target="_vHYFsjjoEe6NHdZbcPR96Q"/>
      <transition xmi:type="uml:Transition" xmi:id="kETkYJjpEe6NHdZbcPR96Q" name="Validate Card" source="_vHYFsjjoEe6NHdZbcPR96Q" target="_92e-oJjoEe6NHdZbcPR96Q"/>
      <transition xmi:type="uml:Transition" xmi:id="_590OUJjpEe6NHdZbcPR96Q" name="Card Validity" guard="_UlyGYJqEe6NHdZbcPR96Q" source="_92e-oJjoEe6NHdZbcPR96Q" target="_92e-oJjoEe6NHdZbcPR96Q"/>
      <ownedRule xmi:type="uml:Constraint" xmi:id="UlyGYJqEe6NHdZbcPR96Q" name="Card Validity">
        <specification xmi:type="uml:LiteralBoolean" xmi:id="UlyGYJqEe6NHdZbcPR96Q" name="Card Validity" value="true"/>
      </ownedRule>
    </region>
    <transition xmi:type="uml:Transition" xmi:id="5xWA4JmfEe6NHdZbcPR96Q" name="Card Validity" guard="_IO-vlJmgEe6NHdZbcPR96Q" source="_92e-oJjoEe6NHdZbcPR96Q" target="_92e-oJjoEe6NHdZbcPR96Q"/>
    <ownedRule xmi:type="uml:Constraint" xmi:id="IO-vlJmgEe6NHdZbcPR96Q" name="Card Validity">
      <specification xmi:type="uml:LiteralBoolean" xmi:id="IO-vlJmgEe6NHdZbcPR96Q" name="Card Validity"/>
    </ownedRule>
    <transition xmi:type="uml:Transition" xmi:id="rILAoJmgEe6NHdZbcPR96Q" name="Idle State" source="_W8CRLJqEe6NHdZbcPR96Q" target="_mSS7YJjoEe6NHdZbcPR96Q"/>
    <transition xmi:type="uml:Transition" xmi:id="C2DtMjmhEe6NHdZbcPR96Q" name="Validate Pin" guard="_RPuQLOtEe6zSb434dTwqA" source="_2Ny5UJjpEe6NHdZbcPR96Q" target="_2Ny5UJjpEe6NHdZbcPR96Q"/>
    <ownedRule xmi:type="uml:Constraint" xmi:id="_RPuQLOtEe6zSb434dTwqA">
      <specification xmi:type="uml:LiteralString" xmi:id="_RPuQLOtEe6zSb434dTwqA" name="no of digits" value="no of digits=4"/>
    </ownedRule>
    <transition xmi:type="uml:Transition" xmi:id="POLJcJmhEe6NHdZbcPR96Q" name="Show Menu" guard="_TYItgJmjEe6NHdZbcPR96Q" source="_51RCLJmgEe6NHdZbcPR96Q" target="_51RCLJmgEe6NHdZbcPR96Q"/>
    <ownedRule xmi:type="uml:Constraint" xmi:id="TYItgJmjEe6NHdZbcPR96Q" name="Pin Validity">
      <specification xmi:type="uml:LiteralBoolean" xmi:id="TYItgZmjEe6NHdZbcPR96Q" name="Pin Validity" value="true"/>
    </ownedRule>
    <transition xmi:type="uml:Transition" xmi:id="a54EJmhEe6NHdZbcPR96Q" name="Pin Validity" guard="_bE-dYJmjEe6NHdZbcPR96Q" source="_51RCLJmgEe6NHdZbcPR96Q" target="_51RCLJmgEe6NHdZbcPR96Q"/>
    <ownedRule xmi:type="uml:Constraint" xmi:id="bE-dYJmjEe6NHdZbcPR96Q" name="Pin Validity">
      <specification xmi:type="uml:LiteralBoolean" xmi:id="bE-dYJmjEe6NHdZbcPR96Q" name="Pin Validity" value="true"/>
    </ownedRule>
  </packagedElement>
</model>

```

Figure 18. XMI representation of ATM System State-Machine

Relevant features for the model clone detection in UML state machine diagram such as *Owned attributes, owned operations, sub vertexes, states, pseudo states, activities, constraints, IDs, transitions, source and target states* etc., are extracted and saved in a file. The output of XMI parsing is shown in Figure. 19.

```

[{'Etype': 'Property', 'name': 'cardvalidity', 'defaultVal': 'true'}]
[{'Etype': 'Property', 'name': 'amountvalidity', 'defaultVal': 'true'}]
[{'Etype': 'Property', 'name': 'time', 'defaultVal': 'true'}]
[{'Etype': 'Property', 'name': 'pinvalidity', 'defaultVal': 'true'}]
[{'Etype': 'Property', 'name': 'withdrawamount', 'defaultVal': None}]
[{'Etype': 'Property', 'name': 'totalamount', 'defaultVal': '5000'}]
[{'Etype': 'Property', 'name': 'selectservice', 'defaultVal': '2'}]
[{'Etype': 'operation', 'name': 'chooseAction'}]
[{'Etype': 'operation', 'name': 'checkwithdrawamountavailability'}]
[{'Etype': 'operation', 'name': 'initializeSystem'}]
[{'Etype': 'Transition', 'name': 'None', 'source': '_UptLILNkEeeb6TZRFZITQ', 'target': '_YXnnULNkEeeb6TZRFZITQ', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'EnterCard', 'source': '_YXnnULNkEeeb6TZRFZITQ', 'target': '_cf0y0LNkEeeb6TZRFZITQ', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'acceptCard', 'source': '_cf0y0LNkEeeb6TZRFZITQ', 'target': '_zfYe0LNkEeeb6TZRFZITQ', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'viewAccountBalance', 'source': '_rag_cLTjEee2Jf56mWoMLA', 'target': '_Krd54LtkEee2Jf56mWoMLA', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'PrintReceipt', 'source': '_Krd54LtkEee2Jf56mWoMLA', 'target': '_qTahcFtvEey_nqM3AIE4FQ', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'SelectAmount', 'source': '_alJZELTjEee2Jf56mWoMLA', 'target': '_ONMUMLTkEee2Jf56mWoMLA', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'ValidateAmount', 'source': '_ONMUMLTkEee2Jf56mWoMLA', 'target': '_jMnUclTkEee2Jf56mWoMLA', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'validAmount', 'source': '_jMnUclTkEee2Jf56mWoMLA', 'target': '_rXzNMLTkEee2Jf56mWoMLA', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'PrintReceipt', 'source': '_rXzNMLTkEee2Jf56mWoMLA', 'target': '_1rYUsFtuEey_nqM3AIE4FQ', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'invalidAmount', 'source': '_jMnUclTkEee2Jf56mWoMLA', 'target': '_0mBswLTjEee2Jf56mWoMLA', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'Exit', 'source': '_wG9gclTjEee2Jf56mWoMLA', 'target': '_0mBswLTjEee2Jf56mWoMLA', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'Invalidpin', 'source': '_Ah2PkLNIeeb6TZRFZITQ', 