

A Model Driven Framework for the Development of Multi-Platform Mobile User Interfaces



Author

Mehreen Khan

FALL 2016-MS-16 (CSE) 00000118628

MS-16 (CSE)

Supervisor

Dr. Farooque Azam

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY
ISLAMABAD
JANUARY, 2019

A Model Driven Framework for the Development of Multi-Platform Mobile User Interfaces

Author

Mehreen Khan

FALL 2016-MS-16(CSE) 00000118628

A thesis submitted in partial fulfillment of the requirements for the degree of
MS Software Engineering

Thesis Supervisor:

Dr. Farooque Azam

Thesis Supervisor's Signature:

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,
ISLAMABAD
JANUARY, 2019

DECLARATION

I certify that this research work titled "*A Model Driven Framework for the Development of Multi-Platform Mobile User Interfaces*" is my own work under the supervision of Dr. Farooque Azam. This work has not been presented elsewhere for assessment. The material that has been used from other sources has been properly acknowledged / referred.

Signature of Student

Mehreen Khan

FALL 2016-MS-16(CSE) 00000118628

LANGUAGE CORRECTNESS CERTIFICATE

This thesis is free of typing, syntax, semantic, grammatical and spelling mistakes. Thesis is also according to the format given by the University for MS thesis work.

Signature of Student

Mehreen Khan

FALL 2016-MS-16(CSE) 00000118628

Signature of Supervisor

COPYRIGHT STATEMENT

- Copyright in text of this thesis rests with the student author. Copies (by any process) either in full, or of extracts, may be made only in accordance with instructions given by the author and lodged in the Library of NUST College of E&ME. Details may be obtained by the Librarian. This page must form part of any such copies made. Further copies (by any process) may not be made without the permission (in writing) of the author.
- The ownership of any intellectual property rights which may be described in this thesis is vested in NUST College of E&ME, subject to any prior agreement to the contrary, and may not be made available for use by third parties without the written permission of the College of E&ME, which will prescribe the terms and conditions of any such agreement.
- Further information on the conditions under which disclosures and exploitation may take place is available from the Library of NUST College of E&ME, Rawalpindi.

ACKNOWLEDGEMENTS

I am thankful to my Creator Allah Subhana-Watala to have guided me throughout this work at every step and for every new thought which You setup in my mind to improve it. Indeed, I could have done nothing without Your priceless help and guidance. Whosoever helped me throughout the course of my thesis, whether my parents or any other individual was Your will, so indeed none be worthy of praise but You.

I would also like to express my gratitude to my supervisor **Dr. Farooque Azam** and my co-supervisor **Dr. Wasi Haider Butt** for their constant motivation and help throughout this thesis. Also, for Software Development and Architecture (SDA), Model-driven Software Engineering (MDSE) and Software Requirement Engineering (SRE) courses which they have taught me. I can safely say that I haven't learned any other engineering subject in such depth.

I would also like to thank my Guidance Committee Members **Dr. Arslan Shaukat** and **Dr. Urooj Fatima** for being on my thesis guidance and evaluation committee. Their recommendations are very valued for improvement of the work. I would like to pay special thanks to **Muhammad Waseem Anwar** for his incredible cooperation. I appreciate his guidance throughout the whole thesis. I am also grateful to **Fatima Samea** for her assistance and support.

Finally, I would like to express my gratitude to all the individuals who have rendered valuable assistance to my study.

Dedicated to my remarkable parents and siblings, their incredible support and cooperation led me to this great achievement

ABSTRACT

Since the emergence of mobile devices, the architecture of mobile applications have been transformed significantly and its complexity has grown enormously. In such mobile applications, the User Interface (UI) is an important ingredient and with the increased complexity, its development is getting cost / time-consuming process. There exist several mobile platforms (e.g. Android, IOS etc.) where each has its own operating requirements. Even single platform contains several versions of operating systems with different operating requirements. In this situation, the development of multifaceted mobile user interfaces for different platforms is very challenging especially dealing with the productivity and time-to-market constraints. Consequently, there is a strong need to develop a simple and automated framework for the development of multi-platform mobile user interfaces by exploiting the concepts of Model Driven Architecture (MDA). Therefore, in this research, a model-based framework is proposed for the development of multi-platform mobile user interfaces. Particularly, the UMUI (Unified modeling language profile for multi-platform **M**obile **U**ser **I**nterfaces) has been developed, which adapts the concepts of UML Class Diagram for representing the mobile user interface requirements at higher abstraction level. As a part of research, a complete MMUI (**M**odel driven multiplatform **M**obile **U**ser **I**nterfaces) transformation engine has been developed, utilizing Model-to-Text (M2T) approach, in order to automatically transform the high level UMUI source models into low level react native implementation. Finally, the proposed framework application is validated through two bench mark case studies i.e. Patient Management System & Library Application. The results prove that the proposed framework allows the modeling of multiplatform user interfaces with styling, layout, native look and feel requirements through UMUI with simplicity. Subsequently, the deployable react native code can automatically be generated from the source models through MMUI transformation engine. It has been concluded that the proposed framework is highly beneficial to achieve major business objectives like productivity and time-to-market for mobile applications.

Keywords: Model driven engineering (MDE), Mobile based application development, Model based user interface development (MUID), User Interface (UI), Unified Modeling Profile (UML)

TABLE OF CONTENTS

DECLARATION	i
LANGUAGE CORRECTNESS CERTIFICATE.....	ii
COPYRIGHT STATEMENT.....	iii
ACKNOWLEDGEMENTS	iv
ABSTRACT	vi
TABLE OF CONTENTS	vi
LIST OF FIGURES.....	ix
LIST OF TABLES.....	x
CHAPTER 1: INTRODUCTION.....	12
1.1. Background Study.....	12
1.1.1. Model-Driven Software Engineering	12
1.1.2. Model-Driven Mobile Application Development	14
1.1.3. Model-Based User Interface Design	14
1.2. Problem Statement.....	14
1.3. Proposed Methodology	14
1.4. Research Contribution.....	15
1.5. Thesis Organization	15
CHAPTER 2: LITERATURE REVIEW	18
2.1. Literature Review.....	18
2.2. Research Gaps	21
CHAPTER 3: PROPOSED METHODOLOGY	24
3.1. Proposed Profile	24
3.2. Input Elements Profile.....	27
3.3. Static Elements Profile.....	29
3.4. Dynamic Elements Profile	32
CHAPTER 4: IMPLEMENTATION	38
4.1. Transformation Rules.....	38
4.2. Transformation Engine Architecture.....	42
CHAPTER 5: VALIDATION	46
5.1. Patient Management System Case Study	46
5.1.1. Requirement Specification	46
5.1.2. Modeling.....	47
5.1.3. Code Generation	51
5.1.4. Verification.....	52
5.2. Library Application Case Study	56
5.2.1. Requirement Specification	56
5.2.2. Modeling.....	57
5.2.3. Code Generation	60

5.2.4. Verification.....	61
CHAPTER 6: Comparative Analysis.....	65
6.1. Comparison with Previous Studies	65
CHAPTER 7: DISCUSSION AND LIMITATION	69
7.1. Discussion.....	69
7.2. Limitations	70
CHAPTER 8: CONCLUSION AND FUTURE WORK	72
APPENDIX A.....	73
REFERENCES	77

LIST OF FIGURES

Figure 1: Research Flow	14
Figure 2: Thesis Outline	16
Figure 3: Proposed Unified Modeling Language profile for Mobile User Interfaces (UMUI)	25
Figure 4: Input Components Profile	28
Figure 5: Static Elements Profile	30
Figure 8: Transformation Engine Architecture	43
Figure 9: Transformation Tool	44
Figure 10: Object diagram of Patient Management System (Home Screen)	48
Figure 11: Object diagram of Patient Management System (Login Screen)	48
Figure 12: Object diagram of Patient Management System (Status Screen)	49
Figure 13: Object diagram of Patient Management System (Patient Information Screen)	49
Figure 14: Object diagram of Patient Management System (Main Screen)	50
Figure 15: Assigning Values to Instance Specifications	50
Figure 16: Transformation for User Interfaces of Patient Management System Application	51
Figure 17: Generated Files of Patient Management System	52
Figure 18: Generated Code of User Interfaces of Patient Management System	52
Figure 19: Expo XDE Desktop Application	53
Figure 20: Expo Mobile Client	53
Figure 21: User Interface after Model Transformation (Login Screen)	54
Figure 22: User Interface after Model Transformation (Profile Screen)	54
Figure 23: User Interface after Model Transformation (Registration Screen)	55
Figure 24: User Interface after Model Transformation (Patient Information Screen)	55
Figure 25: User Interface after Model Transformation (Status Screen)	56
Figure 26: Object Diagram of Library Application Case Study (App Screen)	58
Figure 27: Object Diagram of Library Application Case Study (Home Screen)	58
Figure 28: Object Diagram of Library Application Case Study (Profile Screen)	59
Figure 29: Object Diagram of Library Application Case Study (Login Screen)	59
Figure 30: Transformation for User Interfaces of Library Application Case Study	60
Figure 31: Generated Files of Library Application Case Study	60
Figure 32: Generated Code of User Interfaces of Library Application Case Study	61
Figure 33: Expo XDE Desktop Application	61
Figure 34: Expo Mobile Client	62
Figure 35: User Interface after Model Transformation (Login Screen)	62
Figure 36: User Interface after Model Transformation (Profile Screen)	63
Figure 37: User Interface after Model Transformation (Profile Screen)	63

LIST OF TABLES

Table 1: Transformation rules39
Table 2: Comparison with previous studies65

Chapter 1

Introduction

CHAPTER 1: INTRODUCTION

This chapter offers a detailed introduction of the research. **Section 1.1** discusses the background study, **Section 1.1** presents the problem statement, **Section 1.2** gives proposed methodology in, research contribution is detailed in **Section 1.3** and **Section 1.5** contains thesis organization.

1.1. Background Study

The intention of providing the background study is to introduce the main concepts used in this research. The concepts involved are; 1) Model-Driven Software Engineering 2) Model-Driven Mobile Application Development and 3) Model-Based User Interface Design (MBUID). The details of the following are given in subsequent sections.

1.1.1. Model-Driven Software Engineering

Model-Driven Software Engineering is a field of software engineering which deals with the software's complexity by developing the conceptual meta-models and models of a particular problem. Thus, providing an abstract representation of processes and knowledge which are essential for the execution and creation of a software. It helps in analyzing and understanding problem of a particular domain and foster the communication between developers, architectures and managers.

Model-Based Software Engineering uses different multiple modeling languages to create the meta-models as well the conceptual model of a particular problem of a specific domain. Many software engineers use UML (Unified Modeling Language), which is one the most effective language in this domain. It provides a standard way to present a design of an expected system. Structural and Behavioral diagrams are used by UML to represent the system. Structural diagram deals with the overall structure of a system that is component and their dependencies. Whereas, Behavioral diagram deals with the behavior of the system i.e. step by step activities of a component. Through UML profile diagram, UML conceptual models can be customized and extended. It also helps in preventing the contradiction and refining the standard semantics in a strict manner. Using transformation engine, the models created by above diagrams can be converted into code of any

Language. Transformation engine have ability to synthesize other artifacts too e.g. documentation, model, configuration etc. and is based on transformation rules.

1.1.2. Model-Driven Mobile Application Development

Now a days, due to intensive use of smart devices, mobile applications are becoming more existent in our lives and mobile domain is experiencing intense competition due to increasingly powerful devices to market and innovative operating systems continuously introducing. To face the fast-emerging technology for mobile application development for multiple-platforms user interfaces, model-driven development is a promising approach. Due to increasing attention several researchers proposed a model-based approach for different features of mobile application such as user interaction, user interface, application data and business logic. In model driven paradigm, models are the primary artifacts and through model's implementation is generated automatically. Which allows improvement in quality and productivity of the software. This study [13] presents the state-of-art model-based approaches and classified them on basis of model-driven techniques, mobile application types, supported mobile platforms.

1.1.3. Model-Based User Interface Design

MBUID is a method based on models to structure the development of user interfaces. This approach allows the developers attention on creating the abstract representation of user interfaces and delegate the user interface code generation process to automatic tools. By identifying the high-level models, it makes the development process more structured allowing a designer to analyze and specify applications from an abstract level without being tangled by many implementation details. To specifically state each chunk of development process is the one important aspect of the model-based user interface design in an independent way, thus forming a sheer separation of concerns in the applications definition.

1.2. Problem Statement

Rapid evolution of the mobile market results in significantly transformation of mobile architecture due to which mobile applications complexity has grown enormously. In such mobile applications, the User Interface (UI) is an important ingredient and with the increased complexity, platform fragmentation, stakeholder diversity, its development is getting cost / time-consuming process. Since each platform has its own interfaces and

architecture, developing user interfaces for a multi-platform is a daunting and tedious task for any application developer. Developing multi-platform mobile user interfaces has become need of the hour because poor user interface design may result in rejection of an application. Thus, main problem addressed in our proposed work are platform fragmentation, stakeholder diversity and development complexity.

1.2. Proposed Methodology

The entire research is done in a very systematic way. Step by step, flow of the research is shown in **Figure 1**. First, we identify the problem, then we propose a solution to the identified problem. Then, we carry out a comprehensive literature review which becomes the foundation of the proposed solution. Researches related to the proposed solution are analyzed and compared.

The proposed work includes an automated approach for obtaining the graphical user interface. The proposed tool provides facilities of modeling, transformation and verification. Mapping rules defined for the transformation become the basis of transformation engine. In the implementation phase, the transformation engine helps transform the high-level UMUI source models into low-level implementation for mobile user interfaces. The proposed methodology has been validated through patient management application and library application case study.

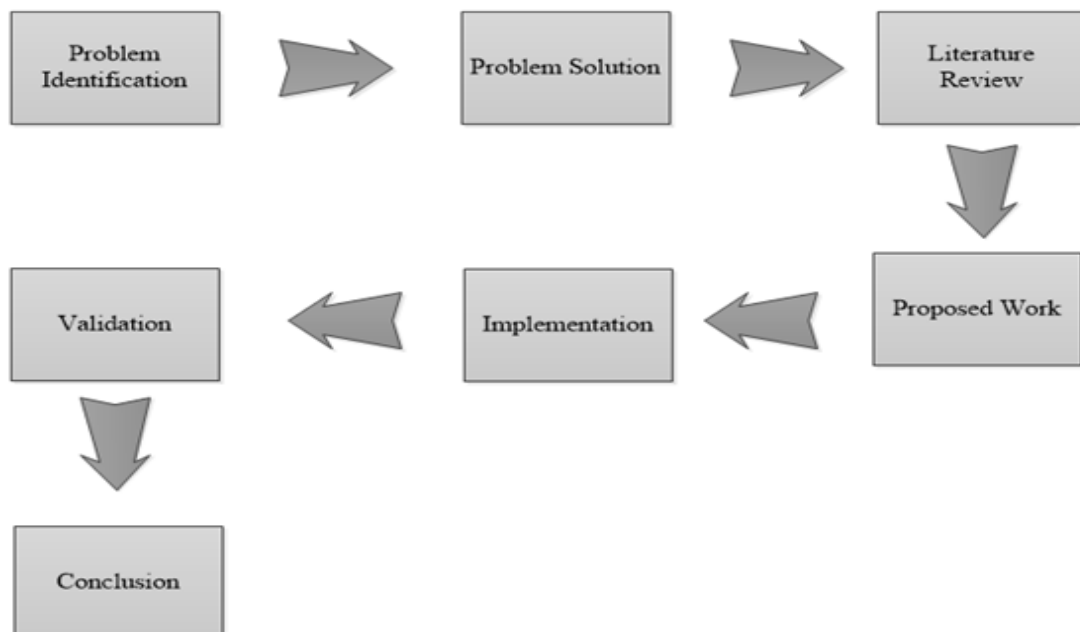


Figure 1: Research Flow

1.3. Research Contribution

Contributions made from this research work are

- We have presented model driven framework to design user interfaces for multi-platform mobile interfaces. It helps to provide a higher-level abstraction of the system and reduces the design complexity.
- The transformation engine is developed using Java and Acceleo for the generation of automated multi-platform mobile user interfaces from the UMUI models.
- We have provided validation of our proposed work using benchmark case studies i.e. patient management system case study and library application case study.

1.4. Thesis Organization

Organization of the thesis is represented in **Figure 2**. **CHAPTER 1: INTRODUCTION** offers a brief introduction containing the background study, problem statement, research contribution and thesis organization. **CHAPTER 2: LITERATURE REVIEW** provides the detailed literature review highlighting the work done in the area of mobile application development using Model Driven Architecture (MDA). The systematic literature review is composed of four main sections. First section is review protocol which gives details on the methodology using which the literature review is carried out. Section two offers details on research works using MDA for mobile user interface development (MBUID). Whereas, section three highlights the research gaps that we encountered. **CHAPTER 3: PROPOSED METHODOLOGY** covers the details of proposed methodology used for identification of problem. **CHAPTER 4: IMPLEMENTATION** presents the detailed implementation regarding the proposed tool and transformation engine along-with its architecture. **CHAPTER 5: VALIDATION** provides the validation performed for our proposed methodology using two case studies. The two case studies selected for validation purpose is Patient Management System and Library Application. **CHAPTER 7: DISCUSSION AND LIMITATION** contains a brief discussion on the work done and contains the limitations to our research. **CHAPTER 8: CONCLUSION AND FUTURE WORK** concludes the research and a future work for the research.

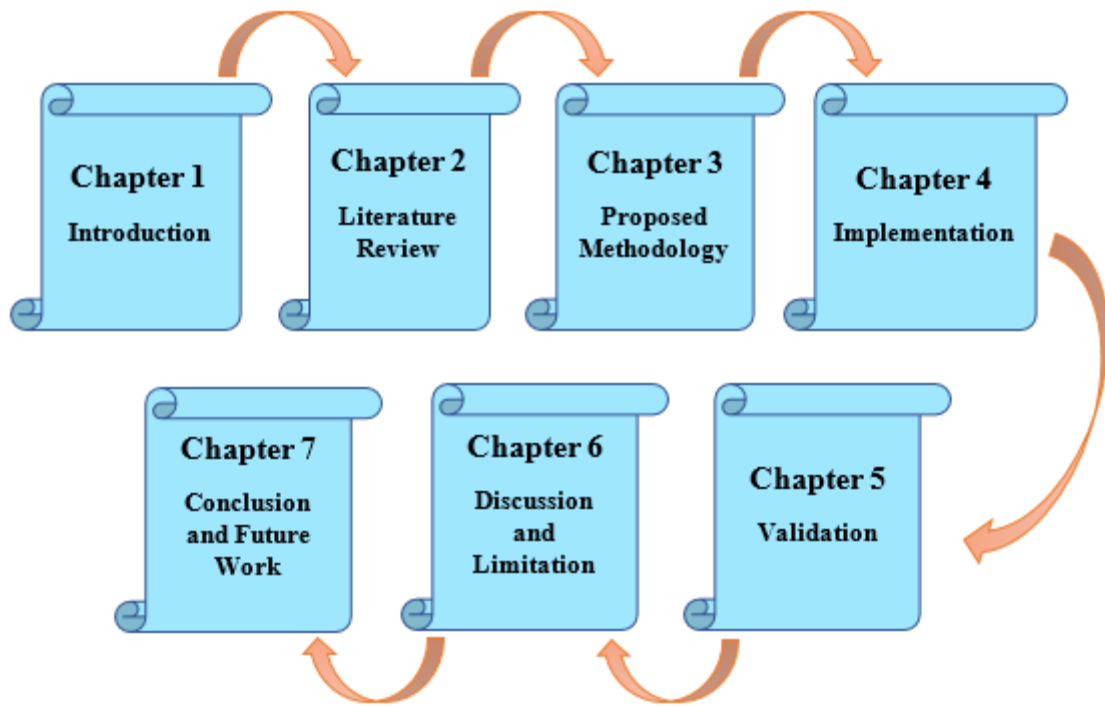


Figure 2: Thesis Outline

Chapter 2

Literature Review

CHAPTER 2: LITERATURE REVIEW

This chapter presents research work conducted in the area of Model Driven Mobile Application Development. After a brief literature review of work conducted in this area we enlightened the research gaps that we found in previous works.

