A model-based framework to bridge architecting, engineering, and testing

Citation for published version (APA):

Document status and date:
Published: 28/09/2017

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 12. Apr. 2020
A Model-based Framework to Bridge Architecting, Engineering, and Testing

Yinghui Wu
A Model-based Framework to Bridge Architecting, Engineering, and Testing

Yinghui Wu

Eindhoven University of Technology
Stan Ackermans Institute - Automotive/Software Technology

PDEng Report: 2017/054

Partners

Océ Technologies
Eindhoven University of Technology

Steering Group
Project Owner: Ronald Fabel (Océ Technologies)
Project Supervisor: Roelof Hamberg (Océ Technologies)
Hristina Moneva (Océ Technologies)
Project Supervisor: Harold Weffers (TU/e)

Date 28 September 2017
Document Status Public
PDEng report no.

The design described in this report has been carried out in accordance with TU/e Code of Scientific Conduct.
The purpose of this project is to construct a model-based framework to bridge architecting, engineering, and testing. The framework supports software development teams in analyzing the architecture of software (Architecture Analysis) and the impacts of changes (Impact Analysis) during software’s development and maintenance.

The software, which the project focused on, is one of the controllers composing the printing products. The controller is complex due to its development history, context, and techniques. Its current characteristics have the possibility of causing costly errors and delays in the controller’s evolution. One solution to prevent such risks is formulating an overview of the controller’s architecture.

The framework realizes the solution by modeling the controller’s structure and interface definitions. Therefore, the controller development team is able to interpret the controller’s architecture comprehensively, discover the controller’s deficiencies promptly, investigate influences of changes efficiently, and document design decisions regularly.

Keywords
- Architecture Analysis
- Impact Analysis
- Software Development
- Model-based

Preferred reference

Partnership
This project was supported by Eindhoven University of Technology and Océ Technologies.

Disclaimer
Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Eindhoven University of Technology or Océ Technologies. The views and opinions of authors expressed herein do not necessarily state or reflect those of Eindhoven University of Technology or Océ Technologies, and shall not be used for advertising or product endorsement purposes.

Disclaimer Liablity
While every effort will be made to ensure that the information contained within this report is accurate and up to date, Eindhoven University of Technology makes no warranty, representation or undertaking whether expressed or implied, nor does it assume any legal liability, whether direct or indirect, or responsibility for the accuracy, completeness, or usefulness of any information.

Trademarks
Product and company names mentioned herein may be trademarks and/or service marks of their respective owners. We use these names without any particular endorsement or with the intent to infringe the copyright of the respective owners.

Copyright
Copyright © 2017. Eindhoven University of Technology. All rights reserved. No part of the material protected by this copyright notice may be reproduced, modified, or redistributed in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of Eindhoven University of Technology and Océ Technologies.
Foreword

Print systems and their controllers are complex systems that pose challenges in managing their growing complexity, while ensuring high quality and delivering on time, as it is with all high-tech embedded systems. Considering that Océ’s controller platform has been developed in the course of more than a decade, by hundreds of developers, in different locations of Europe, and with various technologies, it can be expected that maintaining the overview is a challenge. Moreover, an impact analysis for each new feature proves to be an expert’s task. In order to provide support to all parties involved in the development process, one can envision that the controller can be ultimately fully modeled - data, behavior, interface, but also the controller architecture - and that these models should be the basis of further specification, analysis, and simulation. This vision includes that there will be a bridge not only towards the realization, but also to the higher-level system behavior in context for specification and validation purposes. Luckily a lot of models already exist and can be used as stepping stones towards this vision.

Yinghui’s project "A Model-based Framework to Bridge Architecting, Engineering, and Testing" delivers the first step in our roadmap. It will serve as a base for integrating existing and new models. We believe that connecting models (not just developing them in insulation) delivers exponential added value to the organization. In order to show the feasibility of the approach, we selected our interface definition modeling as one of the oldest and most established model-driven practices in the controller development. We further extended it with "requirements on interface level" (e.g. data constraints and usage scenarios specifying the expected behavior in different situations). Combining these models and putting them in their design context in the controller architecture model demonstrates the initial added value and sketches the potential of further extensions, but more importantly it opens discussion on how far we want and can go in order to achieve our vision.

Yinghui has successfully achieved the goal to provide a proof-of-concept and based on that we already know that this was a valuable step in the right direction. Her hard work in learning new domain and technologies as well as in balancing own work and student supervision has concluded in convincing and complete results. We are happy to adopt these results and continue maturing and developing them further.

ir. Hristina Moneva PDEng

August 2017
Preface

The project, “A Model-based Framework to Bridge Architecting, Engineering, and Testing”, was completed by Yinghui Wu at Océ Technologies. Moreover, this project was a nine-month graduation project of the Software Technology Professional Doctorate in Engineering (PDEng) program of Eindhoven University of Technology.