'target': '_0mBswLTjEee2Jf56mWoMLA', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'InvalidCard', 'source': '_zfYe0LNkEeeb6TZRFZITQ', 'target': '_0mBswLTjEee2Jf56mWoMLA', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'None', 'source': '_0mBswLTjEee2Jf56mWoMLA', 'target': '_YXnnULNkEeeb6TZRFZITQ', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'SelectService', 'source': '_ZRL6sPC8EeeJZKX0AKvBpA', 'target': '_alJZELTjEee2Jf56mWoMLA', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'SelectService', 'source': '_ZRL6sPC8EeeJZKX0AKvBpA', 'target': '_rag_cLTjEee2Jf56mWoMLA', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'Exit', 'source': '_ZRL6sPC8EeeJZKX0AKvBpA', 'target': '_wG9gclTjEee2Jf56mWoMLA', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'validPin', 'source': '_Ah2PkLNIeeb6TZRFZITQ', 'target': '_ZRL6sPC8EeeJZKX0AKvBpA', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'validCard', 'source': '_zfYe0LNkEeeb6TZRFZITQ', 'target': '_jXyJILNkEeeb6TZRFZITQ', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'validatePin', 'source': '_jXyJILNkEeeb6TZRFZITQ', 'target': '_Ah2PkLNIeeb6TZRFZITQ', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'Exit', 'source': '_1rYUsFtuEey_nqM3AIE4FQ', 'target': '_0mBswLTjEee2Jf56mWoMLA', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'SelectService', 'source': '_qTahcFtvEey_nqM3AIE4FQ', 'target': '_1Q_pUftvEey_nqM3AIE4FQ', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'SelectService', 'source': '_1Q_pUftvEey_nqM3AIE4FQ', 'target': '_ZRL6sPC8EeeJZKX0AKvBpA', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'Exit', 'source': '_1Q_pUftvEey_nqM3AIE4FQ', 'target': '_0mBswLTjEee2Jf56mWoMLA', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'SelectService', 'source': '_ZRL6sPC8EeeJZKX0AKvBpA', 'target': '_Zo0zSftzEey_nqM3AIE4FQ', 'body': 'time==3\r\n'}]
[{'Etype': 'Transition', 'name': 'EnterAccountNumber', 'source': '_Zo0zSftzEey_nqM3AIE4FQ', 'target': '_abhXwFtzEey_nqM3AIE4FQ', 'body': 'time==3\r\n'}]

```

Figure 19. Parsed XMI code of ATM Case Study

The parsed XML code of the ATM state machine is further preprocessed and the following tasks are performed:

- Complex IDs like “xmi:id=”_oVDSwJjnEe6NHdZbcPR96Q” are difficult to understand and comprehend, so it makes it difficult to process them, therefore, they are replaced with simplified IDs like “ID_33” Thus replacing them with simple IDs makes it easier to comprehend and process.
- Removing Null values to avoid complications in further steps.
- Removing any unnecessary columns that do not play any role in the clone detection process.
- If necessary representing text in the same format such as separating concatenated words and representing text in lowercase.
- Represent text in a suitable format, in this case, the extracted data is represented in a tabular form for easier understanding and application of the proposed approach.

Figure 20. Shows the output after performing preprocessing steps.

	A	B	C	D	E	F	G	H	I
1	Etype	name	id	source	target	constraint	Constraint_value	trigger	activity
2	region	ATM Transactions	_oVDSwJjnEe6NHdZbcPR96Q	NaN	NaN	NaN	NaN	NaN	NaN
3	Transition	Initial	_x2WNYJjoEe6NHdZbcPR96Q	_pVUngJjoEe6NHdZbcPR96Q	_mSS7YJjoEe6NHdZbcPR96Q	NaN	NaN	NaN	NaN
4	Transition	Insert Card	_ho88QJjpEe6NHdZbcPR96Q	_mSS7YJjoEe6NHdZbcPR96Q	_vHYFsjjoEe6NHdZbcPR96Q	NaN	NaN	NaN	NaN
5	Transition	Validate Card	_kETKYJjpEe6NHdZbcPR96Q	_vHYFsjjoEe6NHdZbcPR96Q	_92e-cJjoEe6NHdZbcPR96Q	NaN	NaN	NaN	NaN
6	Transition	Card Validity	_590OUJjpEe6NHdZbcPR96Q	_92e-cJjoEe6NHdZbcPR96Q	_2Ny5UJjpEe6NHdZbcPR96Q	CardValidity	TRUE	NaN	NaN
7	Transition	Card Validity	_5xWA4JmfEe6NHdZbcPR96Q	_92e-cJjoEe6NHdZbcPR96Q	_W8CRIJjoEe6NHdZbcPR96Q	CardValidity	None	NaN	NaN
8	Transition	IdleState	_rLlAoJmgEe6NHdZbcPR96Q	_W8CRIJjoEe6NHdZbcPR96Q	_mSS7YJjoEe6NHdZbcPR96Q	NaN	NaN	NaN	NaN
9	Transition	Validate Pin	_C2DtMjmhEe6NHdZbcPR96Q	_2Ny5UJjpEe6NHdZbcPR96Q	_51RCIJmgEe6NHdZbcPR96Q	None	no of digits==4	NaN	NaN
10	Transition	Show Menu	_P0LJcmhEe6NHdZbcPR96Q	_51RCIJmgEe6NHdZbcPR96Q	_JGE5WjmhEe6NHdZbcPR96Q	PinValidity	TRUE	NaN	NaN
11	Transition	Pin Validity	_a5f4EJmhEe6NHdZbcPR96Q	_51RCIJmgEe6NHdZbcPR96Q	_Soww8JmhEe6NHdZbcPR96Q	PinValidity	None	NaN	NaN
12	Transition	Enter Pin	_kaZpwJmlEe6NHdZbcPR96Q	_Soww8JmhEe6NHdZbcPR96Q	_kKTD4JmkEe6NHdZbcPR96Q	NaN	NaN	NaN	NaN
13	Transition	Enter Pin	_otz_sJmlEe6NHdZbcPR96Q	_kKTD4JmkEe6NHdZbcPR96Q	_2Ny5UJjpEe6NHdZbcPR96Q	No of Attempts	No of Attempts<=4	NaN	NaN
14	Transition	Eject Card	_v8w5SjmlEe6NHdZbcPR96Q	_kKTD4JmkEe6NHdZbcPR96Q	_W8CRIJjoEe6NHdZbcPR96Q	No of Attempts	No of Attempts>4	NaN	NaN
15	Transition	Choose Service	_aG-nQJmnEe6NHdZbcPR96Q	_JGE5WjmhEe6NHdZbcPR96Q	_MndHEJmnEe6NHdZbcPR96Q	None	select 1 service	NaN	NaN
16	Transition	Select Service	_1ZbEwJmnEe6NHdZbcPR96Q	_MndHEJmnEe6NHdZbcPR96Q	_rBLeKJmnEe6NHdZbcPR96Q	None	select 1 option	NaN	NaN
17	Transition	Option1	_w1KZ8JmoEe6NHdZbcPR96Q	_rBLeKJmnEe6NHdZbcPR96Q	_DUGP4JmoEe6NHdZbcPR96Q	SelectService	SelectService==1	NaN	NaN
18	Transition	Option2	_xjccQJmoEe6NHdZbcPR96Q	_rBLeKJmnEe6NHdZbcPR96Q	_GUv9QJmoEe6NHdZbcPR96Q	SelectService	SelectService==2	NaN	NaN
19	Transition	Option3	_yNv9gJmoEe6NHdZbcPR96Q	_rBLeKJmnEe6NHdZbcPR96Q	_HFJlQJmoEe6NHdZbcPR96Q	None	SelectService==3	NaN	NaN