2.1. Literature Review

In the software engineering field, recent researches show the emerging paradigm from object oriented to model oriented, where everything is an object to everything is a model, demands to change saying “write once, run everywhere” by “model once, generate everywhere” [4,10]. In the recent years, when it comes to MDD (Model Driven Development) of mobile applications, several approaches have been proposed and one is AXIOM [8], a MDE approach for the development of multiplatform application. For model transformation and code generation, it uses Abstract Model Tree and application requirements are described in platform independent model (PIM) using AXIOMs DSL later transformed to a running code. Another Solution Mendix App Platform which is commercial with a MDD platform using Cordova to generate cross platform apps. 82% of the surveyed approaches try to bring the focus in the native application development. The core reason of this is apparent high application quality, due to ability of exploiting, performance, usability and integrating with the most advance features of smart phones. Small and medium sized enterprises (SEMs) with limited resources are losing advantages of the native applications to reach a substantial audience at a maintainable resources such as cost. According to Gartner more than 50% applications deployed in 2016 were hybrid. [5]

Another approach extends the WebML standard for modelling RIA and it focuses in functionality distribution, content model and business logic. Regarding the user interface, only the page structure can be modelled, but not single user interface components. A WebML based tool exists which is also a commercial product, this also focuses on business logic, content models and lacks the capabilities of modelling user interface components. [6] Brambilla describes an extension to IFML for the model based development of application optimized for Apache Cordova Framework based on JavaScript (JS), HTML5 and CSS. Inspired by the experience of WebRatio Tool and WebML, WebRatio developed the IFML and OMG adopted it as a standard. It can be used to model the content, structure and user interaction of web pages, however as stated on their website, “IFML (Interaction Flow Modeling Language) does not

cover the modeling of presentation issues of an application frontend e.g. Layout, Style and look & Feel. Also, does not cater for the specification of native mobile applications. [7]

Christoph proposed MAML framework [1] to model mobile applications using graphical DSL. While the evaluation of this framework supports the advantages of it's related with technical IFML notation, but to confirm its results, applying it to real world case studies and more extensive studies, might reveal further need for improvement. Design of front end mobile user interfaces is a complex task. IFML is an OMG's standard introduced in 2013 and can be used in user interface modeling of desktop, web and mobile applications. Maryam [2] performed a detailed systematic literature review on IFML and concluded that it's a challenging area and it's not mature enough. Mobile extension of IFML is presented in [3] for the control behavior, content and user interaction of front end applications.

Delgado [9] discussed and analyzed the limitations and benefits of reusing User Interface elements supported by Model Based UI development (MBUID) and proposed an approach called WAINE, which supports in its models, the reuse of user interface elements and results shows that with flexible techniques provides a significant benefit in cost consuming task and provides direction with enhanced capabilities for future systems and also highlighted the open issues for future research direction, included: Exploring and analyzing reusability techniques e.g. object oriented, component based etc. by increasing the body of knowledge in the Model Driven User Interface Development (MBUID) context.

Hanane [10] proposed a Model Driven approach by using acceleo based on automatic code generation for android application development. Roberto [11] proposed a tool called Web Ratio Mobile Platform, which is an extended version of Object Management Group standard language called IFML. This tool features full code generation in Apache Cordova Framework. Model driven approach for android application development can be found in Abilio [13] encamping an increased modeling effort because the input model to this tool can be expressed through UML diagrams such as Class Diagram and Sequence Diagrams. The input itself is not a plain PIM, it requires advance Android knowledge is one of the drawbacks.

Muhammad [14] proposed an approach that allows the user to automate the business logic code for mobile platforms by a tool called Mobile Application Generator (MAG) this approach uses UML profile, class diagram, state machine diagram and ALF. Another model driven approach called Mobile Multimodality Creator (MIMIC) [17] that uses M4L that allows the modeling

and code generation of multimodal mobile interfaces for Web, Android and iOS. Obeo Designer is used to specify its graphical editor.

Pierre [18] carried out a brief existing literature review for developing Adaptive Model-Driven User Interfaces, evaluation of these systems, their usability problems. And also, highlighted the issues which can be solved through further research. Such as OpenXava and Hitalia provides diverse model driven approaches for developing user interfaces, but their coupling with programming languages hinders their use as a general solution. Andreas [20] published about Model-Driven Framework for user interface generation called RUMO conforming to MOF standard. DSL is used to define the set of rules to generate the target user interfaces. It uses the template mechanism which addresses the issues of various versions of similar platforms. Mobl is a new language designed by Zef Hemel and Eelco Visser [22], based on DSL to construct mobile web applications for UI design, data modeling and querying.

Eric et al [23] conducted a survey study to find the best trade-offs between various different code generation approaches based on model driven architecture and results indicates that in absolute terms, there is no approach better than others and provides some useful guidelines to identify the best techniques. Recently, the hot discussion in the R&D (research and development) community is to adapt whether to choose between native or cross platform and the result of comparative study included the both for code generators for native application development (iOS and Android) and for cross platform application such as Titanium and PhoneGap.

MobML (24) is a framework that assists the design and development of mobile applications. This framework comprises of four different languages. Each of which focusing a different various concerns such as navigation, UI, Content and Business Logic. This tool receives four different input models and produced the source code for targeted platform. Henning et al [25] proposed a MD2 for the development of mobile applications specifically designed for data driven business applications through a textual DSL, defined in Xtext. Some of the limitations of this approach are its scope is limited with mainly focused on data-driven applications with user interface mainly consisting of form fields and lacks to supports other main device features currently provides access only to GPS.

Wafa et al [12] addresses how the development of native and cross platform mobile applications is a challenging task and categorize into six main approaches, explains their limitations and open research areas. Following are the approaches (1) Compilation (2)

Interpretation (3) Cloud-Based (4) Modeling (5) Component-Based (6) Merged. New platforms support the component-based architecture and focuses on the common functions. By generating the UI code, Modeling saves the time as well useful in prototyping and the language used is an effective tool to define requirements. Limitation mentioned is Model Driven Development (MDD) does not support the native application development. Interpretation approach is sub-divided into Web-based, Virtual Machine and Runtime. Cons of the web-based applications are user interface have less performance and doesn't have a native look and feel. For example, PhoneGap and xFace uses web-based technologies. While using virtual machine results in slow execution time and at run time loading performance is lower. Titanium belongs to this category. There are three different types of mobile applications which are web app, native app and hybrid application. The latter two are more widely used. Titanium and Xamarin are used to produce native apps whereas PhoneGap is used to produce Hybrid apps. Final solution does not exist and still many approaches are under research. The main part in the future is probably be played by Merged approach. By using this approach, we can merge different multiple approaches together to minimize the drawback and benefit from the advantages of each individual approach. For example, ICPMD [26] merges the cross-compilation approach, component-based approach and a new language.

2.2. Research Gaps

This section discusses the research gaps and the proposed solution on multi-platform user interface code generation. On whole, we have identified some research works related to model-based UI code generation. In terms of user interface generation many of the above approaches uses libraries and templates through a complex development process and if any tool and approach was presented, it was mostly web based. They were mostly limited to generating business logic only. For user interface reliance on existing IFML (Interaction Flow Modeling Language) none of them covers the model driven development of multi-platform user interface native components with style, layout and look & feel. And most approaches comprise of DSL (Domain Specific Language), limitation to this approach is that it is complex and to ensure a good startup of the project developer need to study and learn the new language in order to use a solution.

Our proposed solution uses React Native as it is popular due to quality of documentation, development cost, emulators and debugging, response time and speed, commercial appreciation, code reuse and teamwork, maintenance and improvements i.e. upgrades. In our

proposed solution we have adopted the Model driven approach to provide the higher level of abstraction with automated user interface development to deal with complexity.

Chapter 3

Proposed Methodology

CHAPTER 3: PROPOSED METHODOLOGY

This chapter contains details of the proposed methodology. **Section 3.1** discusses the **UMUI** (Unified Modeling Language profile for multiplatform **M**obile **U**ser **I**nterfaces) and **Section 3.2** provides details of proposed solution.

3.1. Proposed Profile

MDA (Model Driven Architecture) is a renowned application development approach that simplifies the design and development of a system. UML profile is a generic extension mechanism of model driven and comprises of following major elements (1) Meta-Class (2) Stereotypes (3) Tagged-Values (4) Enumerations. Stereotype defines how an existing meta-class can be extended, when stereotypes are applied to an element the value of the tagged definitions are referred to as tagged values. Enumerations represent user defined datatypes. And contain a set of named identifiers called enumeration literals. UML consists of two types of modeling. (1) Dynamic Modeling (2) Static Modeling. Dynamic modeling can be described using sequence diagrams and state machine diagrams for behavioral aspects of a system whereas the static modeling can be described using class diagram and covers the structural aspects of a system through instance specification, stereotypes and tagged values.

UML profile is developed to represent mobile interfaces requirements. Stereotypes are extended from meta-class of classifiers, models and some are extended from meta-class of instance specifications. These UML stereotyped elements contain tagged values and parameters which help in derivation of code and instances of the classes. The details of stereotype and their purpose is discussed in sub sections. In our proposed framework we are using UML profiling method for specializing the general constructs of existing modeling language and refining it. The proposed UML profile is an extension to the meta-level UML concepts used to provide the modeling to automatically generate mobile user interface implementations in from the models. The proposed profile is shown in **Figure 3**. The reason for this choice is that this approach greatly facilitates measure the transformations made between the different models and text.

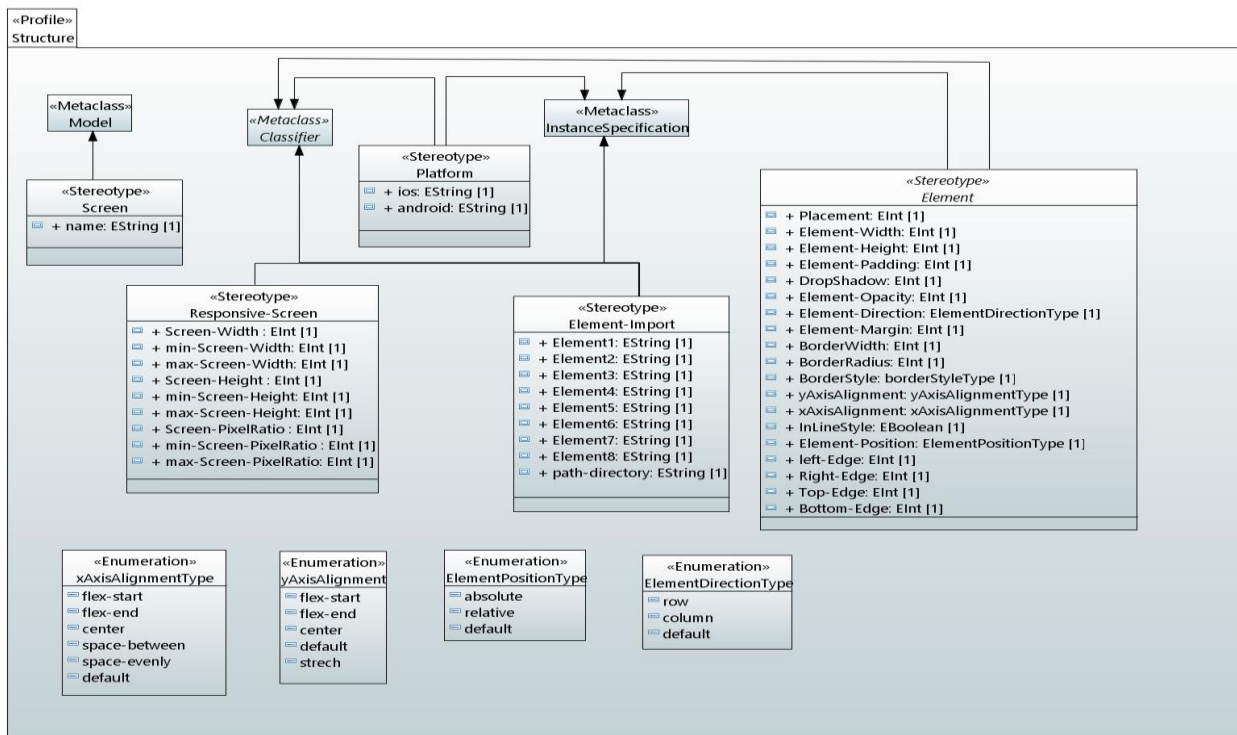


Figure 3: Proposed Unified Modeling Language profile for **Mobile User Interfaces (UMUI)**

- **Screen Stereotype**

Description: An application is composed of several screens with which the user can interact.

Base Class: Model

Tagged Values: It has one tagged value of type string. *name: EString [1]*

- **Platform Stereotype**

Description: Platform stereotype is for implementing separate visual components for different platforms if a scenario may arise for a for a code to be different. Certain elements may have properties that may work in one platform only.

Base Class: Instance Specification, Classifier

Tagged Values: *android* is of type string, for android operating system. *iOS* is for Mac operating system, also of type string.

- **Element Stereotype**

Description: Elements let you split the UI into independent, reusable pieces. by default, all the Elements have a series of properties relative to the size and the position, which they inherit from the superclass called Element. We have categorized the Elements into three sub categories
(1) Dynamic Elements (2) Static Elements (3) Input Elements
To achieve the right layout, algorithm is used to specify the consistent layout on different screen sizes, we need to use a combination of `xAxisAlignment`, `yAxisAlignment` and `Element-Direction`.

Base Class: Instance Specification, Classifier

Tagged Values: *Placement* is of type integer and used for dynamically expand and shrink based on available space. Setting a value of one results to fill all the available space. The higher the value given, the higher the ratio of space taken by the component by its siblings. *Element-Height* and *Element-Width* are of type Integer, elements height and width will determine its size on screen. *Element-Padding* is of type integer and is used for setting padding area on all the four sides of an element. *Drop-Shadow* is of type integer and is used for adding a drop shadow to the item. *Element-Margin* is of type integer and is used for setting margin area on all the four sides of an element. *Border-Width* and *BorderRadius* are of type integer and is used for setting the `borderWidth` and `borderRadius` of an element. *Left-Edge*, *Right-Edge*, *Top-Edge*, *Bottom-Edge* these are the number of logical pixels of type integer to offset the corresponding edge of an element. *zIndex* of type integer and is used to control which element is displayed on top of others. *Position* is an enumeration having two enumeration-literal `absolute` and `relative`. By default, it is `relative`. *xAxisAlignment* is an enumeration and it aligns the component in main direction, controls how are they align vertically having enumeration literals `center`, `flex-end`, `flex-start`, `stretch`. *yAxisAlignment* aligns the element in cross direction and controls how are they align horizontally having enumeration literals `center`, `flex-start`, `flex-end`, `space-between`, `space-evenly`, `space-around`. *Element-Direction* controls which direction

of the container children go column or row. *InStyle* of type Boolean controls whether you want to apply the inline style or not.

- **Responsive-Screen Stereotype**

Description: Responsive-Screen allows to manage the responsive layouts between screen sizes and displays.

Base Class: Instance Specification, Classifier

Tagged Values: *Screen-Width* is of type integer, describes the rendering surface width of the output device. *Min-Screen-Width* is of type integer, describes the min width of the rendering surface of the output device. *Max-Screen-Width* describes the max width of the rendering surface of the output device. *Screen-Height* describes the height of the rendering surface of the output device. *Min-Screen-Height* describes the min height of the rendering surface of the output device. *Max-Screen-Height* describes the max height of the rendering surface of the output device. *Screen-PixelRatio* describes the pixel ratio of the rendering surface of the output device. *Min-Screen-PixelRatio* describes the min pixel ratio of the rendering surface of the output device. *Max-Screen-PixelRatio* describes the max pixel ratio of the rendering surface of the output device. All the above tagged values are of type integer.

3.2. Input Elements Profile

Input Elements: These Elements are oriented to obtain input by the user. **Figure 4** shows the input elements profile. The stereotypes included in this profile are checkbox element, Picker Element, Toast element, Searchbar element, InputGroup element, and radio. The description of each stereotype with tagged values and base classes are described below.

- **Picker Stereotype**

Description: Picker stereotype allows user to point and click to get hovered element

Base Class: Instance Specification, Classifier

Tagged Values: *Enabled* tagged value is of type boolean and it is used to specify for enabling or disabling a picker button. *Placeholder* tagged value is of data type string. *supportedOrientations* tagged value allows the modal to

rotate to any of the specified orientations by type enumeration *supportedOrientationsType* having two enumeration literals landscape and portrait.

- **Radio Button Stereotype**

Description: Radio button stereotype allows the user to select any option from given set of options.

Base Class: Instance Specification

Tagged Values: *Selected* represents the state value of an item from set of choices. *color* of type string represents user define color. *selectedColor* represents active ratio color, is of type string.

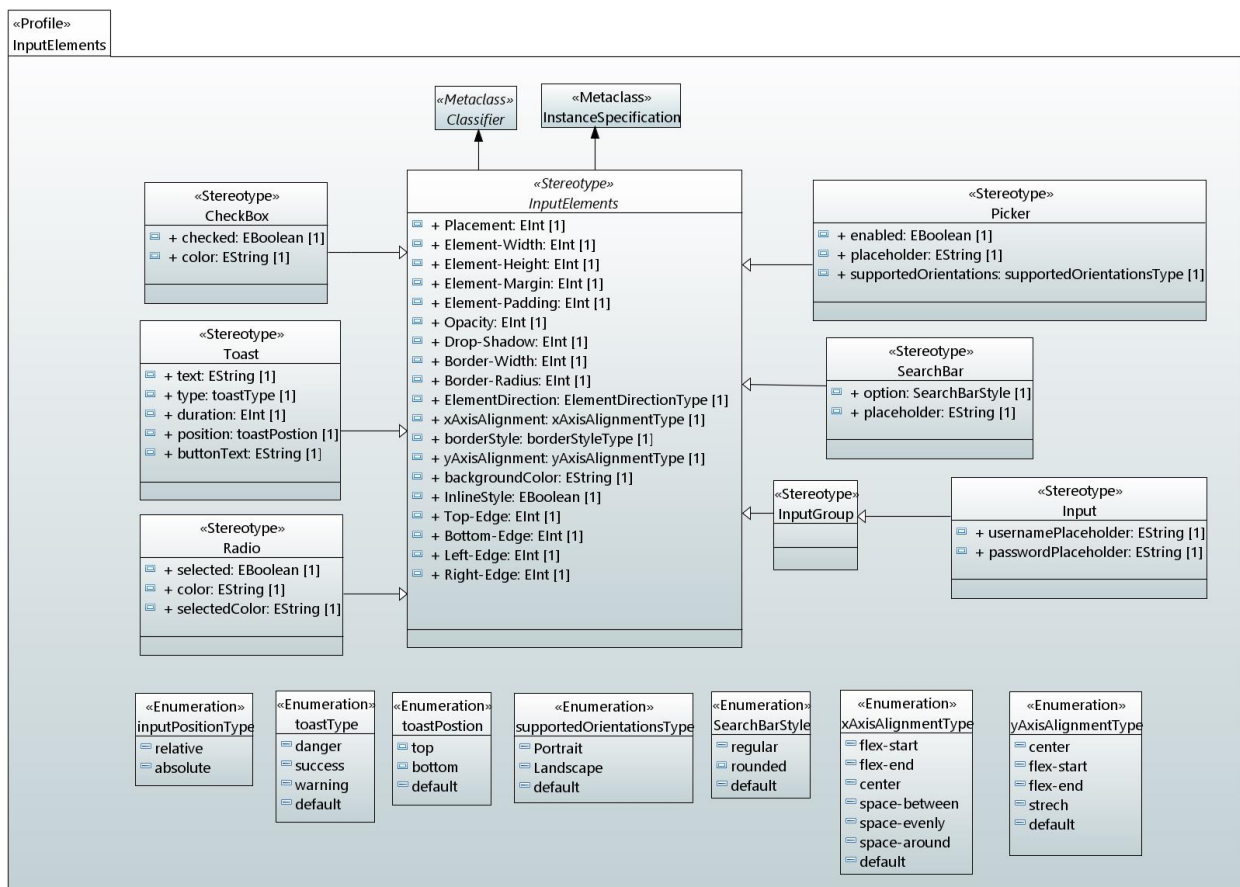


Figure 4: Input Components Profile

- **Checkbox Stereotype**

Description: Checkbox stereotype allows user to select number of items, from set of choices.