The PDEng degree program in Software Technology is provided by the Department of Mathematics and Computer Science of Eindhoven University of Technology in the context of the 4TU.School for Technological Design, Stan Ackermans Institute. The program focuses on strengthening trainees technical and non-technical competencies related to the effective and efficient design and development of software for resource-constrained software-intensive systems.

This report summarizes the development process of the framework in addition to the management process of the project. The target audience can be both technical and non-technical readers. However, primarily it is intended for the controller development team of the company, who is encountering the problems that the project solves. On the one hand, based on various concerns, readers may focus on different chapters as illustrated in the table below. On the other hand, readers, who are interested in the entire project, are welcome to read the entire report.

<table>
<thead>
<tr>
<th>Concerns</th>
<th>Chapters</th>
</tr>
</thead>
<tbody>
<tr>
<td>the motivation and challenge of the project</td>
<td>Chapter 1 Introduction</td>
</tr>
<tr>
<td></td>
<td>Chapter 3 Problem Analysis</td>
</tr>
<tr>
<td>the domains that the project resides</td>
<td>Chapter 4 Domain Analysis</td>
</tr>
<tr>
<td>the technical details of the framework</td>
<td>Chapter 5 Requirement Analysis</td>
</tr>
<tr>
<td></td>
<td>Chapter 6 Architecture,</td>
</tr>
<tr>
<td></td>
<td>Chapter 7 Design,</td>
</tr>
<tr>
<td></td>
<td>Chapter 8 Implementation,</td>
</tr>
<tr>
<td></td>
<td>Chapter 9 Verification and Validation</td>
</tr>
<tr>
<td>the usage of the framework</td>
<td>Chapter 5 Requirement Analysis</td>
</tr>
<tr>
<td></td>
<td>Chapter 10 Results and Recommendations</td>
</tr>
<tr>
<td>the management of the project</td>
<td>Chapter 2 Stakeholder Analysis</td>
</tr>
<tr>
<td></td>
<td>Chapter 11 Project Management</td>
</tr>
<tr>
<td></td>
<td>Chapter 12 Reflection</td>
</tr>
</tbody>
</table>

Yinghui Wu
September 2017
Acknowledgement

The project was accomplished with the support and encouragement of experts and colleagues in both Océ Technologies and Eindhoven University of Technology (TU/e).

First, I owe special gratitude to my company supervisors, Roelof Hamberg and Hristina Moneva, for providing me the opportunity of conducting the graduation project. More importantly, their creative ideas and extensive knowledge inspired me to broaden my project from various perspectives. I am also grateful to my university supervisor, Harold Weffers, for instructing me through the project. Moreover, his profound experience enriched my skills especially in the aspect of project management. The insightful observations and valuable guidance from all the supervisors helped me achieve both professional and personal development.

In addition, a number of colleagues in the company contributed domain expertise related to the controller’s development and maintenance as well as provided feedback on the project’s output: Niels aan de Brugh (Controller Architect), Jorryt Lenoir (Feature Owner), Rob Jacobs (Function Designer), Frans Verhaag (Function Engineer), Johan Pijnappels (Function Engineer), Ton Gerrits (Workflow Architect), and Johan Foederer (Product Tester). Besides, Roel van Bakel, an expert of the company-specific interface definition system, supported me in revising the interface definition domain specific language; Eugen Schindler, a JetBrains MPS expert in the company, assisted me in resolving technical problems of utilizing the tool; and Serguei Roubtsov, Researcher/Consultant, Eindhoven University of Technology, provided technical articles related to architecture modeling. I appreciate their willingness to participate in the project and to share their knowledge with me.

Furthermore, my great appreciation goes to everyone in the PDEng program: Ad Aerts, Yanja Dajsuren, and Desiree van Oorschot for their assistance and supervision as well as the coaches for their contribution of knowledge and experience during these past two years. Additionally, I would like to thank my colleagues in the PDEng program for the great moments and experiences we shared together.

In the end, I want to express my acknowledgments to my family and friends. My parents and parents-in-law, came from China to take care of me when I was suffering from pregnancy disorders. My husband, Yongchao Wang, supported me with unconditional love and patience. My friends, Jie Wang and Liwei Ma, provided unflagging comfort and encouragement.
Executive Summary

The purpose of this project is to construct a model-based framework that can bridge the architecting, engineering, and testing of software development. The framework supports software development teams in analyzing the architecture of software (Architecture Analysis) and the impacts of changes on software (Impact Analysis).

The software, which the project focused on, is one of the controllers composing the printing products. The controller is complex due to its development history, context, and techniques. Its current characteristics have the possibility of causing costly errors and delays in the controller’s evolution. One solution to prevent such risks is to formulate an overview of the controller’s architecture, so that the controller development team is able to

- Interpret the architecture comprehensively.
- Discover the controller’s deficiencies promptly.
- Investigate influences of changes efficiently.
- Document design decisions regularly.