20	Transition	Option4	_y3xK4JmoEe6NHdZbcPR96Q	_rBLeKJmnEe6NHdZbcPR96Q	_HnU_oJmoEe6NHdZbcPR96Q	SelectService	SelectService==4	NaN	NaN
21	Transition	Option5	_0J1PsJmoEe6NHdZbcPR96Q	_rBLeKJmnEe6NHdZbcPR96Q	_IHvIEJmoEe6NHdZbcPR96Q	SelectService	SelectService==5	NaN	NaN
22	Transition	Option6	_OpGjOJmpEe6NHdZbcPR96Q	_rBLeKJmnEe6NHdZbcPR96Q	_JHEKJmpEe6NHdZbcPR96Q	SelectService	SelectService==6	NaN	NaN
23	Transition	Option7	_PVyDoJmpEe6NHdZbcPR96Q	_rBLeKJmnEe6NHdZbcPR96Q	_MlIz8JmpEe6NHdZbcPR96Q	SelectService	SelectService==7	NaN	NaN

Figure 20. Parsed XMI code after further pre-processing

After the feature extraction and preprocessing, the next step is labeling. The extracted data of ATM state machines are then thoroughly analyzed as per the definition of clones in UML state machines as defined in Table 2. Moreover, clones are manually assigned labels such as clone1, clone2, clone3, and nonclone respectively as shown in Figure 21. Furthermore, Figure 22. Highlights clone pairs off all three types of clones in ATM state machine for easier understanding.

Subsequently, the clone detection approach is employed to the data for the identifying clones. The results of type-1 model clones are displayed in Figure 23. Type-2 model clone outcomes are presented in Figure 24 and Figure 25 presents the out of type-3 Model clones.

	A	B	C	D	E	F	G	H	I	J
1	Etype	name	id	source	target	constraint	Constraint_value	trigger	activity	Label
2	region	ATM Transactions	ID_33	ID_44	ID_44	time	time == 5	NaN	execute transaction	nonclone
3	region	ATM Transactions	ID_86	ID_77	ID_67	time	time == 5	NaN	execute transaction	clone1
4	Transition	Initial	ID_176	ID_18	ID_68	time	time == 5	NaN	execute transaction	nonclone
5	Transition	Insert Card	ID_100	ID_68	ID_53	time	time == 5	NaN	execute transaction	nonclone
6	Transition	Validate Card	ID_27	ID_53	ID_40	time	time == 5	NaN	execute transaction	nonclone
7	Transition	Card Validity	ID_139	ID_40	ID_17	Card Validity	TRUE	NaN	execute transaction	clone2
8	Transition	Card Validity	ID_130	ID_40	ID_7	Card Validity	None	NaN	execute transaction	clone2
9	Transition	Idle State	ID_10	ID_7	ID_68	time	time == 5	NaN	execute transaction	nonclone
10	Transition	Validate Pin	ID_150	ID_17	ID_45	None	no of digits==4	NaN	execute transaction	nonclone
11	Transition	Show Menu	ID_25	ID_45	ID_5	Pin Validity	TRUE	NaN	execute transaction	nonclone
12	Transition	Pin Validity	ID_17	ID_45	ID_35	Pin Validity	None	NaN	execute transaction	clone2
13	Transition	Enter Pin	ID_2	ID_35	ID_10	time	time == 5	NaN	execute transaction	nonclone
14	Transition	Enter Pin	ID_154	ID_10	ID_17	No of Attempts	No of Attempts<=4	NaN	execute transaction	clone2
15	Transition	Idle State	ID_88	ID_84	ID_89	time	time == 5	NaN	execute transaction	clone1
16	Transition	Eject Card	ID_82	ID_10	ID_7	No of Attempts	No of Attempts>4	NaN	execute transaction	nonclone
17	Transition	Choose Service	ID_49	ID_5	ID_23	None	select 1 service	NaN	execute transaction	nonclone
18	Transition	Select Service	ID_89	ID_23	ID_29	None	choose 1 option	NaN	execute transaction	clone3
19	Transition	Option1	ID_24	ID_29	ID_73	Select Service	Select Service==1	NaN	execute transaction	nonclone
20	Transition	Option2	ID_52	ID_29	ID_60	Select Service	Select Service==2	NaN	execute transaction	clone2
21	Transition	Option3	ID_173	ID_29	ID_26	None	Select Service==3	NaN	execute transaction	clone2
22	Transition	Option4	ID_30	ID_29	ID_12	Select Service	Select Service==4	NaN	execute transaction	clone2
23	Transition	Option5	ID_5	ID_29	ID_32	Select Service	Select Service==5	NaN	execute transaction	clone2

Figure 21. Labeled Data

Etype	name	id	source	target	constraint	Constraint_value	trigger	activity	Label
region	ATM Transactions	ID_33	ID_44	ID_44	time	time == 5	NaN	execute transaction	nonclone
region	ATM Transactions	ID_86	ID_77	ID_67	time	time == 5	NaN	execute transaction	clone1
Transition	Initial	ID_176	ID_18	ID_68	time	time == 5	NaN	execute transaction	nonclone
Transition	Insert Card	ID_100	ID_68	ID_53	time	time == 5	NaN	execute transaction	nonclone
Transition	Validate Card	ID_27	ID_53	ID_40	time	time == 5	NaN	execute transaction	nonclone
Transition	check Card Validity	ID_139	ID_40	ID_17	time	time == 5	NaN	execute transaction	clone2
Transition	check Card Validity	ID_130	ID_40	ID_7	time	time == 5	NaN	execute transaction	clone2
Transition	Idle State	ID_10	ID_7	ID_68	time	time == 5	NaN	execute transaction	nonclone
Transition	Validate Pin	ID_150	ID_17	ID_45	None	no of digits==4	NaN	execute transaction	nonclone
Transition	Show Menu	ID_25	ID_45	ID_5	Pin Validity	TRUE	NaN	execute transaction	nonclone
Transition	Pin Validity	ID_17	ID_45	ID_35	Pin Validity	None	NaN	execute transaction	clone2
Transition	Enter Pin	ID_2	ID_35	ID_10	time	time == 5	NaN	execute transaction	nonclone
Transition	Enter Pin	ID_154	ID_10	ID_17	time	time == 5	NaN	execute transaction	clone1
Transition	Idle State	ID_88	ID_84	ID_89	time	time == 5	NaN	execute transaction	clone1
Transition	Eject Card	ID_82	ID_10	ID_7	No of Attempts	No of Attempts>4	NaN	execute transaction	nonclone
Transition	Choose Service	ID_49	ID_5	ID_23	None	select 1 service	NaN	execute transaction	nonclone
Transition	Select Service	ID_89	ID_23	ID_29	None	choose 1 option	NaN	execute transaction	clone3