Base Class: Instance Specification, Classifier
Tagged Values: *Checked* is of type boolean, states value of an item from set of choices.
Color sets the color of the check box, is of type string.

- **Toast Stereotype**

Description: toast stereotype can be used to display an error messages or a quick warning

Base Class: Instance Specification, Classifier

Tagged Values: *Text* is for text to be shown in the toast, is of type string. *Duration* is of type integer needs to be set in milliseconds after which the toast disappears. *buttonText* of type string, text to be displayed inside the button. *Type* and *position* are enumerations. *Type* sets the context of the toast to danger, success or warning *position* sets the position of the toast to top or bottom.

- **Input Group/Input Stereotype**

Description: Input Group and Input are a foundational element for inputting a text via keyboard into application

Base Class: Instance Specification, Classifier

Tagged Values: *usernamePlaceholder* and *passwordPlaceholder* are strings that renders before text input is entered.

- **Searchbar Stereotype**

Description: Search bar stereotype is essential part of every almost application, user resort to searching if user fail to get what they are looking for.

Base Class: Instance Specification, Classifier

Tagged Values: *Option* tagged value is used to wrap the search bar with the border options by enumeration called *SearchbarStyle* having two enumeration literals regular and rounded.

3.3. Static Elements Profile

Static Elements: Static elements display a screen element without providing behavior or interaction, i.e. a functionality that does not react on user input or other events. **Figure 5** shows the static elements profile. Following are the static elements with their description.

- **Image Stereotype**

Descriptions: Image stereotype is used for displaying different types of images. Such as from local disk or network.

Base Class: Instance Specification, Classifier

Tagged Values: *Source* tagged value is used to specify the source of an image from local disk or network images through data type string.

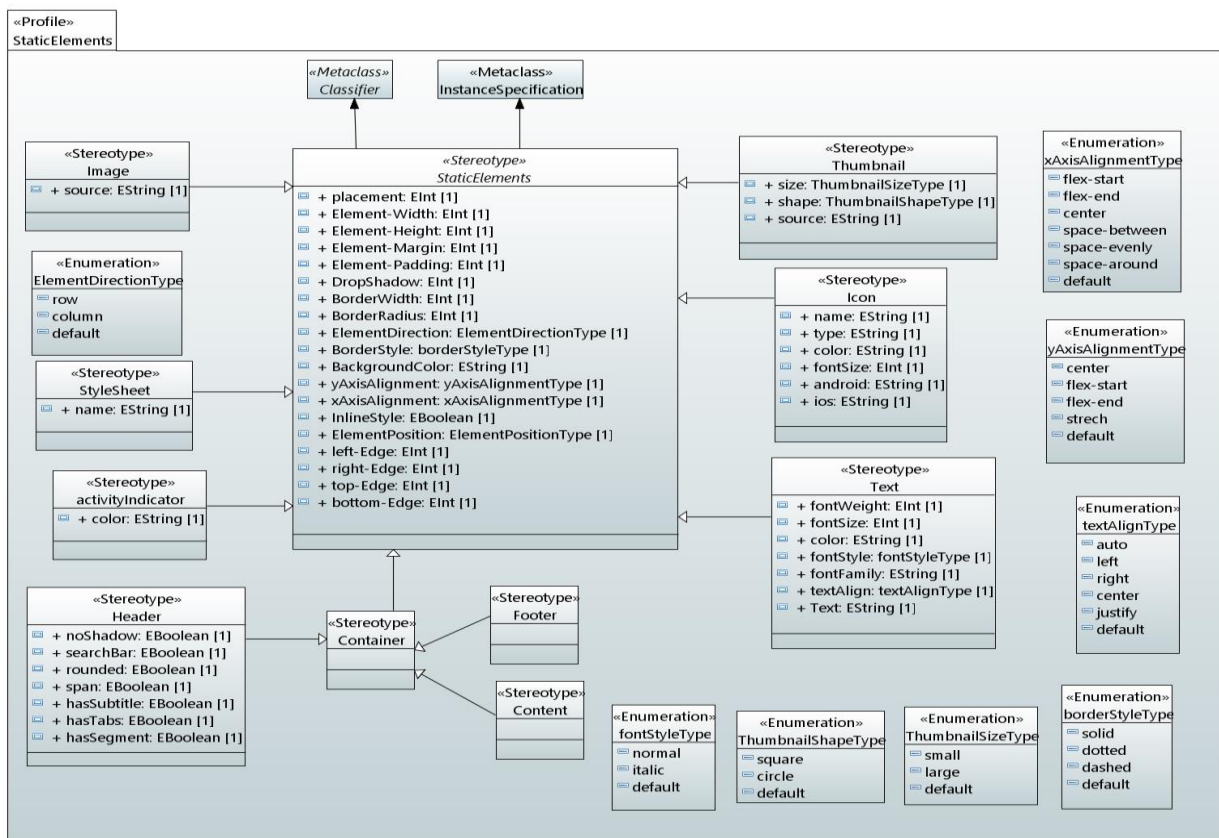


Figure 5: Static Elements Profile

- **Text Stereotype**

Description: Text stereotype is used for displaying text and it supports touch handling, nesting and styling.

Base Class: Instance Specification, Classifier

Tagged Values: *fontWeight* specifies the *fontWeight* through integer datatype. *fontSize* this

Values: tagged value specifies the *fontSize* through string data type. *Color* specifies the text color with data type string. *fontStyle* tagged value

specifies the `fontStyle` with enumeration `fontStyleType` having enumeration literals `normal`, `italic`. `textAlign` specifies the alignment of text to `auto`, `left`, `center` or `right` by enumeration `textAlignmentType`. `fontFamily` is of data type string and it is used to specify the font family.

- **StyleSheet Stereotype**

Description: StyleSheet stereotype represents collection of style rules like CSS stylesheet.

Base Class: Instance Specification, Classifier

Tagged Values: StyleSheet stereotype inherit tagged values from the superclass called Element, which have a series of properties relative to the size and the position. `hairlineWidth` This constant will always be a round number of pixels and will try to match the standard width of a thin line on the underlying platform. `absoluteFill` can be used for convenience and to reduce duplication of these repeated styles.

- **Icon Stereotype**

Description: An icon stereotype is for a small picture, or a symbol used to represent something

Base Class: Instance Specification, Classifier

Tagged Values: Tagged value `name` is used to specify the name of the icon through datatype string. `type` tagged value is used to specify the type of an icon such as `IonIcons`, `EvilIcons`, `MaterialIcons` etc. through datatype string. `color` is of type string and it is used to specify the color of an icon. `fontSize` can be used to `render` the icon with defined icon-size with data type integer. `android` and `iOS` tagged value is used to specify the name of the icon for the corresponding devices with the data type string.

- **Thumbnail Stereotype**

Description: Thumb stereotype helps you to display image with various dimensions and shape, it renders an image by default it is in circular shape.

Base Class: Instance Specification, Classifier

Tagged Values: `Source` tagged value is used to specify the image path for the thumbnail through string data type. `size` tagged value is used to specify the size of the thumbnail to be `small` and `large` with user-defined type enumeration.

shape tagged value is used to specify the shape of the thumbnail to be circle or square with user-defined type enumeration.

- **Container Stereotype**

Description: All elements should be contained in containers, mainly comprises of three types header, content and footer. It comes with predefined stylesheet we can also add user defined styles.

Base Class: Instance Specification, Classifier

Tagged Values: Container stereotype inherit tagged values from the superclass called Element, which have a series of properties relative to the size and the position.

- **Header Stereotype**

Description: Header stereotype can be used to provide you with a stylesheet or you can also add custom styles to it. Include header element in a container.

Base Class: Instance Specification, Classifier

Tagged Values: *noShadow* tagged value of data type integer is for removing the shadow. Tagged value *searchBar* is for adding search bar to the header or not through data type boolean. *rounded* tagged value is for making search bar rounded through data type boolean. tagged value *span* can be used for doubling the header size through data type boolean. *hasSubtitle* is of type boolean and is used to add subtitle to the header. *hasTabs* is used to add tabs to the header through datatype boolean. *hasSegment* tagged value adds segment to the header through data type boolean.

3.4. Dynamic Elements Profile

Dynamic Elements: These elements can be activated that defines a functionality that has interactive features such as reacts to input by the user or other events, and can cause the display of another screen area e.g. Form, navigation. **Figure 6** shows the dynamic elements profile. Following are the dynamic elements with their description:

- **Navigation Stereotype**

Description: Navigation stereotype is for screen to navigate to other screens because an application is composed of several screens with which users can interact.

Base Class: Instance Specification, Classifier

Tagged Values: Tagged value *name* can be used to specify the name of the navigator through data type string.

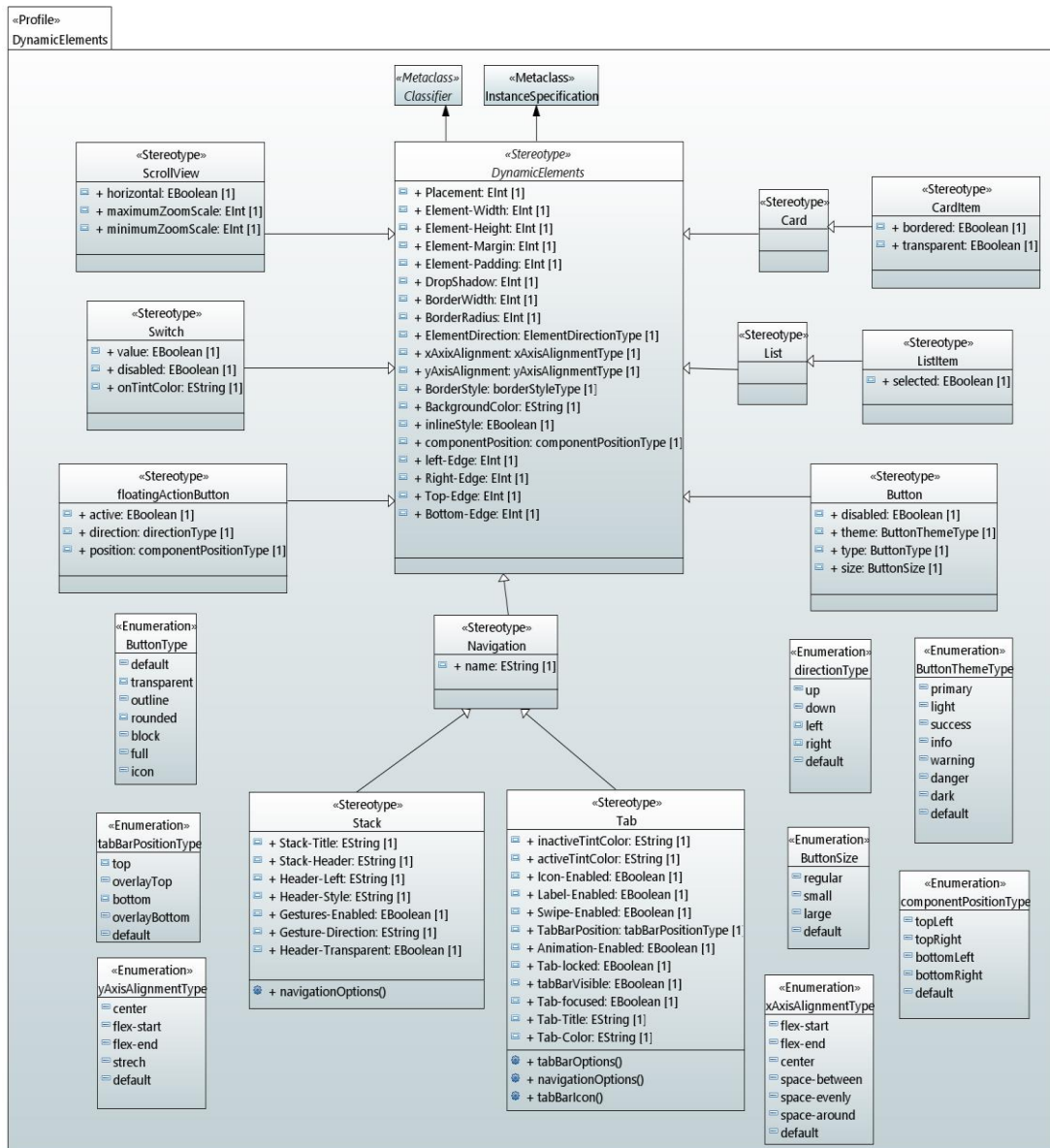


Figure 6: Dynamic Elements Profile

- **Scroll View Stereotype:**

Description: Scroll view stereotype represents a generic container that can host a multiple elements.

Base Class: Instance Specification, Classifier

Tagged Values: *horizontal* tagged value is of data type boolean and is used for scrolling vertically and horizontally. *maximumZoomScale* and *minimumZoomScale* is for allowing the user to zoom content in and out through data type integer.

- **Switch Stereotype:**

Description: Switch stereotype works like toggle, it renders a Boolean input

Base Class: Instance Specification, Classifier

Tagged Values: *value* tagged value is of data type string and is used for specifying the value of switch. *disabled* is for toggling the switch to be true or false with data type boolean *onTintColor* is for specifying the tint color on switch through data type string.

- **Button Stereotype**

Description: Button stereotype is for an integral part of an application and can be used for various purposes like performing interactive actions navigate on click of a button show or hide something, reset or submit a form etc.

Base Class: Instance Specification, Classifier

Tagged Values: *Disabled* is of data type boolean and this can be used to disable the click option for the button. *Size* tagged value can be used to specify the size of the button through enumeration *ButtonSize* having enumeration literals primary, light, success, info, warning, danger, dark. *Theme* tagged value can be used to specify the theme of the button through enumeration *ButtonThemeType* having enumeration literals primary, light, success, info, warning, danger, dark. *Type* tagged value can be used to specify the type of the button through enumeration *ButtonType* having enumeration literals transparent, outline, rounded, block, full, icon, defaultType.

- **Stack Stereotype**

Description: Stack stereotype can be used on screen to navigate forward and backwards only.

Base Class: Instance Specification, Classifier

Tagged Values: *title* tagged value is used to specifies the title of the navigator with the data type string. *header* is used to specifies the header of the navigator, *headerStyle* specifies the header style of the navigator through data type string. *gesturesEnabled* is used to select whether the gestures to be enabled or not through data type boolean. *gesturesDirection* is of data type boolean and is used to specify the direction of the gestures *headerTransparent* is used to select whether the header of the navigator to be transparent or not with data type boolean.

- **Tab Stereotype**

Description: Tab stereotype can be used to access multiple tabs on the same screen.

Base Class: Instance Specification, Classifier

Tagged Values: *showIcon* tagged value is used to specify whether to enable the icon to be shown or not. *showLabel* is used to specify whether to enable the label to be shown or not. *swipeEnabled* is used specify whether to enable the swipe or not. *animationEnabled* is used specify whether to enable the animation or not. *Locked* is used to specify whether to locked or not. *Focused* is used to specify the navigator to be focused or not. All the above tagged values have data type boolean. *activeTintColor* specifies the active tint color with data type string. *tabBarPosition* gives the option to select different options for tab bar position with enumeration *tabBarPositionType* having enumeration literals top, overlayTop, bottom, overlayBottom

- **Card/ Card Item Stereotype:**

Description: Card and CardItem stereotype is used to include options for powerful display, contextual background colors and support a wide variety of content like icon, images, links, text and more. CardItem is the child element of card and card can take any number of carditems.

Base Class: Instance Specification, Classifier

Tagged Values: *bordered* tagged value is for adding border to carditems having data type boolean. Tagged value *transparent* is for removing the shadow from card with data type boolean.

- **floatingActionButton Stereotype**

Description: floatingActionButton stereotype is used for special type of promoted action. They are fixed above the user interface in a fixed position and are distinguished by special motion behavior and circled icon.

Base Class: Instance Specification, Classifier

Tagged Values: *Active* tagged value is for toggling the status with data type boolean. *direction* is for specifying the direction of buttons that popup on click of FAB with enumeration directionType. *Position* is for positioning of FAB on screen with enumeration positionType having enumeration literals topLeft, topRight, bottomLeft, bottomRight.

- **List/ ListItem Stereotype**

Description: List stereotype is used for a continuous group of an images or text and must contain one or more list items. Adds border at the bottom of each ListItem.

Base Class: Instance Specification, Classifier

Tagged Values: *Selected* tagged value is for highlighting the selected list item by data type boolean.

Chapter 4

Implementation

CHAPTER 4: IMPLEMENTATION

This chapter presents the implementation details for Automated MMUI (Model driven multi-platform Mobile User Interfaces) tool. The tool we have implemented has three main features. Firstly, it provides facility to user interface models in Eclipse editor. Secondly, it provides a transformation engine that transforms the high-level UMUI models into low-level user interface implementation. Main interface of MMUI Transformation Tool is shown in **Figure 7**. MDE has become the basis of our work. **Section 4.1** presents the transformation rules used to develop MMUI Tool and **Section 4.2** discusses the architecture of transformation engine.

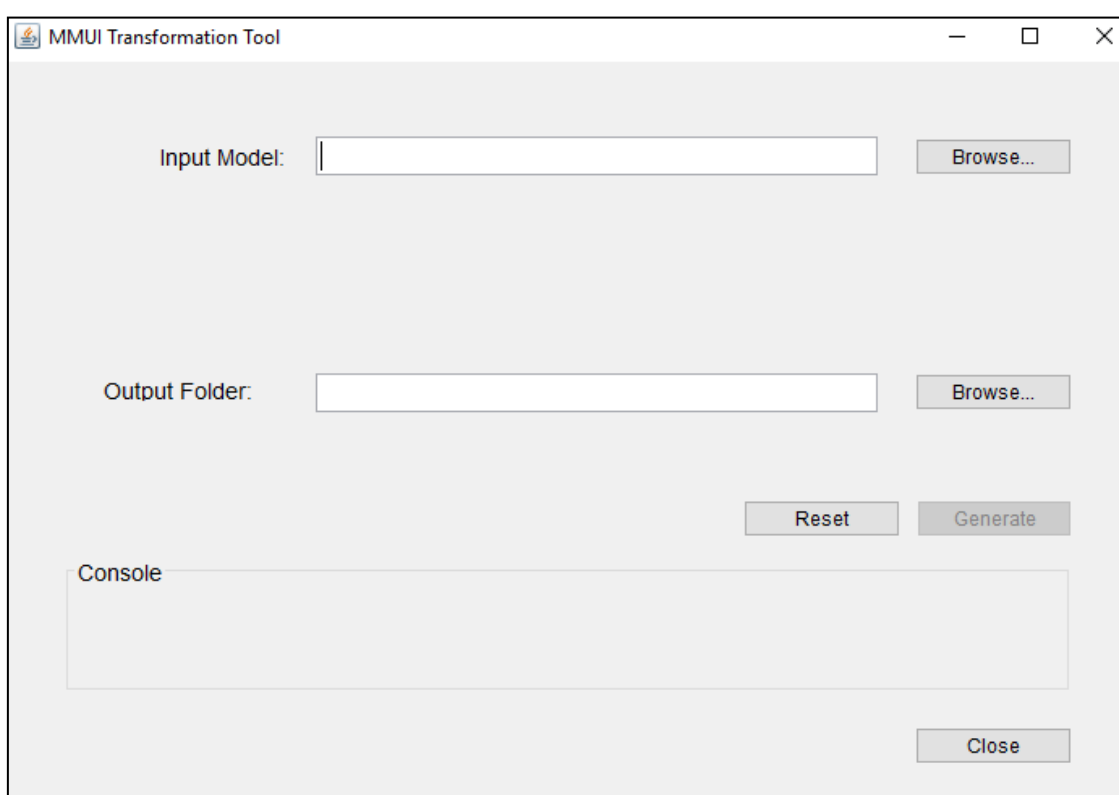


Figure 7: Main Interface of Model based Multi-Platform UI Transformation Tool

4.1. Transformation Rules

We have described the transformation rules in detail, which we have defined in order to transform the models to code artifacts. Mapping rules used for transformation of user interface model components into text are provided in this section. The transformation rules defined in **Table 1** are used to transform the UI components from model to code is denoted as M2T transformation. First column represents the UMUI concept, second column represents the corresponding react native concept and last column represents the description.

Table 1: Transformation rules

UMUI Concept	Corresponding REACT NATIVE Concept	Description
Element	Component	<p>Element → Component Placement → flex Element-Width → width Element-Height → height dropShadow → elevation Element-Direction → flexDirection Element-Padding → padding Element-Margin → margin yAxisAlignment → alignItems xAxisAlignment → justifyContent left-Edge → Left right-Edge → Right top-Edge → Top bottom-Edge → Bottom Element-BorderWidth → borderWidth Element-BorderRadius → borderRadius Element-BorderStyle → borderStyle</p>
Element-Import	Import	<p>Element-Import → Import Element1 → Component1 Element2 → Component2</p>
		<p>Responsive-Screen → MediaQuery screenWidth → deviceWidth minScreenWidth → minDeviceWidth</p>

Responsive-Screen	MediaQuery	maxScreenWidth→maxDeviceWidth screenHeight→deviceHeight minScreenHeight→minDeviceHeight maxScreenHeight→maxDeviceHeight screenPixelRatio→devicePixelRatio minScreenPixelRatio→minDevicePixelRatio maxScreenPixelRatio→maxDevicePixelRatio
Container	View	Container→View
List ListItem	List ListItem	List→List ListItem→ListItem ListLabel→label ListDisabled→disabled ListTitle→title
Navigation	TabNavigator	Navigation → TabNavigator Navigator-Name→name LabelEnabled→ShowLabel IconEnabled→ShowIcon AnimationEnabled→ShowAnimation GestureEnabled→ShowGestures inactiveTintColor→inactiveTintColor activeTintColor→activeTintColor tabNavigationPosition→tabBarPosition TabTitle→title Nav-color→color

Button	Button	<p>Button → Button ButtonTheme → theme ButtonDisabled → disabled ButtonType → type ButtonSize → size</p>
Text	Text	<p>Text → Text TextWeight → fontWeight TextSize → fontSize TextStyle → fontStyle TextColor → color alignmentOfText → alignText</p>
Icon	Icon	<p>Icon → Icon IconName → name IconType → type IconColor → color IconFamily → type IconSize → fontSize</p>
Image	Image	<p>Image → Image ImageSource → source</p>
InputGroup Input	InputGroup Input	<p>InputGroup → InputGroup Input → Input Username → usernamePlaceholder Password → passwordPlaceholder</p>

StyleSheet	StyleSheet	StyleSheet → StyleSheet Name → name
Card CardItem	Card CardItem	Card → Card CardItem → CardItem transparentCard → transparent selectedCard → selected
ActivityIndicator	Spinner	ActivityIndicator → Spinner indicatorColor → color

Element of the UMUI concept which is extended from meta class classifier and instance specification is mapped to Component of corresponding react native concept. Tagged values such as xAxisAlignment and yAxisAlignment are mapped to justifyContent and alignItem which are corresponding react native concepts. Name of Enumeration is mapped to name of Enum in code and Enumeration Literal is mapped to user defined Enum values. Similarly, other concepts of UMUI are mapped to corresponding react native concepts as shown on the table above.

4.2. Transformation Engine Architecture

Architecture of our transformation engine is described in detail in **Figure 8**. We have implemented a transformation engine based on model-based code generation idea. This transformation engine fully automates the user interfaces from models. Tool used for transformation of high-level UMUI models to low-level implementation is Eclipse Acceleo. Our transformation engine is composed of three main components which are Main Interface, Acceleo Transformation and Java Services. Details of functionality performed by each component is explained below.

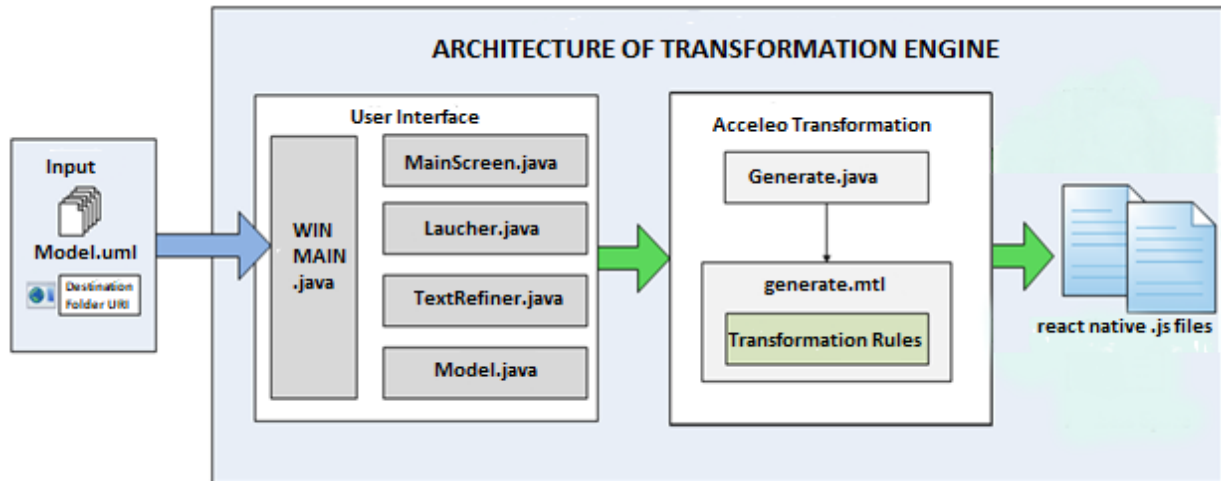


Figure 8: Transformation Engine Architecture

User Interface: User Interface component consists of three classes i.e. *MainScreen*, *Launcher* and *WinMain*. These three classes are used in development of graphical user interface of our tool. *MainScreen* is a class which provides execution and *WinMain* and *Launcher* contain its actual functionality. The transformation engine takes *UML* models as input using a *Browse* button. It also asks for path of *Destination folder*. By clicking the *Generate* button, the engine generates the files. To represent if the transformation has been performed successfully or if some error has occurred while transformation, a *Console* bar is shown. *Reset* button, empties all the fields i.e. input models path, destination folder path, status and all checkboxes except for the test case document checkbox which is by default checked. *Close* button is provided to close the interface from the screen as shown in **Figure 9**.

Acceleo Transformation: Main interface takes *UML* models as input and passes them to *Acceleo Transformation*. Foremost files included in *Acceleo Transformation* are *Generate.java* containing java code for transformation and *generate.mtl* containing acceleo transformation code. These two files work together to produce java script document (.js) which contains all the UI code related concepts captured from model.

Java Services: Java Services are developed using two main classes, *JavaServices* and *Model*. *Model* class only used to get the input model and instantiates output. *JavaServices* is the main class we have used to store every detail needed for our helper functions as *Acceleo* does not provide facility in easy way inside the *mtl* file. The functions defined in *JavaServices* class can then be used inside *generate.mtl* file using queries.

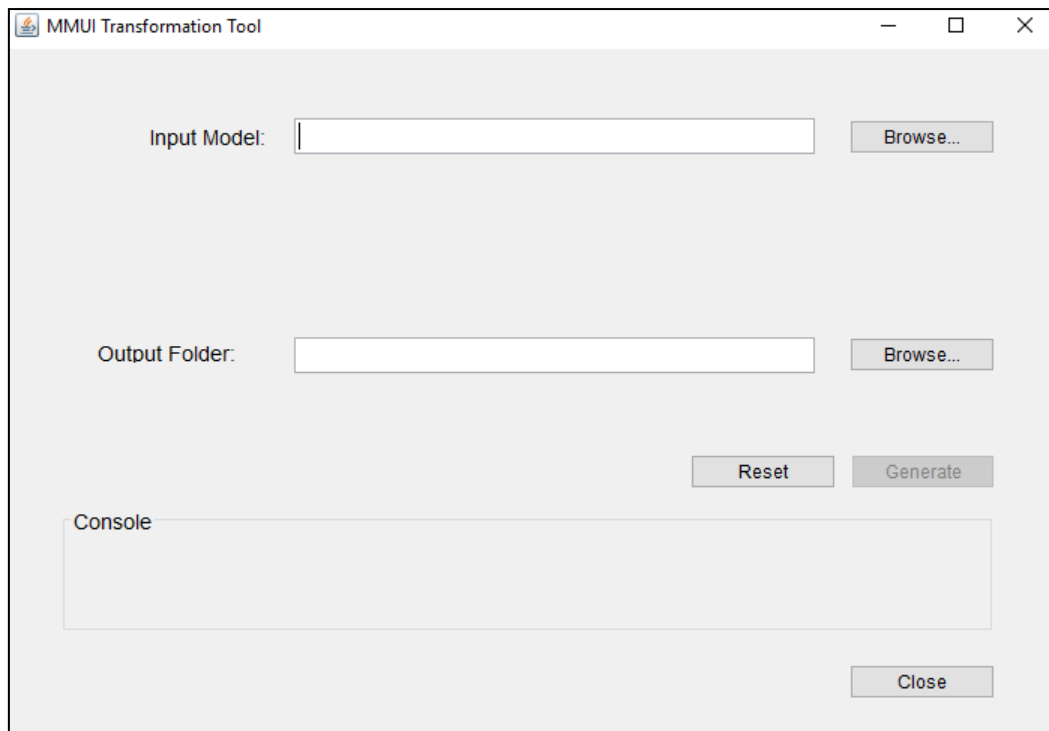


Figure 9: Transformation Tool

Chapter 5

Validation

CHAPTER 5: VALIDATION

In this chapter, the applicability and validity of our proposed approach is presented with the help of two detailed case studies. Details and validation of Patient Management case study is given in **Section 5.1** and Library Application is given in **Section 5.2**

5.1. Patient Management System Case Study

Patients Management System case study has been explained and validated using four sections. **Section 5.1.1** covers the requirement specification for an application named as Patient Management System. **Section 5.1.2** presents UMUI application, the UML object diagram of this case study with applied profile in Eclipse editor using its Papyrus plugin. Lastly, the transformation in Aceleo and verification has been provided in **Section 5.1.3** and **Section 5.1.4** respectively.

5.1.1. Requirement Specification

Following are the details of the mobile user interfaces and their specifications included in the Patient Management System case study.

Home Interface: The Home user interface should have a navigator through which each user interface should be accessible from all other user interfaces that exist in that application.

Login Interface: Login interface should contain a section with a login in form containing the input placeholders for Email and Password. Person and lock icons, Login Button should be attached to this form.

Register Interface: Register interface should contain a section with a registration form containing all the necessary information such as name, age, gender, address etc. Register interface should be accessible from all other interfaces.

Status Interface: Status interface should contain a section to show the current status of the patient and allow the nurse or doctor to update it. Status interface should be accessible from all other interfaces.

Patients Information Interface: This interface contains a section showing a list of patients containing name, token number and status with additional file attached of patient history. This interface should also be accessible from all other user interfaces that exist in that application.

5.1.2. Modeling

Our tool provides the option to open Papyrus Editor. By clicking on it, Eclipse environment is opened, and Papyrus model can be designed by creating a new project for Papyrus modeling. UML object diagram containing the main concepts as instance specifications and instance specification links can easily be created in Eclipse Papyrus editor. Object Diagram models of the Patient Management System case study are shown in **Figure 10**, **Figure 11**, **Figure 12**, **Figure 13**, and **Figure 14**.

Object diagrams of Patient Management System screens are designed in Papyrus editor using Eclipse. Main Instances used in Patient Management System are View, Icon, Button, Text, StyleSheet, Lists and Navigator. Relationships between all the instances have also been shown. Due to complexity, parts of object diagram of Patient Management case study are shown in several subsequent figures.

In this case, our application is composed of different various screens. Registration view is displayed by the first screen in which user can register the patient and second screen displays the lists of patients and similarly others with their related information. <<Screen>> stereotype which is extended by meta-class model is mapped to corresponding user interface screens. Layout of each screen is set with applying responsive-screen stereotype by setting screen width and screen-height. This stereotype is mapped to corresponding react native concept called media query. This component is like any other react native component with props. These allows to set screen width and height to make it good on all devices. For the user interface react native view, we use the stereotype <<Container>> mapped to the registration view, patient view and other screens views accordingly. it is a container of all the user interface components. The home user interface has a navigator through which each user interface should be accessible from all other user interfaces that exist in that application. So <<Navigation>> stereotype is mapped to TabNavigator because it can be used to access multiple tabs on the same screen. <<Button>> stereotype is mapped to register button instance specification having inline styles such as disabled which is Boolean and set to false and theme which is set to be light etc. <<Import>> is applied for specifying the directory path of screens if one want to import file from another folder or want to import packages from node package manager. Other stereotypes such as <<Text>>, <<Image>>, <<Icon>>, <<Card>>, <<CardItem>>, <<Input>>, <<List>>, <<ListItem>> and <<StyleSheet>> are mapped to their corresponding instance specifications. Relationships between all the instances have also been shown. **Figure 15** shows how the values are assigned to instances of classes in papyrus.

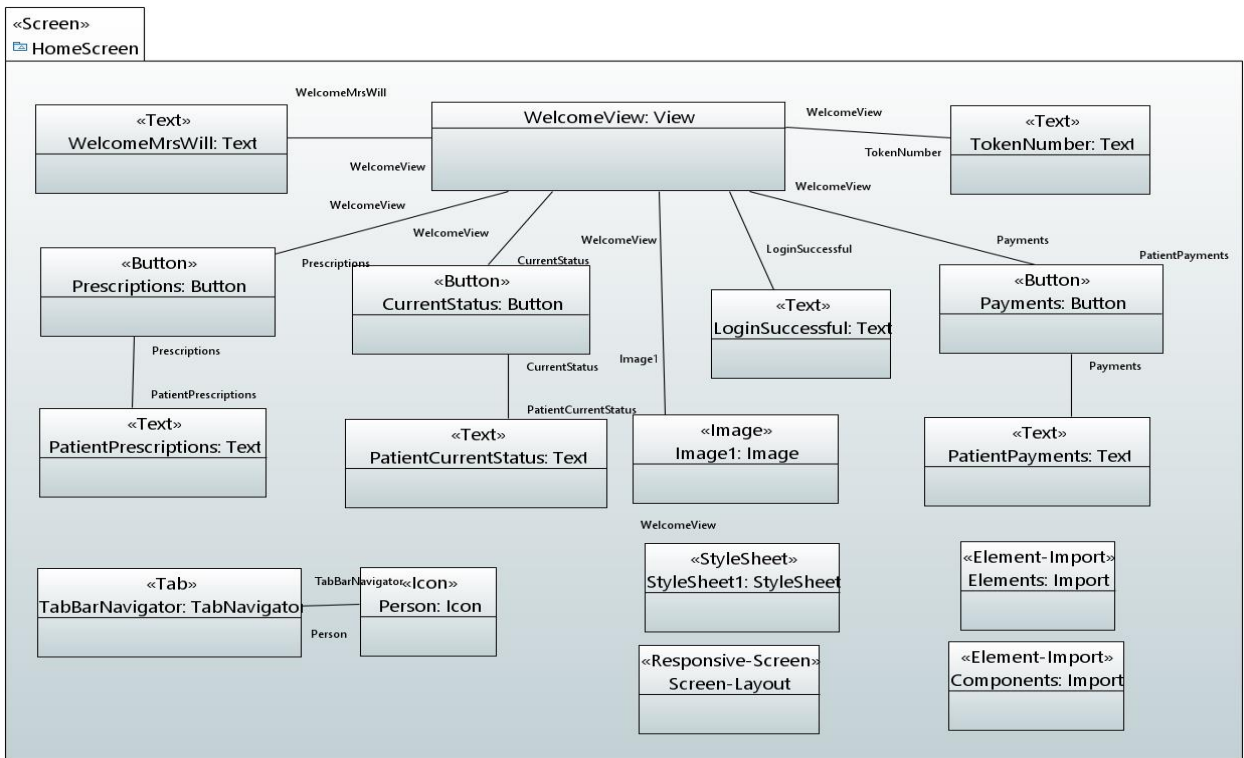


Figure 10: Object diagram of Patient Management System (Home Screen)

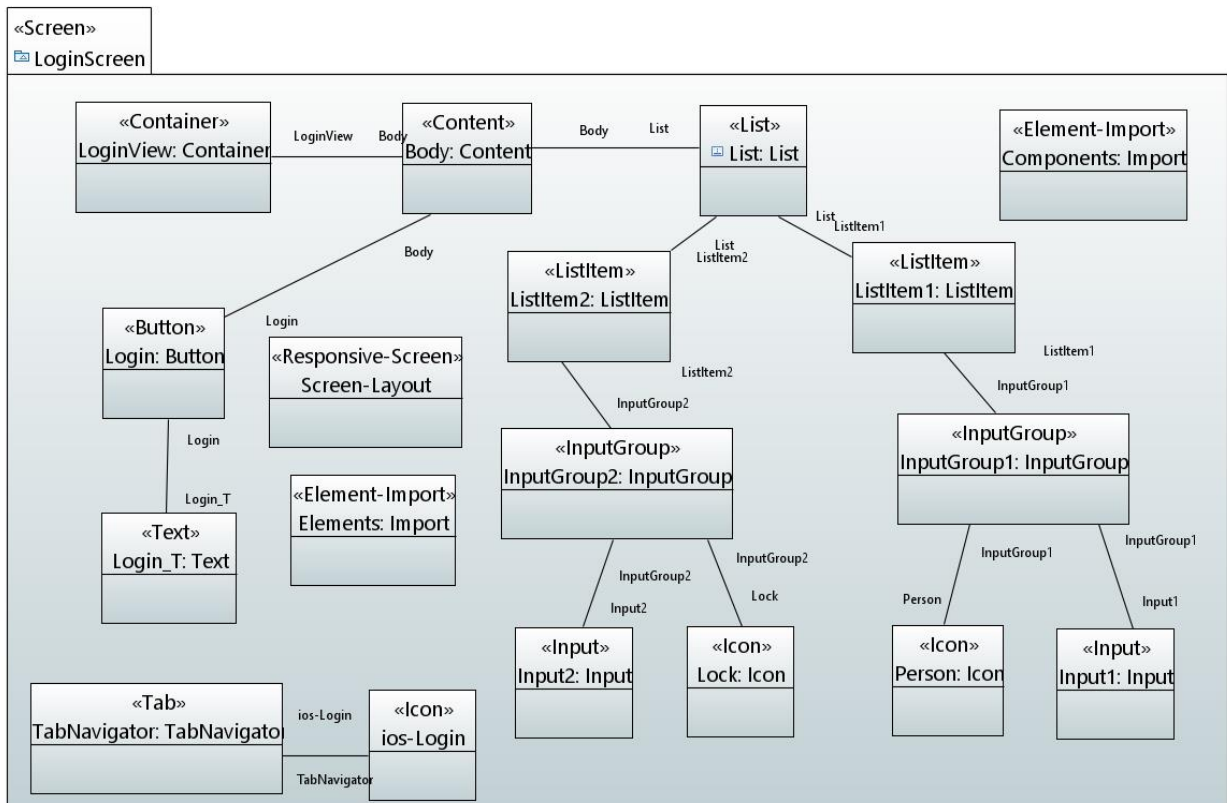


Figure 11: Object diagram of Patient Management System (Login Screen)