Transition	Option1	ID_24	ID_29	ID_73	Select Service	Select Service==1	NaN	execute transaction	nonclone
Transition	Option2	ID_52	ID_29	ID_60	Select Service	Select Service==2	NaN	execute transaction	clone2
Transition	Option3	ID_173	ID_29	ID_26	None	Select Service==3	NaN	execute transaction	clone2
Transition	Option4	ID_30	ID_29	ID_12	Select Service	Select Service==4	NaN	execute transaction	clone2
Transition	Option5	ID_5	ID_29	ID_32	Select Service	Select Service==5	NaN	execute transaction	clone2

Figure 22. Highlighting clones in the Data

	A	B	C	D	E	F	G	H	I
1	Etype	name	id	source	target	constraint	Constraint_value	activity	Label
2	region	ATM Transactions	ID_33	ID_44	ID_44	time	time == 5		nonclone
3	region	ATM Transactions	ID_86	ID_77	ID_67	time	time == 5		clone1
4	State	Confirm Payment	ID_58	ID_44	ID_44	time		confirm payment	nonclone
5	State	Confirm payment	ID_75	ID_44	ID_44	time		confirm payment	clone1
6	State	Eject Card	ID_19	ID_44	ID_44	time		Release card	clone1
7	State	Eject Card	ID_41	ID_44	ID_44	time		Release card	clone1
8	State	Eject Card	ID_7	ID_44	ID_44	time		release card	nonclone
9	State	Eject Cash	ID_70	ID_44	ID_44	time		Release funds	clone1
10	State	Eject Cash	ID_16	ID_44	ID_44	time		Release funds	nonclone
11	State	Enter Pin	ID_35	ID_44	ID_44	time		Enter 4 digit pin	clone1
12	State	Enter Pin	ID_17	ID_44	ID_44	time		Enter 4 digit pin	nonclone
13	State	Print Receipt	ID_43	ID_44	ID_44	time		Generate Receipt	clone1
14	State	Print Receipt	ID_39	ID_44	ID_44	time		Generate Receipt	clone1
15	State	Print Receipt	ID_25	ID_44	ID_44	time		Generate Receipt	nonclone
16	State	Print Receipt?	ID_4	ID_44	ID_44	time		check if receipt is required	clone1
17	State	Print Receipt?	ID_36	ID_44	ID_44	time		check if receipt is required	nonclone
18	Transition	Choose Service	ID_49	ID_5	ID_23		select 1 service		nonclone
19	Transition	Choose Service	ID_200	ID_92	ID_87		select 1 service		clone1
20	Transition	Display Menu	ID_125	ID_22	ID_20	Want More Tran	TRUE		clone1
21	Transition	Display Menu	ID_85	ID_3	ID_20	Want More Tran	TRUE		clone1
22	Transition	Enter Pin	ID_177	ID_32	ID_28	time			clone1
23	Transition	Enter Pin	ID_2	ID_35	ID_10	time			nonclone

Figure 23. Clone Type 1 Results

	A	B	C	D	E	F	G	H	I	J
2	Transition	Validate Card	ID_27	ID_53	ID_40	time	time == 5		execute transaction	nonclone
3	Transition	Card Validity	ID_139	ID_40	ID_17	Card Validity	TRUE		execute transaction	clone2
4	Transition	Card Validity	ID_130	ID_40	ID_7	Card Validity			execute transaction	clone2
5	Transition	Validate Pin	ID_150	ID_17	ID_45		no of digits==4		execute transaction	nonclone
6	Transition	Pin Validity	ID_17	ID_45	ID_35	Pin Validity			execute transaction	clone2
7	Transition	Validate Pin	ID_84	ID_71	ID_8	time	time == 5		execute transaction	clone1
8	Transition	validate New Pin	ID_109	ID_28	ID_71	time	time == 5		execute transaction	clone2
9	Transition	Validate The Pin	ID_95	ID_14	ID_8	time	time == 5		execute transaction	clone2
10	Transition	Show Menu	ID_25	ID_45	ID_5	Pin Validity	TRUE		execute transaction	nonclone
11	Transition	Display Menu	ID_125	ID_22	ID_20	Want More Transacti	TRUE		execute transaction	clone2
12	Transition	Eject Card	ID_142	ID_51	ID_19	time	time == 5		execute transaction	nonclone
13	Transition	Eject The Card	ID_163	ID_77	ID_46	time	time == 5		execute transaction	clone2
14	Transition	Choose Service	ID_49	ID_5	ID_23		select 1 service		execute transaction	nonclone
15	Transition	Select Service	ID_89	ID_23	ID_29		choose 1 option		execute transaction	clone3
16	Transition	Choose Service	ID_200	ID_92	ID_87		select 1 service		execute transaction	clone1
17	Transition	choose Service	ID_114	ID_49	ID_65	time	time == 5		execute transaction	clone2
18	Transition	Option1	ID_24	ID_29	ID_73	Select Service	Select Service==1		execute transaction	nonclone
19	Transition	Option2	ID_52	ID_29	ID_60	Select Service	Select Service==2		execute transaction	clone2
20	Transition	Option3	ID_173	ID_29	ID_26		Select Service==3		execute transaction	clone2
21	Transition	Option4	ID_30	ID_29	ID_12	Select Service	Select Service==4		execute transaction	clone2
22	Transition	Option6	ID_94	ID_29	ID_49	Select Service	Select Service==6		execute transaction	clone2
23	Transition	Option5	ID_5	ID_29	ID_32	Select Service	Select Service==5		execute transaction	clone2
24	Transition	Option7	ID_88	ID_29	ID_77	Select Service	Select Service==7		execute transaction	clone2

Figure 24. Clone Type 2 Results

	A	B	C	D	E	F	G	H	I
1	Etype	name	id	source	target	constraint	Constraint_value	activity	Label
2	Transition	Transaction Unsuccessful	ID_101	ID_2	ID_22	time	funds transfer failed	execute transaction	nonclone
3	Transition	End transaction	ID_73	ID_3	ID_41	Want More Transaction	close transaction	execute transaction	clone3
4	Transition	Enter transfer Amount	ID_28	ID_34	ID_48	time	enter an amount that is a multiple of 500	execute transaction	clone3
5	Transition	Enter Payment Amount	ID_7	ID_55	ID_76	time	inserted payment Amount>=500 and is <=500	execute transaction	clone3
6	Transition	Funds Not Available	ID_118	ID_24	ID_2	Check Availability	Entered Amount<500 Entered Amount>50	execute transaction	clone2
7	Transition	Payment Unsuccessful	ID_136	ID_72	ID_3	time	transfer of amount unsuccessful	execute transaction	clone3
8	Transition	Acquire user credentials	ID_18	ID_78	ID_66	time	Obtain user details	execute transaction	clone3
9	Transition	Payment Complete	ID_76	ID_50	ID_47	time	Amount Transferred	execute transaction	nonclone
10	Transition	Complete Payment	ID_48	ID_75	ID_47	time	successful transfer of amount	execute transaction	clone3
11	State	Idle	ID_68	ID_44	ID_44	time	time == 5	wait for card	nonclone
12	State	Idle State	ID_54	ID_44	ID_44	time	time == 5	wait for card	clone2
13	State	Insert card	ID_53	ID_44	ID_44	time	time == 5	check card	nonclone
14	State	Generate Card	ID_51	ID_44	ID_44	time	time == 5	Print ATM card	nonclone
15	State	Change Pin	ID_32	ID_44	ID_44	time	time == 5	set new PIN	nonclone
16	State	Enter New Pin Again	ID_14	ID_44	ID_44	time	time == 5	Ensure new PIN	clone3
17	State	Menu Display	ID_5	ID_44	ID_44	time	time == 5	display ATM services	nonclone
18	State	Menu	ID_65	ID_44	ID_44	time	time == 5	display options	clone3
19	State	select Bill	ID_56	ID_44	ID_44	time	time == 5	display options	nonclone
20	State	Withdraw Cash	ID_73	ID_44	ID_44	time	time == 5	withdraw cash	nonclone
21	State	Transfer Funds	ID_60	ID_44	ID_44	time	time == 5	Transfer amount	nonclone
22	State	Account Transfer	ID_55	ID_44	ID_44	time	time == 5	Enter account credentials	clone3
23	State	Generate ATM Card	ID_12	ID_44	ID_44	time	time == 5	create ATM card	nonclone

Figure 25. Clone Type 3 Results

CHAPTER 5

VALIDATION

The validity of the proposed approach is demonstrated using several case studies. A case study of an ATM is demonstrated below to validate the proposed approach. To exemplify the key outcomes and insights of the UMCD an ATM state machine model is used in the following case study.

5.1 Case Study

ATM remains idle initially, when the user inserts his card (insert card state) the system checks if the card is valid or not, if the card is valid system asks the user to enter PIN (Give Pin). If the pin is valid a menu “Select-Service” appears on the screen and the user selects the services represented as states in the state machine i.e., Cash-Withdraw, Balance-Inquiry, Transfer-Funds, Generate ATM card, Bill payment, change pin or quit. If the user selects Cash-Withdraw system asks the user to insert the amount, if the amount is valid cash is ejected (eject cash), and a receipt is printed (Print-receipt) and then the card is ejected (eject card). If the user selects ‘Balance-Inquiry’ the user's account balance is displayed as ‘View-Balance’ and a receipt is printed (Print-Receipt), if the user has no more functions card is ejected (eject card). If the user selects ‘Transfer-Funds’ the system asks the user to ‘Enter-Receiver-Account’ and then insert the amount. If the amount is valid cash is transferred (transfer cash), a receipt is printed (Print-receipt) and then the card is ejected (eject card) if the user selects ‘Generate ATM card’ the system asks the user to input user credentials and Account number. If provided data is valid card is generated otherwise the request is denied. If the user selects ‘change pin’ the system asks the user to enter ‘previous pin’ and ‘Enter New pin’ if both meet the system requirements new pin is generated otherwise the request is denied. If the user selects the ‘Bill Payment’ option, the system asks the user to select bill type, and mode of payment and enter the bill amount if the data is valid, and the user has sufficient account balance the transaction is completed otherwise the transaction is terminated.

5.1.1 Application of UML Model Clone Detection (UMCD) framework:

Following is the output of applying our proposed framework:

- Nine useful features are extracted from the XMI code of the ATM state machine. Such as element type, element name, element ID, source state, target State, constraint, constraint value, trigger, and state activity.

- In the preprocessing step, only relevant features are considered such as element type, name, constraints, and activity, and irrelevant columns are dropped.
- 184 elements are derived from the ATM state machine model.
- Out of those 184 elements, 28 element pairs are labeled as clone 1, 22 element pairs are labeled clone type 2, 25 element pairs are labeled as clone type 3, and the remaining 34 elements are labeled non-clone.

The performance of our proposed approach is calculated by using a confusion matrix, UMCD identified type-1 clones with 100% accuracy, clones of type 2 with 97.8% accuracy, and clones of type 3 with 92% accuracy for the case study of ATM with an overall accuracy of 96.6% as shown in the following table 2.

Table 5. Results of UMCD on ATM Case Study

Clone Type	Total Clones	Correctly identified	Falsely Identified	Accuracy (%)
Type 1	28	28	0	100
Type 2	22	20	2	97.8
Type 3	25	18	7	92
Total clones	75	66	9	96.6

The results of application of UMCD framework on ATM case study are also displayed in Figure 26.

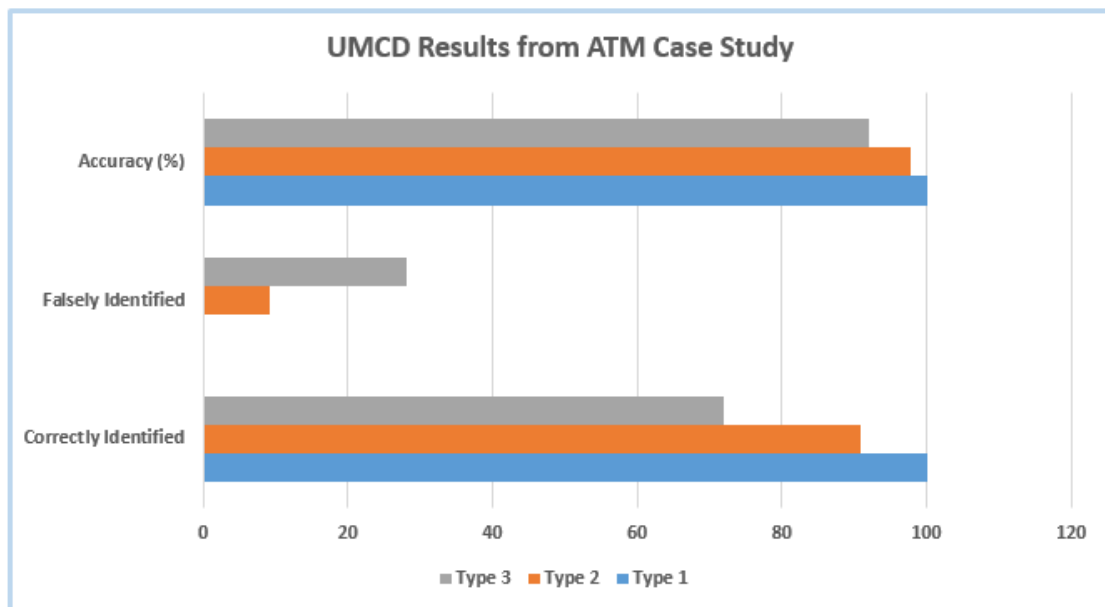


Figure 26. UMCD Results from ATM case study

The framework is also validated by applying to several other case studies. Figure 27 shows the results of UMCD on three case studies i.e., ATM machine, Telephone line system, and online shopping. The figure shows that our proposed framework yielded satisfactory results.