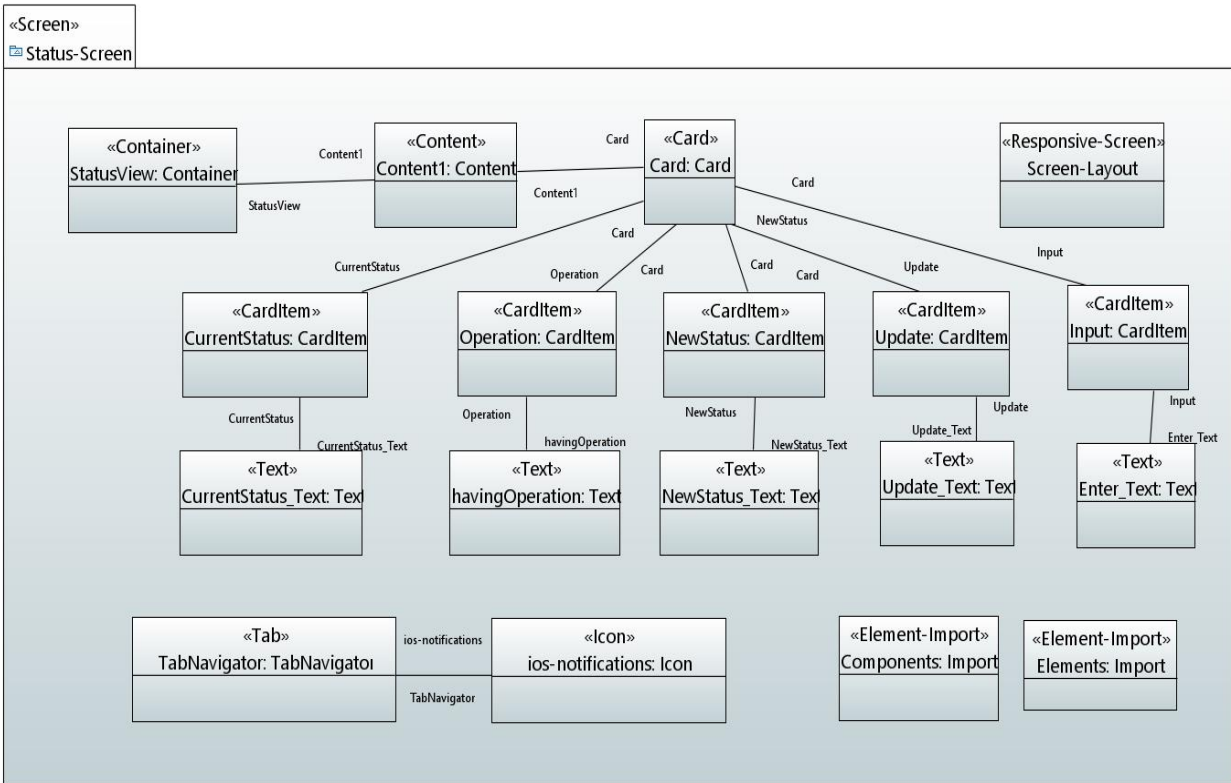


Figure 12: Object diagram of Patient Management System (Status Screen)

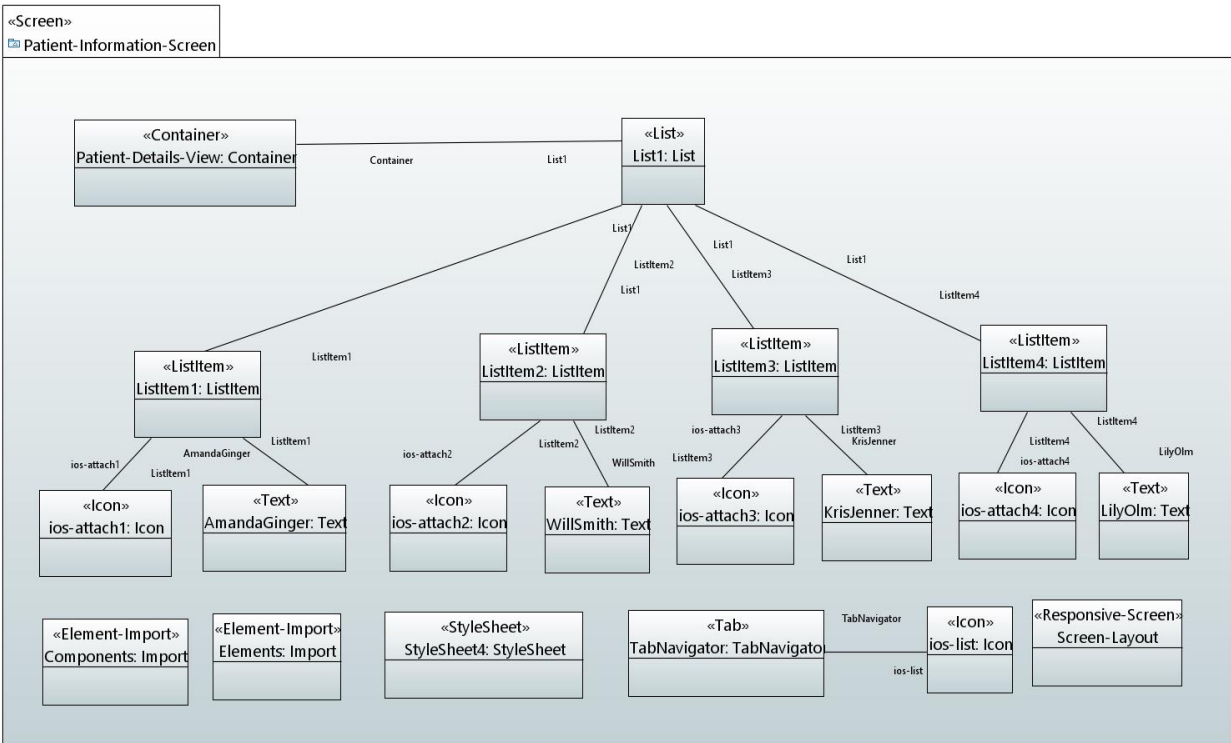


Figure 13: Object diagram of Patient Management System (Patient Information Screen)

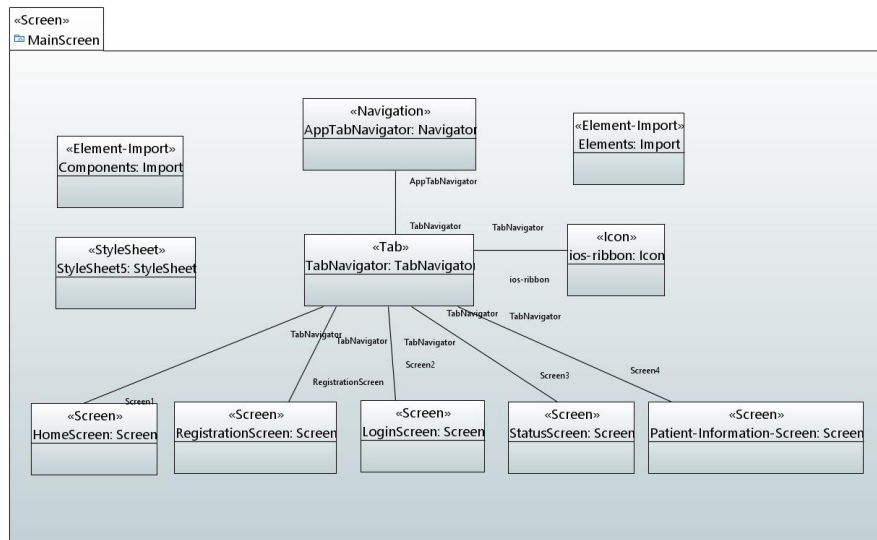


Figure 14: Object diagram of Patient Management System (Main Screen)

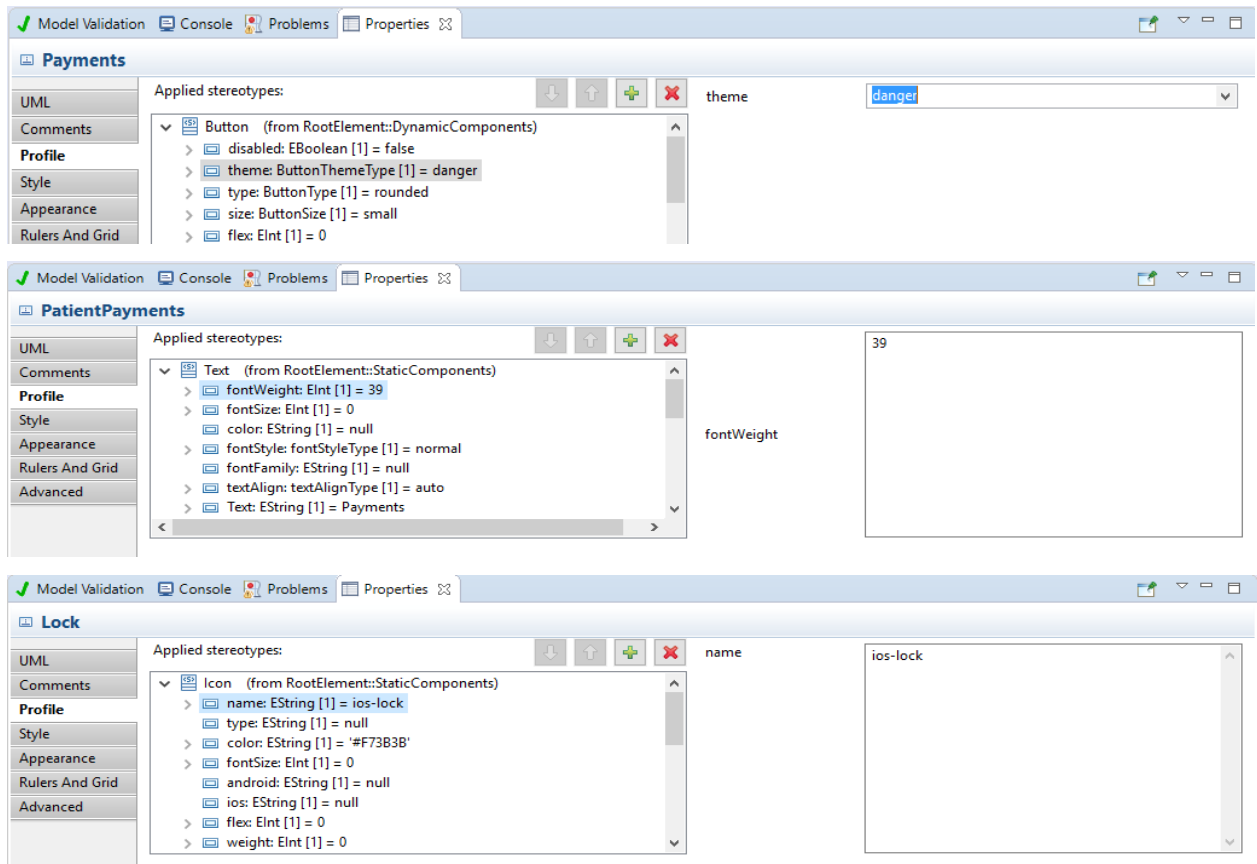


Figure 15: Assigning Values to Instance Specifications

In next step, we used the UML approach, to transform the above instance model representing the user interface of our case study into a code. After the mapping, we are going to define the corresponding rules using acceleo

5.1.3. Code Generation

The case study model of the screens is given as input to MMUI transformation tool whose interface is shown in **Figure 16**. The transformation tool uses the transformation rules for mapping and transformation models into code. UML model with .uml extension is selected as input model and target folder on desktop is provided as a destination folder for generating code files. On clicking the generate button, output will be generated in the target folder and console shows the status of the transformation process as a result **Figure 17** shows the desired output of user interfaces code.

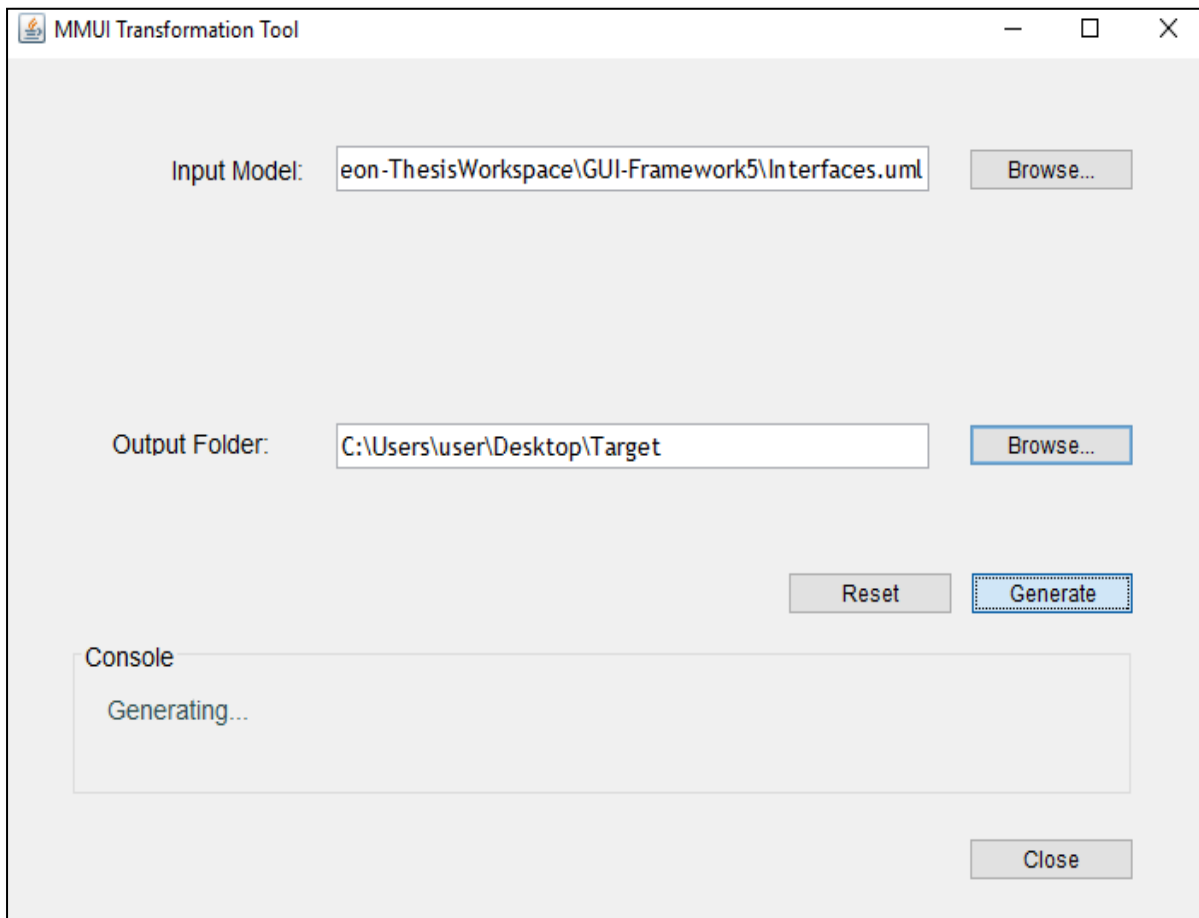


Figure 16: Transformation for User Interfaces of Patient Management System Application






Name	Date modified	Type
 HomeScreen.js	11/9/2018 3:23 PM	JavaScript File
 MainScreen.js	11/9/2018 3:24 PM	JavaScript File
 Patient-Information-Screen.js	11/9/2018 3:23 PM	JavaScript File
 RegistrationScreen.js	11/9/2018 3:24 PM	JavaScript File
 StatusScreen.js	11/9/2018 3:24 PM	JavaScript File

Figure 17: Generated Files of Patient Management System

```
import React from 'react';
import { StyleSheet, Text, View, Button } from 'react-native';

class LoginScreen extends React.Component {
  render() {
    return (
      <View>
        <List>
          <ListItem>
            <InputGroup>
              <Icon name="ios-person" />
              <Input placeholder="EMAIL" />
            </InputGroup>
          </ListItem>
          <ListItem>
            <InputGroup>
              <Icon name="ios-lock" />
              <Input placeholder="PASSWORD" secureTextEntry={true}/>
            </InputGroup>
          </ListItem>
        </List>
      </View>
    );
  }
}
```

Figure 18: Generated Code of User Interfaces of Patient Management System

5.1.4. Verification

For the verification of generated JavaScript code, compilation and execution is necessary. For this purpose, we have used the Expo Local Development Tool (Desktop and Mobile Client) for the compilation and execution of code. We have created the new react native project using Expo XDE as shown in **Figure 19** and pasted our generated code to run the react native packager. After that open the mobile expo client as shown in **Figure 20** and transformation results are shown in **Figure 21, Figure 22, Figure 23, Figure 24** and **Figure 25**.

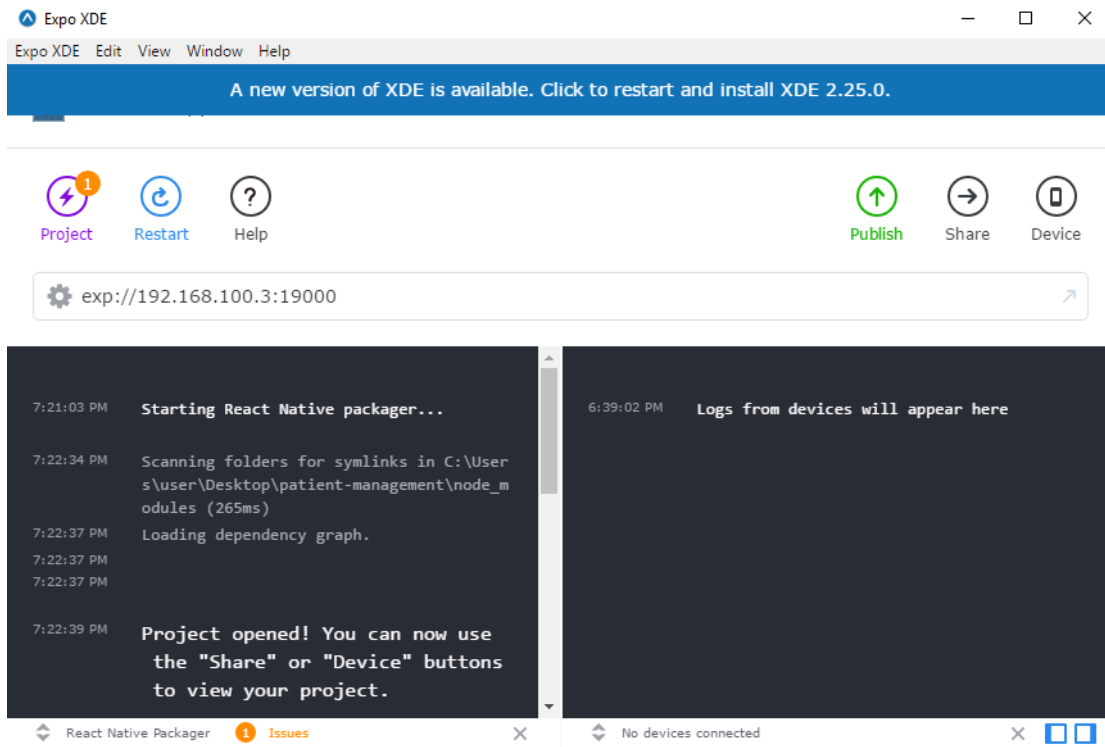


Figure 19: Expo XDE Desktop Application

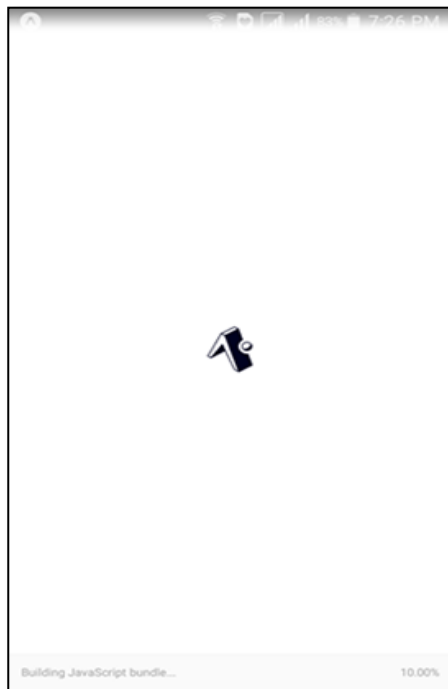


Figure 20: Expo Mobile Client

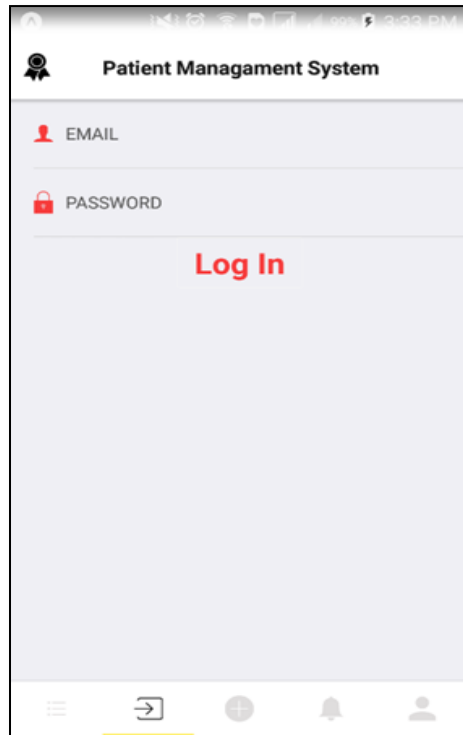


Figure 21: User Interface after Model Transformation (Login Screen)