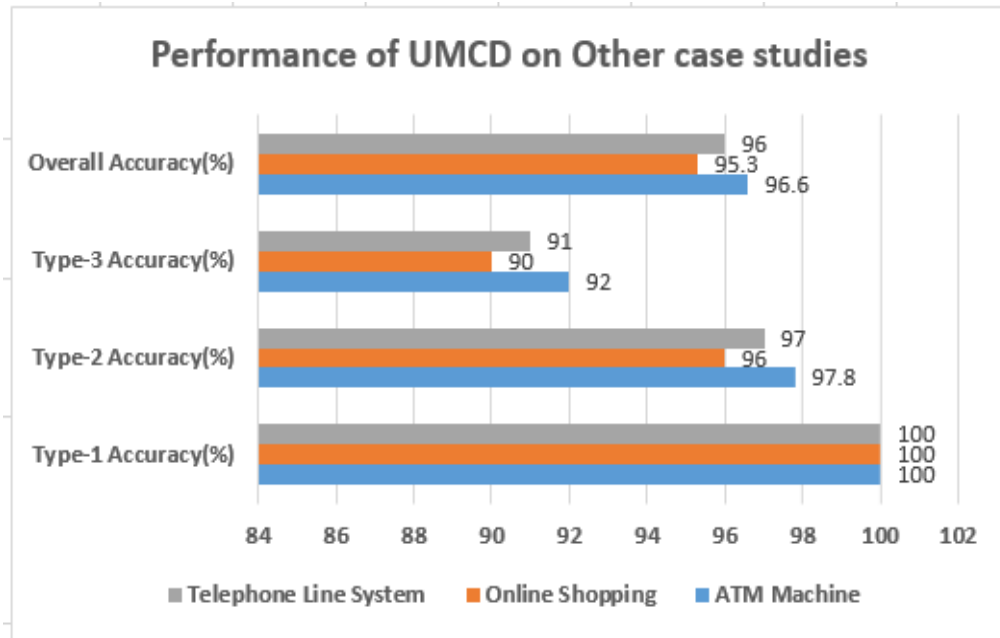


Figure 27. Performance results of UMCD on Other case studies

CHAPTER 6

DISCUSSION

Clone detection in software engineering is critical to assuring the quality and maintainability of software systems. Developers frequently reuse several code components in their products and fail to conduct code reviews to detect clones or refactor copied code. This produces code clones. These cloned components can lead to quality, consistency, maintainability, and bug propagation issues. UML diagrams such as class diagrams, activity diagrams, state machines, etc., are the core artifacts of the process of software development, used to specify and represent software design.

These models serve as a blueprint for navigating all aspects of the process of software development. Consequently, if there are clones in these UML models, they will also induce clones in later software development stages. Consequently, these clones will propagate and intensify the clone-related issues. In this research, a framework, UML Model Clone Detection (UMCD) is proposed for model clone detection. The proposed framework utilizes Natural language processing techniques to identify clones of types 1, 2, and 3 in UML state machine diagrams. No significant research has been conducted on identifying clones in UML state machine diagrams in previous research studies.

A state machine is used to describe a system's behavior in the form of states, triggers, transitions, etc. It provides a well-organized way to demonstrate and analyze the transitions between several states of a system in response to triggers, events, or inputs. Advanced NLP techniques make it easier to identify duplications in UML models since models contain a lot of textual information therefore making it easier to semantically analyze the text and apply textual similarity measures of natural language processing (NLP). Model clones are detected in UML State Machines and compared with assigned labels to identify if a clone pair is correctly identified by UMCD or not.

There are several reasons for the choice of using sentence transformers for our approach of clone detection in UML models, especially when compared to other NLP models like BERT or GPT which are more commonly used. First, Sentence transformers are optimized specifically for the assessment of semantic similarity, which is essential to the process of identification of clones in text or models. Whereas BERT or GPT, which are general-purpose language models are not specifically designed for semantic similarity but for a wide range of NLP tasks, sentence transformers have been fine-tuned on datasets focused on sentence or paragraph-level semantic similarity tasks. Thus, enhancing their ability to comprehend and compare meanings in text. Therefore, sentence transformers are more efficient for our framework for model clone detection where semantic equivalence is crucial. Furthermore, Sentence transformers also provide several efficient ways to compute embeddings for comparing similarity. They generate embeddings that are directly useful for cosine similarity calculations without the need for

further processing or pooling strategies that may be required by other complex models. Moreover, in systems where several UML models have to be analyzed quickly and effectively, this Direct Approach reduces the complexity of computation and speeds it up significantly.

Research shows that sentence transformers generally perform better on tasks involving the comparison of textual semantic content. In addition, The sentence transformers library offers an extensive array of pre-trained t and tools and models that facilitate easy integration and application.

Our framework successfully demonstrates the application of the sentence transformer models in identifying renamed clones and near-miss clones. Our UMCD framework was capable of detecting exact clones, renamed clones as well as near-miss clones in complex UML state-machine models.

6.1. Threats to validity

The proposed approach of clone detection in UML models has some threats to validity. Since the UML state machines are graphical by nature. As our technique makes use of XMI representation of the state machines, transforming these diagrams into textual descriptions for applying NLP techniques may lead to loss of information. Another threat to validity is that the results obtained from a specific set of UML state machines might not generalize to other domains or types of state machines. The proposed approach might not handle all variations in UML state machine designs.

CHAPTER 7

CONCLUSION AND FUTURE WORK

Clone detection in UML models is indeed an active research area within the broader fields of software engineering and model-driven development. As software systems become increasingly complex, maintaining the quality, consistency, and efficiency of UML models becomes crucial. Clone detection aims to identify duplicated or similar components within these models, enabling better software design and maintenance practices. Clone detection in UML models has several valuable applications across numerous phases of the software development lifecycle. It has significant a role in the design, development as well as maintenance phase of a software system in a cost and time-effective way. Identifying clones in UML models helps in maintaining and evolving software systems efficiently. By removing duplicated or redundant components, developers can focus on making changes to a single representation, reducing the risk of introducing errors. It also improves the quality of UML models by ensuring that design elements are unique, discrete, and concise.

UML models are used as a basis for generating code as they provide a roadmap to the implementation and development of software systems. Therefore, detecting and removing clones in UML models as per requirement can generate cleaner, more efficient, and maintainable code by eliminating redundant code patterns before they are translated into programming languages. Identifying clones in UML models can guide refactoring efforts by highlighting areas where design patterns can be applied or where common components can be consolidated for better maintainability.

Based on the significance of clone detection in UML models, researchers are continuously exploring and developing techniques, algorithms, and tools for detecting clones in UML models. Several researchers are investigating the application of graph-based methods, deep learning approaches, and machine learning algorithms to detect clones in UML diagrams. This approach uses the power of machine learning to automatically identify complex patterns and relationships in models. Researchers are working on techniques to improve the efficiency and scalability of clone detection algorithms, making them applicable to large-scale software systems. Some also aim to integrate clone detection tools directly into popular UML modeling and development environments, providing developers with real-time feedback and automated clone detection features. In conclusion, clone detection in UML models remains an active and evolving research area, driven by the need to improve software design practices, ensure maintainability, and enhance the overall quality of complex software systems.

Our proposed approach UML Model Clone Detection (UMCD) focuses on model clone detection in UML state machines. They are versatile modeling tools that have a wide range of uses across various domains, including software engineering, control systems, embedded systems, and more. In the domain

of software engineering, they are extensively used to model the behavior of complex systems. They help developers visualize how a system's behavior changes in response to different events, inputs, and conditions. Our approach uses advanced NLP concepts such as Sentence Transformers and TF-IDF. By utilizing Sentence Transformers, we effectively capture semantic similarities between sentences, while TF-IDF allows us to assess the relevance of terms within the documents. Together, these techniques enhance the accuracy and efficiency of our proposed framework.

Our future efforts will focus on improving the efficiency of the UMCD framework and enhancing the accuracy of the proposed framework by incorporating machine-learning approaches along with NLP. We also aim to identify clones in Object Constraint Language (OCL) properties associated with state-machine elements in the future. Further, we aim to enhance the system to incorporate the identification of Type 4 clones in UML state-machine models. Furthermore, in the future, we also aim to Identify if a clone is intentional and needed or if it is not needed and may cause bug propagation, quality, or maintenance issues later on. Therefore, to devise a technique to remove or refactor the unwanted clones as per requirement.

REFERENCES

- [1] Q. U. Ain, W. H. Butt, M. W. Anwar, F. Azam and B. Maqbool. 2019. A Systematic Review on Code Clone Detection," in IEEE Access, vol. 7, pp. 86121-86144, 2019, doi: 10.1109/ACCESS.2019.2918202.
- [2] Booch, G., Jacobson, I., & Rumbaugh, J. 1999. The unified modeling language user guide. Reading: Addison-Wesley.
- [3] Harald Störrle, 2010. Towards clone detection in UML domain models, ACM, 2010 ECSA 2010, August 23-26, 2010, Copenhagen, Denmark.
- [4] Franzago, M., Di Ruscio, D., Malavolta, I., & Muccini, H. 2017. Collaborative model-driven software engineering: a classification framework and a research map. IEEE Transactions on Software Engineering, 44(12), 1146-1175
- [5] Baker, Brenda S. 1995. On finding duplication and near-duplication in large software systems. In Reverse Engineering, 1995. Proceedings of 2nd Working Conference on, pp. 86-95. IEEE.
- [6] Chou, A. Yang, J. Chelf, B. Hallem, S. Engler, D.R. 2001. An empirical study of operating system errors. In Proceedings of the 18th ACM Symposium on Operating Systems Principles, Banff, AB, Canada, 21–24 October 2001; pp. 73–88.
- [7] Li, Z. Lu, S. Myagmar, S. Zhou, Y. 2006. CP-Miner: Finding copy-paste and related bugs in operating system code. IEEE Trans. Software. Eng. 2006, 32, 289–302.
- [8] Qurat Ul Ain, Farooque Azam, Muhammad Waseem Anwar, and Ayesha Kiran. 2019. A Model-driven Approach for Token-Based Code Clone Detection Techniques - An Introduction to UMLCCD. 8th ICEIT 2019. Association for Computing Machinery, New York, NY, USA, 312–317. <https://doi.org/10.1145/3318396.3318440>
- [9] Roy, Chanchal Kumar, and James R. Cordy. 2017. A survey on software clone detection research. Queen's School of Computing TR 541, no. 115 (2007): 64-68.
- [10] Vislavski, Tijana, Gordana Rakic, Nicolás Cardozo, and Zoran Budimac. 2018. LICCA: A tool for cross-language clone detection. In 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering s(SANER), pp. 512-516. IEEE, 2018.
- [11] Y Nakamura, Yuta, Eunjong Choi, Norihiro Yoshida, Shusuke Haruna, and Katsuro Inoue. 2016. Towards detection and analysis of interlanguage clones for multilingual web applications. In Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on, vol. 3, pp. 17-18. IEEE, 2016.
- [12] Jadon, Shruti. 2016. Code clone detection using machine learning technique: Support vector machine. In Computing, Communication, and Automation (ICCCA), 2016 International Conference on, pp. 399-303. IEEE, 2016.
- [13] Wang, Pengcheng, Jeffrey Svajlenko, Yanzhao Wu, Yun Xu, and Chanchal K. Roy. 2018. CCAAligner: a token-based large-gap clone detector. In Proceedings of the 40th International Conference on Software Engineering, pp. 1066-1077. ACM, 2018.

- [14] Yang, Yanming, Zhilei Ren, Xin Chen, and He Jiang. 2018. Structural Function Based Code Clone Detection Using a New Hybrid Technique. In 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), pp. 286- 291. IEEE, 2018.
- [15] Svajlenko, Jeffrey, and Chanchal K. Roy. 2017. Cloneworks: A fast and flexible large-scale near-miss clone detection tool. In Proceedings of the 39th International Conference on Software Engineering Companion, pp. 177-179. IEEE Press, 2017.
- [16] Sargsyan, Sevak, Sh Kurmangaleev, A. Belevantsev, and Arutyun Avetisyan. 2016. Scalable and accurate detection of code clones. In Programming and Computer Software 42, no. 1 (2016): 27-33.
- [17] Singh, Gurpreet. 2017. To enhance the code clone detection algorithm by using the hybrid approach for detection of code clones. In Intelligent Computing and Control Systems (ICICCS), 2017 International Conference on, pp. 192-198.