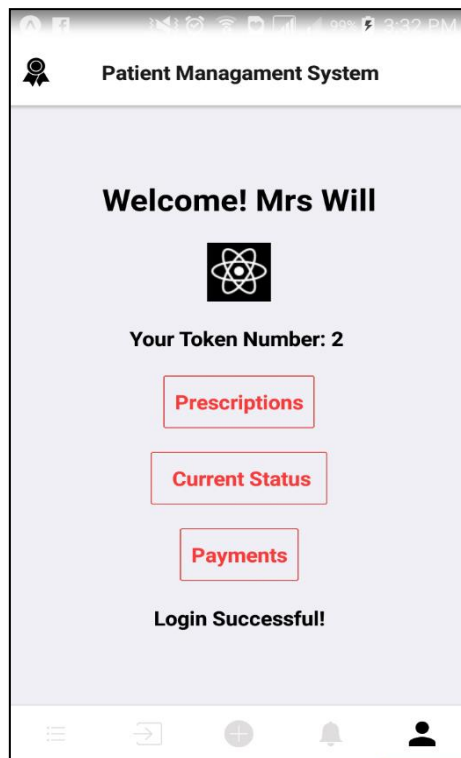


Figure 22: User Interface after Model Transformation (Profile Screen)

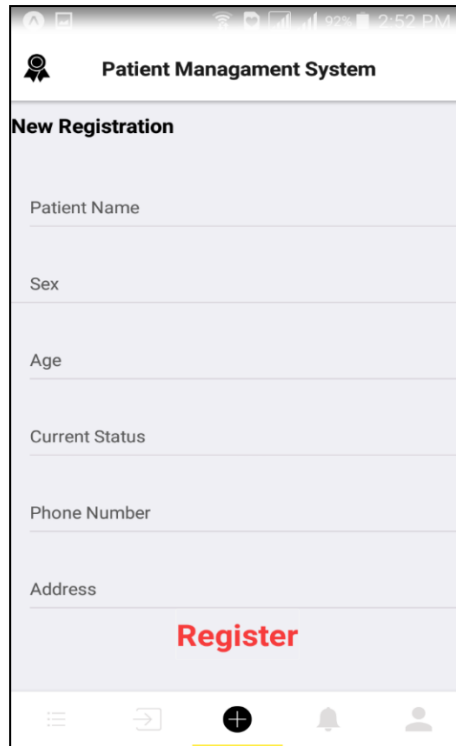


Figure 23: User Interface after Model Transformation (Registration Screen)

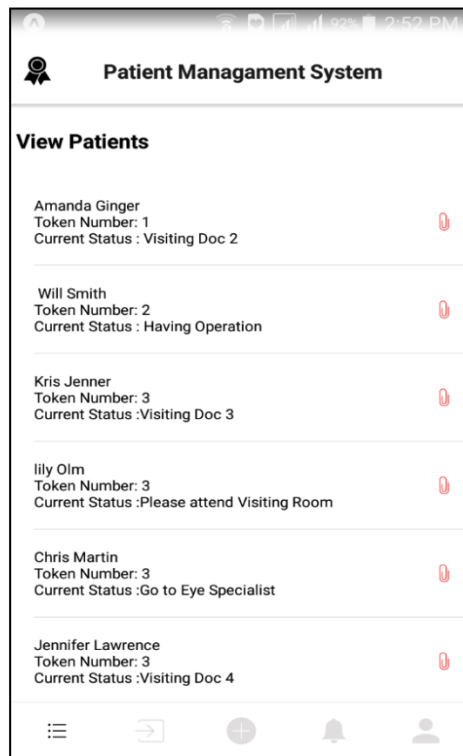


Figure 24: User Interface after Model Transformation (Patient Information Screen)

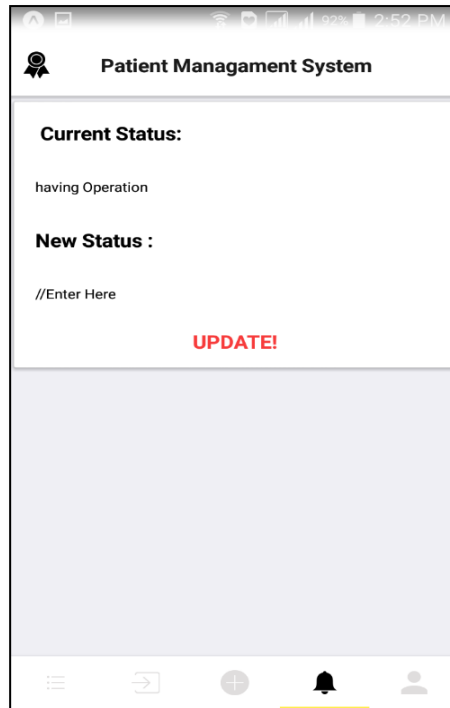


Figure 25: User Interface after Model Transformation (Status Screen)

5.2 Library Application Case Study

Library application assists us providing educational resources online to expedite the finding process by library members using digital library. So, this case study has been explained and validated using four sections. **Section 5.2.1** covers the requirement specification for an application for Library. **Section 5.2.2** presents UMUI application, the UML object diagram of this case study with applied profile in Eclipse editor using its Papyrus plugin. Lastly, the transformation in Acceleo and verification has been provided in **Section 5.2.3** and **Section 5.2.4** respectively.

5.2.1. Requirement Specification

We have taken a piece of Library system. Following are the details of the main interfaces and their specifications included in this case study. Three interfaces have been selected in this case study which are as follows

LOGIN: Login interface should contain a section with a login in form containing the input placeholders for Email and Password. Person and lock icons, Login Button should be attached to this form.

HOME: Home screen contains a section in which library timings, Website, Notice Board and Members etc. are shown in a list along with their Icons Button and Text. Home screen should be accessible from profile screen and can navigate back to login screen.

PROFILE: Profile interface should contain a section containing all the necessary information such as name, name, image, membership details, currently issued books etc. profile screen should be accessible from home screen profile page should be accessible from home screen.

5.2.2. Modeling

Object diagrams of library application screens are designed in Papyrus editor using Eclipse. Main concepts of UMUI applied in library case-study are <<Screen>>, <<Container>>, <<Import>> <<Icon>>, <<Button>>, <<Text>>, <<StyleSheet>>, <<List>>, << ListItem>>, <<Import>>, <<ResponsiveScreen>> and <<Navigation>>. In this case, our application is constituted by different screens. <<Screen>> stereotype which is extended by meta-class model is mapped to corresponding user interface screens. Layout of each screen is set with applying <<Responsive-Screen>> stereotype by setting screen width and screen-height. This stereotype is mapped to corresponding react native concept called media query. This component is like any other react native component with props. These allows to set screen width and height to make it good on all devices. For the user interface react native view, we use the stereotype <<Container>> mapped to the home view, login view and other screens views accordingly. it is a container of all the user interface components. The home user interface has a navigator through which each user interface should be accessible from all other user interfaces that exist in that application. So <<Navigation>> stereotype is mapped to TabNavigator because it can be used to access multiple tabs on the same screen. <<Button>> stereotype is mapped to login button instance specification having inline styles such as disabled which is Boolean and set to false and theme which is set to be light etc. In this case study we have used both kinds of navigation stack and tab. Tab Navigation is at top of screen as compare to previous case study. Import instance is for specifying the directory path of screens if one want to import file from another folder or want to import packages from node package manager. Relationships between all the instances have also been shown. Due to complexity, parts of object diagram of library application case study are shown in several subsequent figures are shown in **Figure 26, Figure 27, Figure 28** and **Figure 29** in next section.

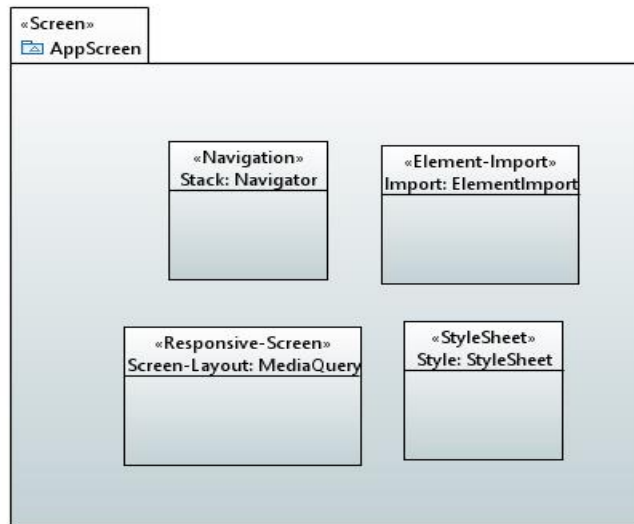


Figure 26: Object Diagram of Library Application Case Study (App Screen)

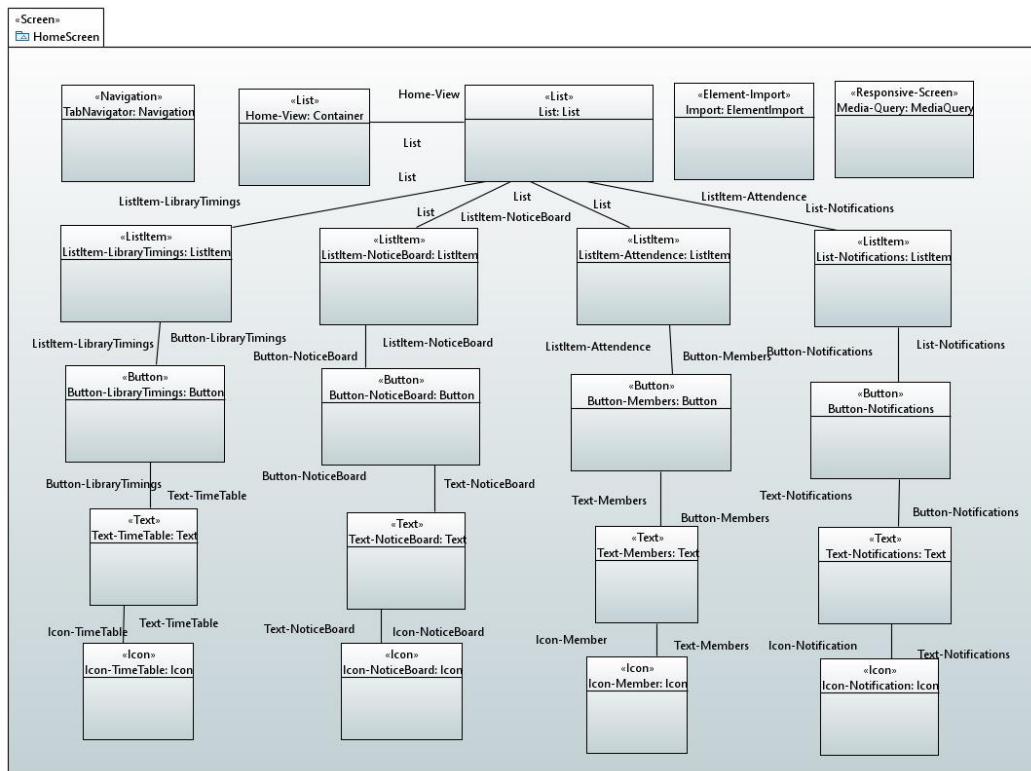


Figure 27: Object Diagram of Library Application Case Study (Home Screen)

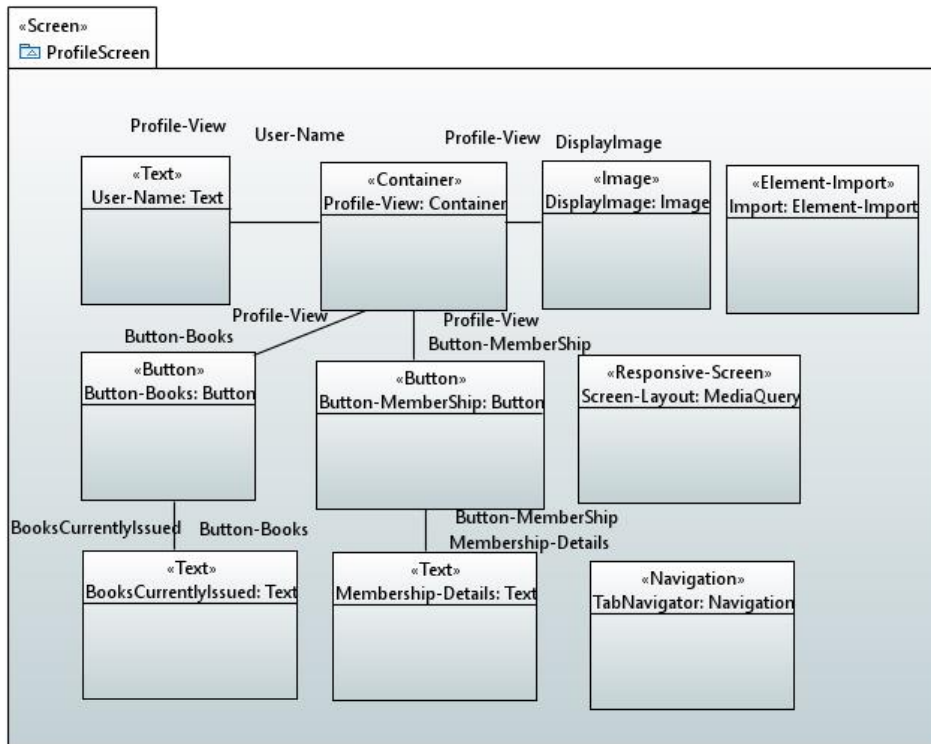


Figure 28: Object Diagram of Library Application Case Study (Profile Screen)

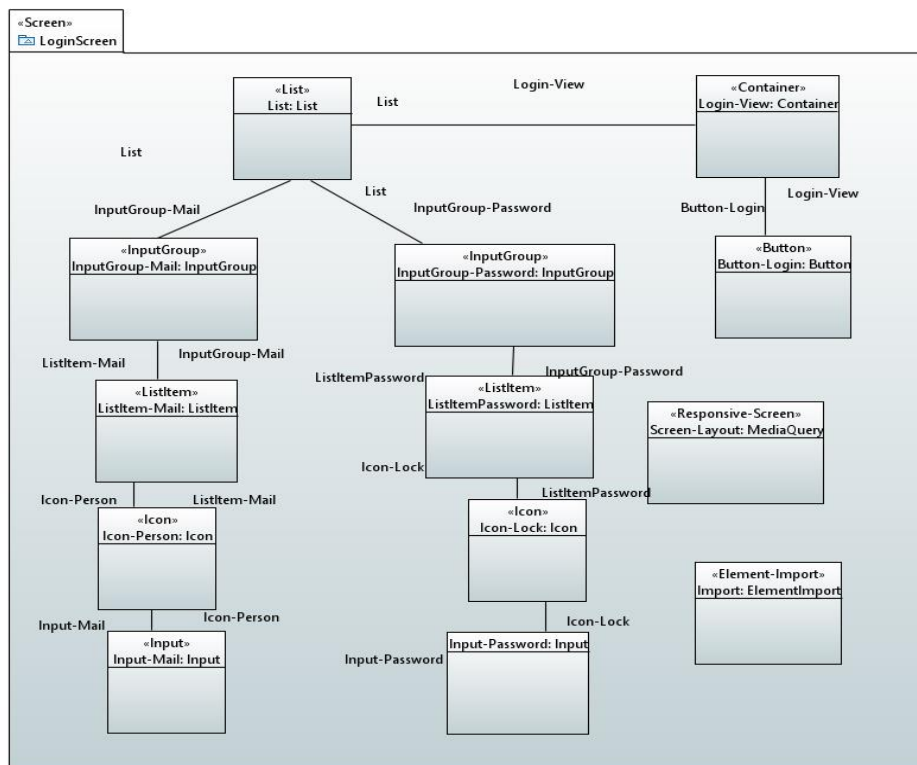


Figure 29: Object Diagram of Library Application Case Study (Login Screen)

5.2.3. Code Generation

The case study model is given as input to our proposed transformation tool whose interface is shown in **Figure 28**. In transformation engine, Mapping rules are used to transform the Model into Code. UML model with .uml extension is selected as input model and destination folder on desktop is provided as a target folder for generating code files. On clicking the generate button output are generated in the target folder and console shows the status of the transformation process as a result. **Figure 31** and **Figure 32** shows the desired output of user interfaces code.

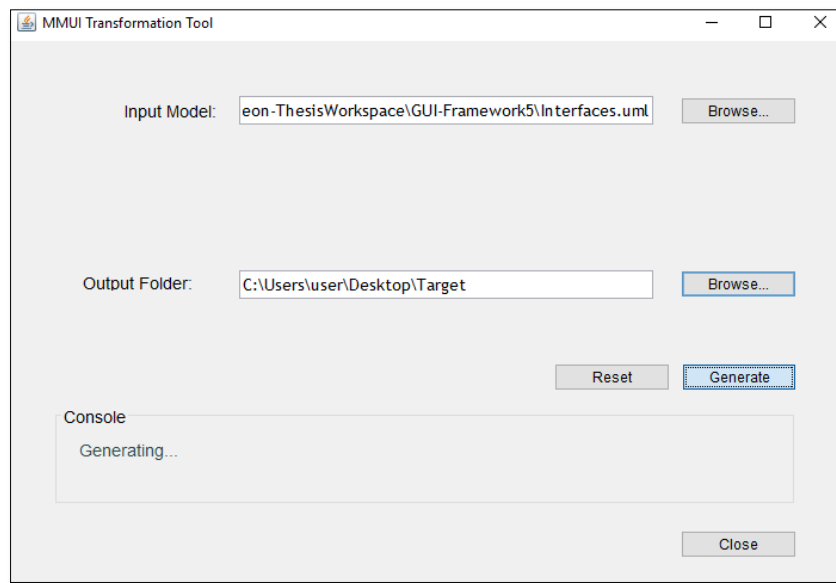


Figure 30: Transformation for User Interfaces of Library Application Case Study

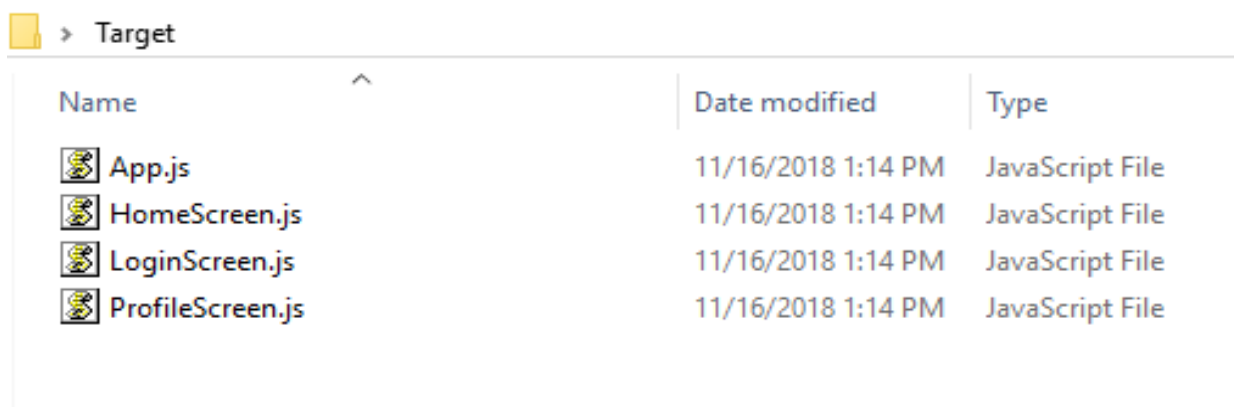


Figure 31: Generated Files of Library Application Case Study

```

1  import React from 'react';
2  import { StyleSheet, Text, View } from 'react-native';
3  import { StackNavigator } from 'react-navigation'
4  import LoginScreen from './Screens/LoginScreen'
5  import HomeScreen from './Screens/HomeScreen'
6  export default class App extends React.Component {
7    render() {
8      return (
9        <AppNavigator/>
10     );
11   }
12 }
13 const AppNavigator = StackNavigator({
14   LoginScreen: {screen: LoginScreen},
15   HomeScreen: {screen: HomeScreen}
16 })
17 const styles = StyleSheet.create({
18   container: {
19     flex: 1,
20     backgroundColor: '#fff',
21     alignItems: 'center',
22     justifyContent: 'center',

```

Figure 32: Generated Code of User Interfaces of Library Application Case Study

5.2.4. Verification

For the verification of generated JavaScript code, compilation and execution is necessary. For this purpose, we have used the Expo Local Development Tool (Desktop and Mobile Client) for the compilation and execution of code. We have created the new react native project using NPM CLI as shown in **Figure 33** and **Figure 34**. After pasting our generated code run the command *npm start* to start the react native packager. Transformation results are shown in **Figure 35**, **Figure 36** and **Figure 37**.

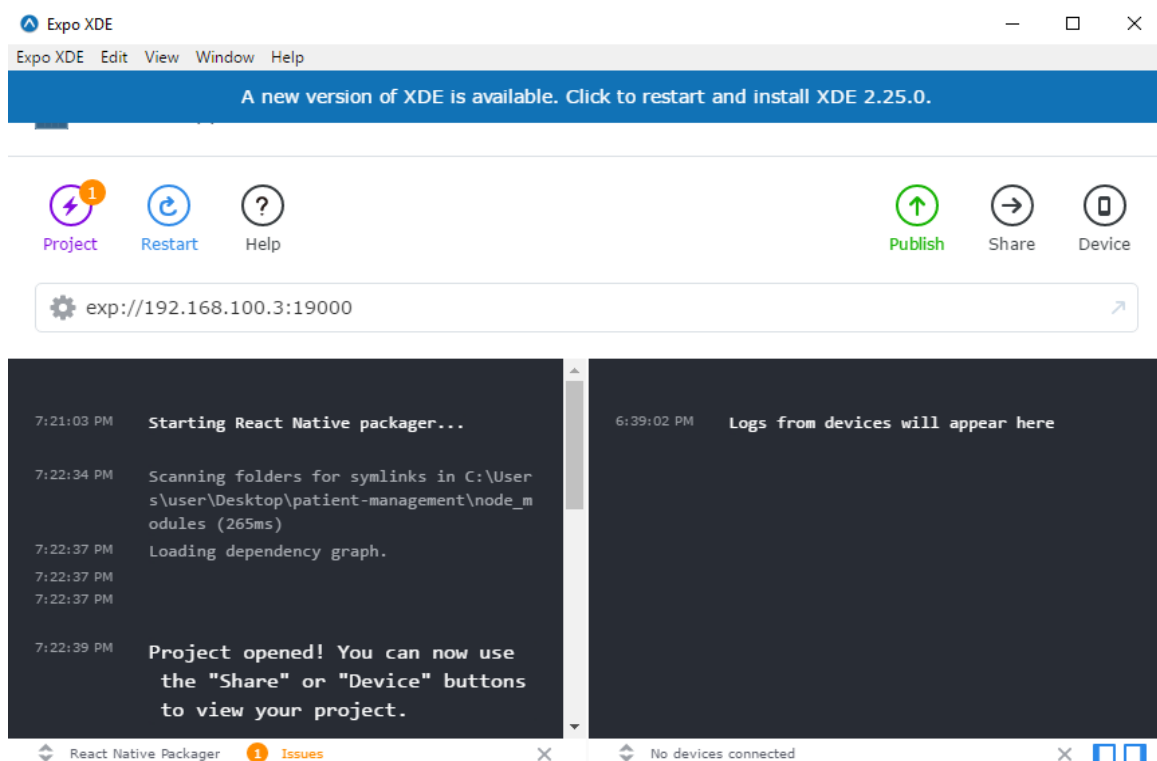


Figure 33: Expo XDE Desktop Application

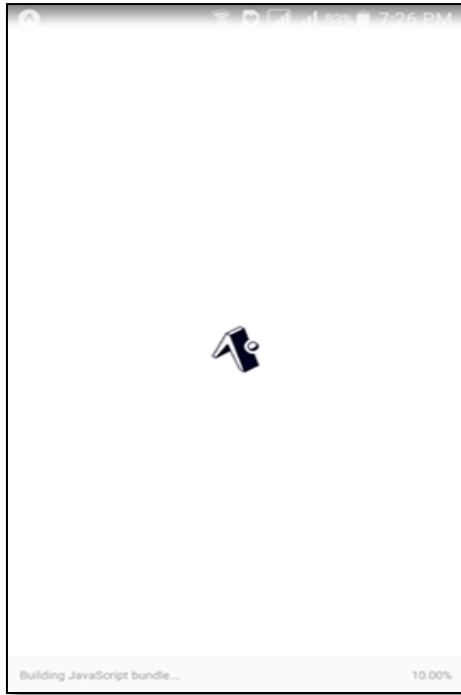


Figure 34: Expo Mobile Client

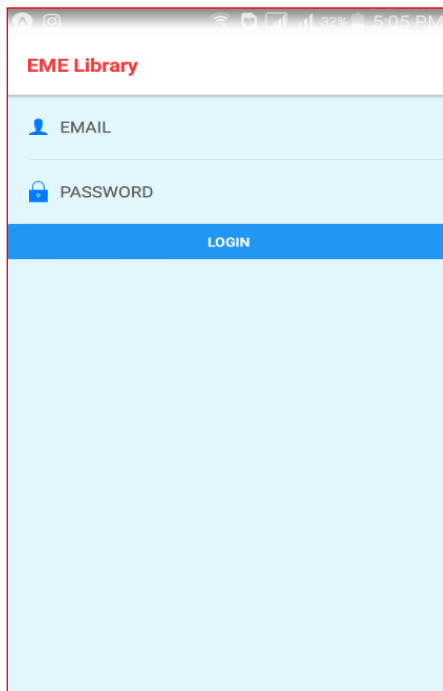


Figure 35: User Interface after Model Transformation (Login Screen)

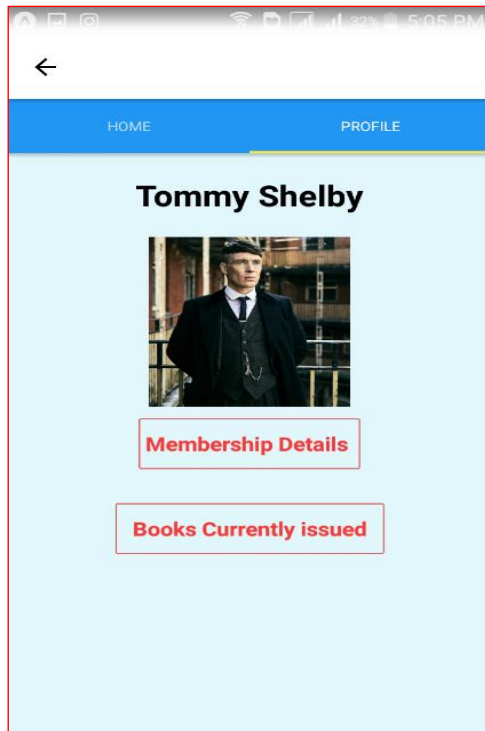


Figure 36: User Interface after Model Transformation (Profile Screen)

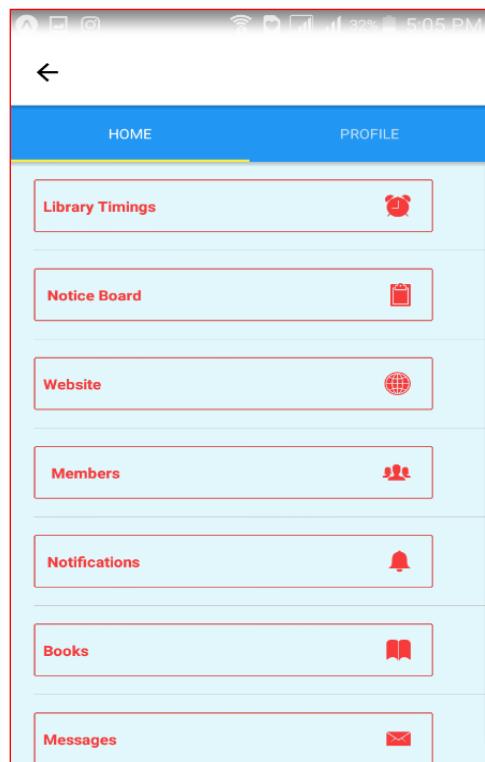


Figure 37: User Interface after Model Transformation (Home Screen)

Chapter 6

Comparative Analysis

CHAPTER 6: Comparative Analysis

The previous chapter deals with the implementation and validation aspects of the proposed work. The proposed framework presents the modeling of the concepts involved in implementation of a multi-platform user-interfaces. To model the concepts of system Unified Modeling Language (UML) profile is proposed and transformation of model into code has been carried out using transformation engine for implementation purpose. Our proposed approach provided a major contribution in the field and in this section, we have discussed the strengths, weaknesses and achievements of previous researches and their comparison with our proposed solution.

6.1. Comparison with Previous Studies

Multiple researches have been carried out to provide an optimal solution for automated multi-platform mobile application development. For comparison purpose we have selected some researches from the previous studies and defined the seven parameters as shown in **Table 2**.

Table 2: Comparison with previous studies

Sr #	Recent Approaches	Multi-Platform	UI Presentation	Applications	Complexity	Techniques	Tool Support
[15]	Juse4Android	No	Yes	Native	Low	DSML-Based	Yes
[25]	MD2	Yes	No	Native	High	DSML-Based	Yes
[14]	MAG	Yes	No	Native	Medium	UML-Based-DSML	Yes
[24]	MobML	Yes	No	Native	High	DSML-Based	Yes
[17]	MIMIC	Yes	No	Mob Web.	High	UML-Based-DSML	Yes
[5]	Applause	Yes	No	Mob Web.	High	DSML-Based	Yes
[22]	Mobl	No	Yes	Mob Web	High	DSML-Based	Yes
[33]	Francese et al.	Yes	No	Hybrid	Medium	UML-Based-DSML	Yes
[19]	TransUI	No	No	Mob Web	High	DSML-Based	No
[5]	Mendex App Platform	Yes	No	Hybrid	Medium	UML-Based-DSML	Yes

[5]	IBM Rational Rhapsody	Yes	No	Native	High	DSML-Based	Yes
[11] [6] [7]	Web Ratio Mobile Platform	Yes	No	Hybrid	High	UML-Based-DSML	Yes
	Proposed Work	Yes	Yes	Native	Low	UML-Based	Yes

We have provided a detailed discussion on aforementioned studies in literature review section but here we use these studies to compare it with our proposed solution as some of the work has been conducted over the last few years. **Table 2** summarizes the comparison of our proposed solution with the previous studies. The parameters compare the researches on basis user interfaces with styling and layout with native look and feel, application types such as native, hybrid or mob web, native apps have full access to device capabilities and have highest performance. Hybrid apps use the Web View widget that is part of both platforms and have slow performance and some access to device capabilities whereas mob web applications are slow and have no access to device capabilities. UI complexity represents the degree to which users observe the info displays of the system to be easy to use and understandable. Also, based on element interface specification, we can identify an interface complexity through its element's complexity, thereby providing an excellent sign of element reusability. So, we have considered complexity with usability and understandability aspect as high, medium and low. Techniques such as DSML based or UML based DSML and lastly, tool support provided or not. These previous studies are mostly focused to generating business logic only such as Juse4Android, MAG, MobML and MD2. In terms of user interface generation many of the above approaches uses libraries and templates and if any tool and approach was presented, it was mostly web based. For user interface reliance on existing IFML (Interaction Flow Modeling Language) none of them covers the model driven development of multi-platform user interface native components themselves. limitation in others model-driven approaches such as Mobl which is for mob web applications, it results in delaying developer productivity due to complexity of asynchronous programming style enforced by HTML5 JavaScript API's and in MD2 user interface is mainly consisting of form fields and due to use of html5, they have to face restrictions with respect to device specific features therefore, its complexity is high. And for tools like applause which is DSML- based complexity is high because it uses their own language to describe the user interface and cannot support to generate new UI for another platform from existing model, therefore it is essential to amend the whole tool, to

support multiple platforms, as it is challenging to capture the UI information of different various platforms simultaneously. We used UML which is a generic language, provides primitives of object oriented language and can cover a wide variety of system across broad range of domains while most of the above mentioned approaches comprises of DSML (Domain Specific Language) provides primitives of a specific domain, another limitation to this approach is, to ensure a good startup of the project, in order to use a solution developer, need to learn the new language. UML also includes the mechanism of profile which allows the modeler to customize and add new concept in the model. Thus, UML is more flexible than DSML. Furthermore, most of the previously proposed solutions contain high level of complexity due to usability and understandability aspect which arise the development and implementation issues and also few of them contains the partial tool support such as MD2, Mendex App Platform etc. For UI does not cover the modeling of presentation issues of a native application frontend e.g. layout, style and look & feel. While our proposed solution covers the style, layout, native look & feel covering the multiplatform as well, which is economically cheap, provide a high level of abstraction reducing the complexity and contain a tool support for M2T transformation.

Chapter 7

Discussion and Limitation

CHAPTER 7: DISCUSSION AND LIMITATION

This section presents a detailed discussion on this research work **Section 7.1** and limitations to the research are also presented in **Section 7.2**.

7.1. Discussion

From this research, it has been analyzed that there is some amount of research work done in the area of model-based UI design and the available research work did not capture complete UI related requirements. Most of the work has been done only for control behavior, content and user interaction of front-end applications and does not cover the modeling of the presentation aspects and also does not cater for the specification of native mobile applications. Our proposed approach is a step toward automated native mobile UI development covering the presenting issues using models as a primary artifact of development process. In this dissertation, UMUI has been proposed at higher level of abstraction to specify the user interface requirements and generates low-level implementation of target platform react native. The reason of this choice is that according to study by Hugo et al [27] React Native have the highest recognition. It is created by the company Facebook, its development is done in JavaScript and basing the visual components on React. This framework is considered revolutionary because of its attempt to approximate native code of each of the operating systems. In their study they evaluated different frameworks based on different criteria, which are essential in the development of mobile applications and that may have preponderance in choosing the implementation model of the business by an entity or programmer. The criteria were selected according to the needs of mobile development are as follows: quality of documentation, development cost, emulators and debugging, response speed and time, commercial acknowledgement, code reuse and teamwork, maintenance and upgrades. After an evaluation of the most recognized applications for each framework and after performing implementation tests results were presented on a Likert scale. It was then verified that React Native is the best solution to have a team focused on a single language and develop all kinds of front-end solutions. In relation to other frameworks, React Native presents the best results, due to Facebook's support in creating an efficient framework. Recently React Native has had an exponential market recognition because Facebook, Instagram, Skype, Uber, Airbnb or even Tesla, are examples of sound names developed in this framework.

The MMUI Transformation Tool we have built, produces all the real UI artifacts that a developer will need in order to develop the right product meeting the user specified UI requirements. Motivation behind this work is to provide early UI design so that quality can be build inside the application which eventually proved out to be a cost and time efficient approach. The automated UI provide complete and detailed design for example, how it will cover style, layout, look and feel of the front-end of mobile applications.

Automated MMUI (Model driven multiplatform Mobile User Interfaces) tool can generate the user interface of the application easily by any level of people like managers, product owners. So, mostly coding sessions can be avoided that way because job can be seen and examined even before it will be coded. This leads to initiate user interface development in the early requirement specification phases especially without considering the ultimate application development technologies.

Case studies containing different user interface components have been selected in order to validate our proposed approach. We selected a very detailed case studies on Patient Management System and Library application which included more than 70 components.

7.2. Limitations

As we have taken the first step to UI generation for React Native, there are a few limitations to our work. UMUI has a lot of potential but due to limited amount of time and resources, we have currently only selected limited core elements i.e. Containers, Navigation of two types i.e. tab navigation and stack navigation, Button, List and Icon etc. There are many UMUI that we have not yet considered for example RadioButton, CheckBox, and Picker etc. on which we intend to work in future.

Chapter 8

Conclusion and Future Work

CHAPTER 8: CONCLUSION AND FUTURE WORK

In this dissertation, proposed approach is focused on mobile application domain that allows us to design the user interfaces in a platform independent way. So, we have presented UMUI (Unified Modeling Language profile for multiplatform **M**obile **U**ser **I**nterfaces) for automatically generating the low-level implementation for react native framework from high-level source models. The proposed UMUI consists of several stereotypes, meta-classes and enumerations. The proposed approach makes use of MDE for user interface generation through acceleo transformations based on different rules resulting in automated code which cover the navigational aspects along-with the structural aspects of mobile application user interface by using MMUI (**M**odel driven multi-platform **M**obile **U**ser **I**nterfaces) tool. The results of our proposed approach proved the potential and viability of UMUI models for user interface generation by different case-studies of varying sizes.

We analyzed the major multi-platform mobile user interface development frameworks on the market and their software development environments. This analysis has led to highlight the recognition of react native framework. Generation of low level react native implementation from UMUI models in the early stages of development cycle will allow developers and testers to develop the right product by embedding the quality in it, from the beginning of development and by making the development process less complex. Modeling under unified modeling language notation is great benefit of our approach, by proposing a graphical way to design the multi-platform mobile user interfaces in latest trend and technology.

Future work includes improving and extending this approach in order to support other components like ScrollView, Thumbnail, FAB (Floating Action Buttons) and CheckBox etc. The applicability of the proposed automated approach on other UI frameworks such Ionic and Xamarin can be explored by the UMUI profile. The approach can be expanded to incorporate business logic requirements and interaction with the user, in order to generate fully functional mobile applications.

APPENDIX A

USER MANUAL

1. Download Instructions

1.1. Model driven multiplatform Mobile User Interface transformation tool (MMUI)

Download Model driven multiplatform Mobile User Interface transformation tool as:

“MMUI Transformation Tool.zip”

Extract the MMUI Transformation Tool.zip file. You will find the “MMUI Transformation Tool” Folder.