- [18] Nam H. Pham, Hoan Anh Nguyen, Tung Thanh Nguyen, Jafar M. Al-Kofahi, and Tien N. Nguyen. 2009. Complete and accurate clone detection in graph-based models. In Proceedings of 31st ICSE '09. IEEE Computer Society, USA, 276–286. <https://doi.org/10.1109/ICSE.2009.5070528>
- [19] M. H. Alalfi, J. R. Cordy, T. R. Dean, M. Stephan, and A. Stevenson. 2012. Models are code too: Near-miss clone detection for Simulink models, 28th IEEE International Conference on Software Maintenance (ICSM), Trento, Italy, 2012, pp. 295–304
- [20] Störrle, H. 2015. Effective and Efficient Model Clone Detection. In: De Nicola, R., Hennicker, R. (eds) Software, Services, and Systems. Lecture Notes in Computer Science, vol 8950. Springer, Cham. https://doi.org/10.1007/978-3-319-15545-6_25
- [21] Manar H. Alalfi, Elizabeth P. Antony, James R. Cordy. 2016. An Approach to Clone Detection in Sequence Diagrams and Its Application to Security Analysis, Software & Systems Modeling, 2016. <https://doi.org/10.1007/s10270-016-0557-6>
- [22] Mohammad Azangoo, Amirhosein Taherkordi, Jan Olaf Blech. 2020. Digital Twins for Manufacturing Using UML and Behavioral Specifications, 2020 25th IEEE conference.
- [23] H. Liu, Z. Niu Z. Ma W. Shao. 2011. Suffix tree-based approach to detecting duplications in sequence diagrams, IET Software., 2011, Vol. 5, Iss. 4, pp. 385–397
- [24] E. P. Antony, M. H. Alalfi and J. R. Cordy. 2013. An approach to clone detection in behavioral models, 2013 20th Working Conference on Reverse Engineering (WCRE), Koblenz, Germany, 2013, pp. 472-476
- [25] Sandeep Kumar Nain, Manila. 2017. Detecting Similarities and Clones Using UML Diagrams, 2017 IJEDR, Volume 5, Issue 1, ISSN: 2321-9939
- [26] D. Rattan, R. Bhatia and M. Singh. 2012. Model clone detection based on tree comparison, 2012 Annual IEEE India Conference (INDICON), Kochi, India, 2012, pp. 1041-1046
- [27] D. Vallejo-Huanga, J. Morocho and J. Salgado. 2023. SimilaCode: Programming Source Code Similarity Detection System Based on NLP, 2023 15th International Congress on Advanced Applied Informatics Winter (IIAI-AAI-Winter), Bali, Indonesia, 2023, pp. 171-178
- [28] S. Mhatre, S. Satre, M. Hajare, A. Hire, A. Itankar and S. Patil. 2023. Text Comparison Based on Semantic Similarity, 2023 3rd International Conference on Intelligent Technologies (CONIT), Hubli, India, 2023, pp. 1-5, doi: 10.1109/CONIT59222.2023.10205616.
- [29] V. V. Mayil and T. R. Jeyalakshmi. 2023. Pretrained Sentence Embedding and Semantic Sentence Similarity Language Model for Text Classification in NLP, 2023 3rd International Conference on

Artificial Intelligence and Signal Processing (AISP), VIJAYAWADA, India, 2023, pp. 1-5

- [30] Muslim Chochlov, Gul Aftab Ahmed, James Vincent Patten, Guoxian Lu, Wei Hou, David Gregg, Jim Buckley. 2023. Using A Nearest-Neighbor, BERT-Based Approach for Scalable Clone Detection, ARXIV-CS.SE, 2023.
- [31] S. Arshad, S. Abid and S. Shamail. 2022. CodeBERT for Code Clone Detection: A Replication Study, 2022 IEEE 16th International Workshop on Software Clones (IWSC), Limassol, Cyprus, 2022, pp. 39-45
- [32] U. Kelte, J. Wehren, and J. Niere. 2005. A generic difference algorithm for UML models” Proceedings of SE 2005, Essen, Germany, pp. 105-116.
- [33] <https://www.uml.org/what-is-uml.htm>
- [34] M. A. Mahima, N. C. Patel, S. Ravichandran, N. Aishwarya and S. Maradithaya. 2021. A Text-Based Hybrid Approach for Multiple Emotion Detection Using Contextual and Semantic Analysis," 2021 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES), Chennai, India, 2021
- [35] M. Alodadi and V. P. Janeja. 2015. Similarity in Patient Support Forums Using TF-IDF and Cosine Similarity Metrics, 2015 International Conference on Healthcare Informatics, Dallas, TX, USA, 2015, pp. 521-522
- [36] P. P. Gokul, B. K. Akhil, and K. K. M. Shiva. 2017. Sentence similarity detection in Malayalam language using cosine similarity, 2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), Bangalore, India, 2017
- [37] A. Desku, B. Raufi, A. Luma and B. Selimi. 2021. Cosine Similarity through Control Flow Graphs For Secure Software Engineering," 2021 International Conference on Engineering and Emerging Technologies (ICEET), Istanbul, Turkey, 2021
- [38] M. W. Anwar, M. Rashid, F. Azam, A. Naeem, M. Kashif and W. H. Butt, "A Unified Model-Based Framework for the Simplified Execution of Static and Dynamic Assertion-Based Verification," in IEEE Access, vol. 8, pp. 104407-104431, 2020
- [39] Satwinder Singh and Raminder Kaur. 2014. Clone detection in UML class models using class metrics. SIGSOFT Softw. Eng. Notes 39, 3 (May 2014), 1–3.
- [40] Martin Beckmann, Vanessa N. Michalke, Andreas Vogelsang, and Aaron Schlutter. 2017. Removal of redundant elements within UML activity diagrams. In Proceedings of the ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS '17). IEEE Press, 334–343. <https://doi.org/10.1109/MODELS.2017.7>
- [41] H. Jnanamurthy, F. Henskens, D. Paul and M. Wallis, "Clone Detection in Model-Based Development Using Formal Methods to Enhance Performance in Software Development," 2018 3rd International Conference for Convergence in Technology (I2CT), Pune, India, 2018, pp. 1-8, doi: 10.1109/I2CT.2018.8529446.
- [42] Strüber, D., Acrețoai, V. & Plöger, J. Model clone detection for rule-based model transformation languages. *Softw Syst Model* 18, 995–1016 (2019). <https://doi.org/10.1007/s10270-017-0625-6>
- [43] Al-Batran, B., Schätz, B., Hummel, B. (2011). Semantic Clone Detection for Model-Based Development of Embedded Systems. In: Whittle, J., Clark, T., Kühne, T. (eds) *Model Driven Engineering Languages and Systems. MODELS 2011. Lecture Notes in Computer Science*, vol 6981. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-24485-8_19