In the “MMUI Transformation Tool” Folder, you will find the following folder shown in the *Figure 1* below

Name	Date modified	Type
 MMUI.jar	11/21/2018 6:14 PM	Executable Jar File

Figure 1: Files in MMUI Transformation folder

1.2. Sample Case Studies

Download sample case studies (Patient Management System Application) and (Library Application) from MMUI website as “**Sample-CaseStudies.zip**”

Extract **Sample-CaseStudies.zip** file. You will find “PatientManagementCaseStudy” and “Library Application” folder as shown in *Figure 2*. Folder contains UML models for the corresponding case studies developed in Eclipse using Papyrus plugin.



Name	Date modified	Type
 LibraryApplication	11/27/2018 11:49 PM	File folder
 PatientManagementSystem	11/21/2018 6:16 PM	File folder

Figure 2: Sample Case Study folder

You can use the existing UML models to generate complete Code or you can update the UML model to include modeling of System.

2. Prerequisites for MMUI Transformation Tool

It is required to install **Java Runtime Environment (JRE) version 8 or above** in order to execute MMUI Transformation Tool.

3. Execution of MMUI Transformation Tool

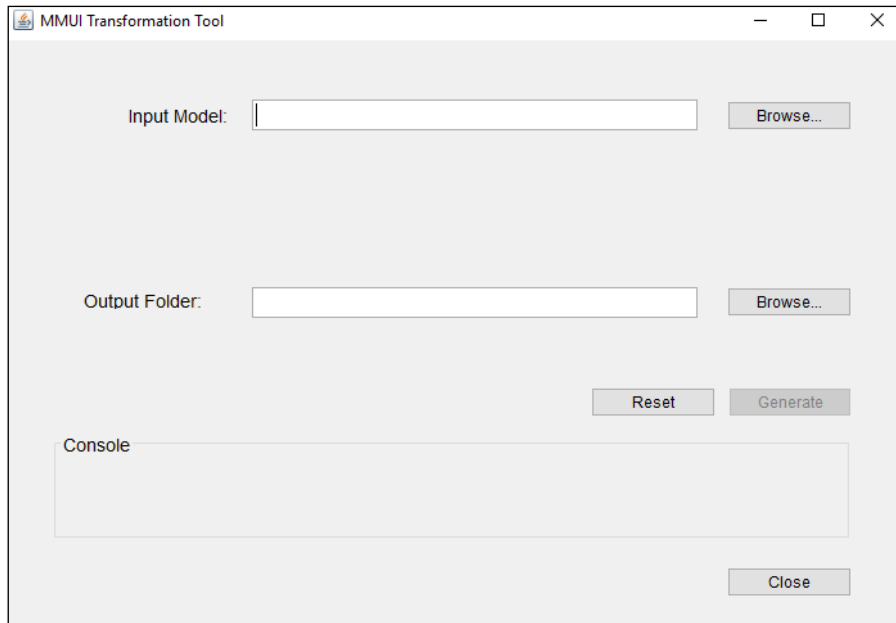


Figure 3: Interface for Model to Code Transformation Engine

Input UML Model: Button called browse is used to select the UML model for the case study.

Destination Folder: Button called browse is used to specify the target folder for the generated files.

Generate Files: User can select the required files from the given four options by checking the checkbox.

Reset: Reset Button is used to clear all the existing selections to define new configurations.

Generate: Generate button transforms the designated UML models into the required testing artifacts. It is mandatory to fill all the above field in order to click generate button.

Console: Console displays the status of current transformations which are either list of generated files or files generated with errors (in case of any problem in transformation).

Close: By using close button, you can close the interface.

The UML models can be selected using browse button against each selection. *Figure 4* shows the selection of user interface model using browse button.

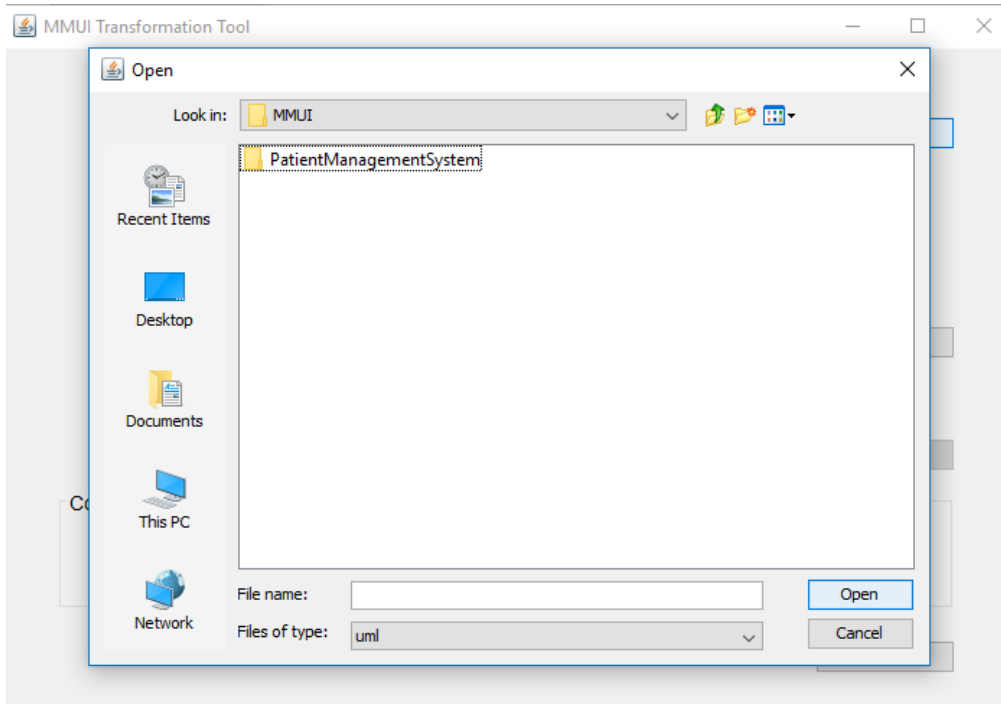


Figure 4: Selection of the UML model using Browse Button

The UML models can be transformed into JavaScript code artifacts through *Generate* Button as shown in *Figure 5*.

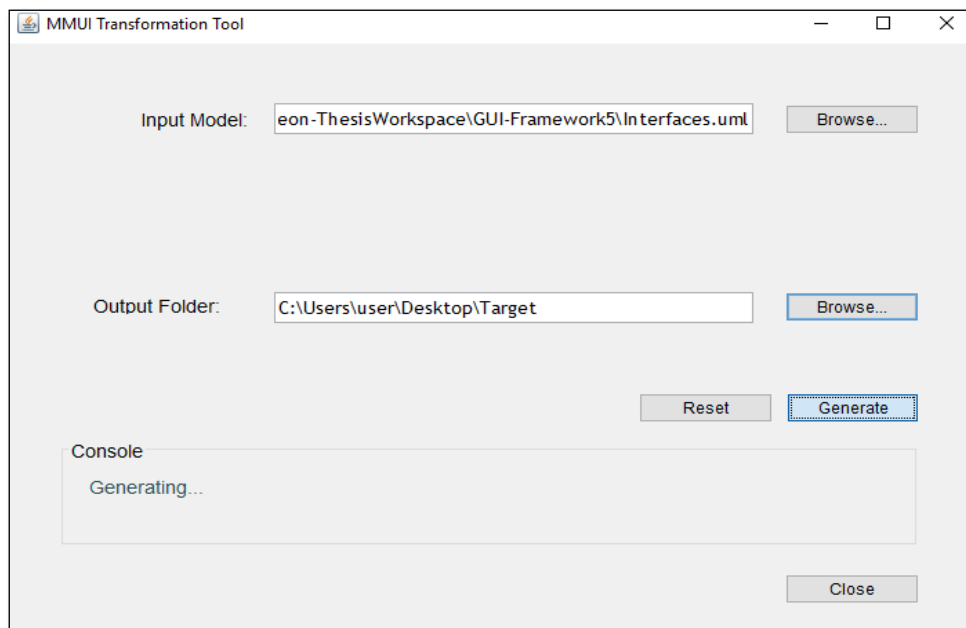


Figure 5: Generating Code Artifacts for Patient Management System Model

The screenshot for the output folder containing generated files and generate code are shown in *Figure 6* and *Figure 7*.

This PC > Desktop > Target

Name	Date modified	Type	Size
MainScreen.js	11/15/2018 4:41 PM	JavaScript File	2 KB
Screen1.js	11/15/2018 4:41 PM	JavaScript File	1 KB
Screen2.js	11/15/2018 4:41 PM	JavaScript File	1 KB
Screen3.js	11/15/2018 4:41 PM	JavaScript File	1 KB
Screen4.js	11/15/2018 4:41 PM	JavaScript File	1 KB

Figure 6: Output folder containing generated files

```

Working Files
x MainScreen.js
Target
C:
MainScreen.js
Screen1.js
Screen2.js
Screen3.js
Screen4.js

1 import React, { Component } from "react";
2 import {
3   View,
4   Text,
5   StyleSheet,
6   Platform
7 } from "react-native";
8
9 class MainScreen extends Component{
10
11   static navigationOptions = {
12     tabBarIcon :({ tintColor }) => (
13       title : "Patient Management System"
14     )
15   }
16   headerLeft: <Icon name= "ios-ribbon" style:{{ paddingLeft: 10,}}/> </Icon>
17   render()
18   return(
19     <AppTabNavigator/>
20     );
21   }
22 }
23 export default MainScreen;
24 const AppTabNavigator = TabNavigator({
25
26   Screen1 :{
27     screen :Screen1
28   },
29
30   Screen2 :{
31     screen :Screen2
32   },
33

```

Figure 7: Generated Code

REFERENCES

- [1] Christoph Rieger, Herbert Kuchen, “A process-oriented modeling approach for graphical development of mobile business apps”, In *Computer Languages, Systems & Structures* Volume 53, September 2018, Pages 43-58
- [2] Maryam Hamdani, Wasi Haider Butt, Muhammad Waseem Anwar and Farooque Azam “A Systematic Literature Review on Interaction Flow Modeling Language (IFML)” *Proceeding CMSS 2018 Proceedings of the 2018 2nd International Conference on Management Engineering, Software Engineering and Service Sciences*, Pages 134-138
- [3] Marco Brambilla, Eric Umuhoza¹ and Roberto Acerbis, “Model-driven development of user interfaces for IoT systems via domain-specific components and patterns” *Journal of Internet Services and Applications* (2017) 8:14
- [4] Claudia Raibulet, Francesca Arcelli Fontana, And Marco Zanoni, “Model-Driven Reverse Engineering Approaches: A Systematic Literature Review”, 2017.
- [5] Eric Umuhoza and Marco Brambilla, “Model Driven Development Approaches for Mobile Applications: A Survey”, In *13th International Conference, MobiWIS 2016, Vienna, Austria, August 22-24, 2016, Proceedings*
- [6] Matthias Stürner and Philipp Brune, “Virtual Worlds on Demand? Model-Driven Development of JavaScript-based Virtual World UI Components for Mobile Apps”, 2016.
- [7] Brambilla, M. and Fraternali, “Large-scale model driven engineering of web user interaction: The webml and webratio experience.”, 2014. *Science of Computer Programming*, 89:71–87.
- [8] Xiaoping Jia and Christopher Jones. AXIOM: A model-driven approach to cross platform application development. In *ICSOFTE 2012*
- [9] A. Delgado, A. Estepa, J.A. Troyano, R. Estepa, “Reusing UI elements with Model-Based User Interface Development” in *2015 Journal of Human Computer Studies*
- [10] Hanane BENOUDA, Redouane ESSBAI, Mostafa AZIZI and Mimoun MOUSSAOUI, “Modeling and Code Generation of Android Applications Using Acceleo”, *International Journal of Software Engineering and Its Applications* Vol. 10, No. 3 (2016), pp. 83-94
- [11] Roberto Acerbis, Aldo Bongio, Stefano Butti, Marco Brambilla, “Model-Driven Development of Cross-platform Mobile Applications with WebRatio and IFML” *2015 2nd ACM International Conference on Mobile Software Engineering and Systems*

- [12] Wafaa S. El-Kassas, Bassem A. Abdullah, Ahmed H. Yousef, Ayman M. Wahba, “Taxonomy of Cross-Platform Mobile Applications Development Approaches” 2017.
- [13] Marco Brambilla, Roberto Acerbis, Aldo Bongio, Stefano Butti, “Model-Driven Development of Cross-Platform Mobile Applications with Web Ratio and IFML” in 2015 2nd ACM International Conference on Mobile Software Engineering and Systems
- [14] Muhammad Usman, Muhammad Zohaib Iqbal, Muhammad Uzair Khan, “A Model-driven Approach to Generate Mobile Applications for Multiple Platforms” in 2014 21st Asia-Pacific Software Engineering Conference
- [15] Luís Pires da Silva and Fernando Brito e Abreu, “Model-Driven GUI Generation and Navigation for Android BIS Apps” in 2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)
- [16] Mohamed LACHGAR, Abdelmounaïm ABDALI, “Generating Android Graphical User Interfaces using an MDA Approach” in 2014 Third IEEE International Colloquium in Information Science and Technology (CIST)
- [17] Nadia Elula, Xavier Le Pallec, José Rouillard, Jean-Claude Tarby “A Model-based Approach for Engineering Multimodal Mobile Interactions” in MoMM '14 Proceedings of the 12th International Conference on Advances in Mobile Computing and Multimedia
- [18] Pierre A. Akiki, Arosha K. Bandara, and Yijun Yu, “Adaptive Model-Driven User Interface Development Systems” in ACM Computing Surveys (CSUR) Surveys Homepage archive Volume 47 Issue 1, July 2014 Article No. 9
- [19] Zhiyi Ma, Wei Zhang, Chih-Yi Yeh, “Model-Driven Development of Diverse User Interfaces” in ICSE Companion 2014 Companion Proceedings of the 36th International Conference on Software Engineering Pages 616-617
- [20] Andreas Schuler, Barbara Franz, “Rule-Based Generation of Mobile User Interfaces” in 2013 10th International Conference on Information Technology: New Generations
- [21] Ayoub SABRAOUI, Mohammed EL KOUTBI, Ismail KHRISS “GUI Code Generation for Android Applications Using a MDA Approach” in 2012 IEEE International Conference on Complex Systems (ICCS).
- [22] Zef Hemel and Eelco Visser, “Declaratively Programming the Mobile Web with Mobl” in OOPSLA '11 Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications Pages 695-712
- [23] Eric Umuhoza. Hamza Ed-douibi, Marco Brambilla, Jordi Cabot, Aldo Bongio “Automatic Code Generation for Cross-Platform, Multi-device Mobile Apps: Some

- Reflections from an Industrial Experience” in · Proceeding MobileDeLi 2015 Proceedings of the 3rd International Workshop on Mobile Development Lifecycle Pages 37-44
- [24] Marco Brambilla, Andrea Mauri, Mirco Franzag, Henry Muccini, “A Model-Based Method for Seamless Web and Mobile Experience” in Mobile! 2016 Proceedings of the 1st International Workshop on Mobile Development Pages 33-40
- [25] Henning Heitkötter, Tim A. Majchrzak, Herbert Kuchen, “Cross-Platform Model-Driven Development of Mobile Applications with MD2” in SAC '13 Proceedings of the 28th Annual ACM Symposium on Applied Computing Pages 526-533
- [26] Joachim Perchata,b, Mikael Desertotb, Sylvain Lecomteb, “Component Based Framework to Create Mobile Cross-platform Applications” in The 3rd International Symposium on Frontiers in Ambient and Mobile Systems (FAMS)
- [27] Hugo Brito, Anabela Gomes, Álvaro Santos and Jorge Bernardino, “JavaScript in mobile applications: React Native vs Ionic vs NativeScript vs native development” in 2018 13th Iberian Conference on Information Systems and Technologies (CISTI)
- [28] Chi-Kien Diep, Quynh-Nhu Tran, Minh-Triet Tran, “Online model-driven IDE to design GUIs for cross-platform mobile applications” in SoICT '13 Proceedings of the Fourth Symposium on Information and Communication Technology Pages 294-300
- [29] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Bryani Dzung Ta, Atif M. Memon, “MobiGUITAR: Automated Model-Based Testing of Mobile Apps” IEEE Software (Volume: 32, Issue: 5, Sept.-Oct. 2015)
- [30] HenningHeitkötter, HerbertKuchen, TimA.Majchrzak, “Extending a model-driven cross-platform development approach for business apps” in Science of Computer Programming
Volume 97, Part 1, 1 January 2015, Pages 31-36
- [31] Mohamed Lachgar and Abdelmounaïm Abdali, “Modeling and generating native code for cross-platform mobile applications using DSL” in Intelligent Automation & Soft Computing, 2016
- [32] Roman Popp, Jurgen Falb, David Raneburger and Hermann Kaindl, “A Transformation Engine for Model-driven UI Generation” in TAMODIA'07 Proceedings of the 6th international conference on Task models and diagrams for user interface design Pages 112-125
- [33] Rita Francese, Michele Risi(B), Giuseppe Scanniello, and Genoveffa Tortora, “Model-Driven Development for Multi-Platform Mobile Applications” in MOBILESoft

'15 Proceedings of the Second ACM International Conference on Mobile Software Engineering and Systems

- [34] Sarra Roubi, Mohammed Erramdani, Samir Mbarki, “Modeling and Generating Graphical User Interface for MVC Rich Internet Application using a Model Driven Approach” in 2016 International Conference on Information Technology for Organizations Development (IT4OD)
- [35] Juliano de Almeida Monte-Mor, Elton Oliveira Ferreira, Henrique Fernandes Campos, Adilson Marques da Cunha, and Luiz Alberto Vieira Dias, “Applying MDA Approach to Create Graphical User Interfaces” in 2011 Eighth International Conference on Information Technology: New Generations
- [36] Javier Rodríguez Escolar, “A Context-aware Dialog Model for Multi-Device Web Apps” EICS '13 Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems Pages 167-170
- [37] Ei Ei Thu, Nwe New, “Model Driven Development of Mobile Applications Using Drools Knowledge-based Rule” 2017 IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA)
- [38] G. Botturi, E. Ebeid, F. Fummi, D. Quaglia, “Model-Driven Design for the Development of Multi-Platform Smartphone Applications” in Proceedings of the 2013 Forum on specification and Design Languages (FDL)
- [39] André Ribeiro, Alberto Rodrigues da Silva , “Survey on Cross-Platforms and Languages for Mobile Apps” in Proceeding SAC '14 Proceedings of the 29th Annual ACM Symposium on Applied Computing Pages 1316-1323
- [40] Muhammad, R., Muhammad, W.A., Aamir, M.K., 2015, “Towards the Tools Selection in Model Based System Engineering for Embedded Systems - A Systematic Literature Review”, Elsevier Journal of Systems and Software, Vol. 106
- [41] Object Management Group(OMG) [Online] Available: <https://www.omg.org/spec/OCL/2.4/> [Accessed: 26-Oct-2018].
- [42] Papyrus User Guide [Online Available] : https://wiki.eclipse.org/Papyrus_User_Guide [Accessed: 26-Oct-2018].
- [43] Acceleo [Online Available]: <https://www.eclipse.org/acceleo/documentation/> [Accessed: 26-Oct-2018].
- [44] Eclipse [Online Available]: <https://wiki.eclipse.org/> [Accessed: 26-Oct-2018].

- [45] UML Diagrams [Online Available]: <https://www.uml-diagrams.org/> [Accessed: 26-Oct-2018].
- [46] ECORE TOOLS [Online Available]: <https://www.eclipse.org/ecoretools/doc/index.html> [Accessed: 26-Oct-2018].
- [47] React Native [Online Available]: <https://facebook.github.io/react-native/> [Accessed: 26-Oct-2018].
- [48] IONIC DOCUMENTATION [Online Available]: <https://ionicframework.com/docs/> [Accessed: 26-Oct-2018].
- [49] EXPO [Online Available]: <https://expo.io/> [Accessed: 26-Oct-2018].
- [50] XAMARIN [Online Available]: <https://visualstudio.microsoft.com/xamarin/> [Accessed: 26-Oct-2018].
-