The framework realizes the solution by modeling the controller’s architecture from two aspects: the controller’s structure and interface definitions. Accordingly, the project is divided into two phases

1. Controller Structure Modeling
   The applications created in this phase first extract the information that is essential for Architecture Analysis and Impact Analysis from the controller’s code base. Then, on the grounds of the information, the controller’s structure is illustrated in four abstraction levels according to the C4 Model (Context, Containers, Components, and Classes). Moreover, these models are presented as structured documents in a certain way. Unlike the existing references, the models and the documents are generated automatically. More importantly, they reflect the instance status of the controller’s architecture by virtue of the timeliness and correctness of the extracted information.

2. Interface Modeling
   The program completed in this phase enables representing and editing the interface definitions both textually and graphically. Compared with the interface definitions’ original format, the textual presentation displays the content concisely and the graphical visualization demonstrates the relationship among the elements defined in an interface definition. Furthermore, a modified interface definition is generated to its original format, which guarantees the interface definition to be compatible with the company’s other software.

The framework was accomplished within nine months with general techniques (Java, Unified Modeling Language, and Systems Modeling Language) in addition to a new concept (C4 Model) and state-of-the-art technologies (Interface Definition, Domain Special Language, and Meta-programming). Additionally, the Probability and Impact Matrix and Milestone Trend Analysis methods were employed to ensure that the project could be finished on time and within budget.

As a result, the framework manages to establish an overview of the controller’s architecture that benefits the controller development team in conducting Architecture Analysis and Impact Analysis. One output of the project is a log that records probable design deficiencies. I recommend examining the log in the
interest of identifying possible risks in advance. In addition, considering further development of the framework, I recommend referring to the framework’s development documents, which contain the complete list of requirements as well as architecture and design specifications. Moreover, besides the two aspects that were modeled by the framework, the third object to model is usage scenarios. Related tasks were achieved by a group of undergraduate students separately at Eindhoven University of Technology under my supervision. I recommend integrating their deliverables to the framework in favor of supporting the controller development team from an extra aspect.
Contents

Foreword.......................................................................................................................... III
Preface ............................................................................................................................... V
Acknowledgement .............................................................................................................. VII
Executive Summary ........................................................................................................... IX

Chapter 1  Introduction ...................................................................................................... 1
  1.1  Context and Scope ..................................................................................................... 1
  1.2  Preliminary Results .................................................................................................. 1
  1.3  Report Outline......................................................................................................... 2

Chapter 2  Stakeholder Analysis ....................................................................................... 4

Chapter 3  Problem Analysis ............................................................................................. 5
  3.1  Research on the Status Quo...................................................................................... 5
  3.2  Problem Definition.................................................................................................... 5
    3.2.1.  Problems .......................................................................................................... 5
    3.2.2.  Project Goal...................................................................................................... 6
    3.2.3.  Project Scope ................................................................................................... 6
  3.3  Challenges............................................................................................................... 7
    3.3.1.  Compatibility with Current Way of Working .................................................... 7
    3.3.2.  Technology Acceptance by End Users ............................................................. 7

Chapter 4  Domain Analysis ............................................................................................. 8
  4.1  Modeling Object....................................................................................................... 8
    4.1.1.  C4 Model......................................................................................................... 8
    4.1.2.  Interface Definition.......................................................................................... 8
    4.1.3.  Behavior-Driven Development....................................................................... 9
  4.2  Modeling Technology.............................................................................................. 9
  4.3  Controller Code Base .............................................................................................. 10
    4.3.1.  Source ............................................................................................................. 11
    4.3.2.  Build ............................................................................................................... 12

Chapter 5  Requirement Analysis ................................................................................... 13
  5.1  Primary Use Cases.................................................................................................. 13
  5.2  Functional Requirement ......................................................................................... 14
List of Figures

Figure 1 One of Océ’s Printing Products .......................................................... 1
Figure 2 Report Outline ...................................................................................... 3
Figure 3 Reasoning of Project Scope ................................................................. 7
Figure 4 JetBrains MPS Plugins ................................................................... 10
Figure 5 Controller Code Base ....................................................................... 11
Figure 6 Primary Use Case .............................................................................. 14
Figure 7 Architecture of the Framework ......................................................... 19
Figure 8 Application of the Framework ............................................................ 20
Figure 9 Application of Controller Structure Modeling .................................. 20
Figure 10 Application of Interface Modeling .................................................. 21
Figure 11 Application of Usage Scenario Modeling (TBC) ............................ 21
Figure 12 Design of Structure Info Extractor ............................................... 25
Figure 13 Project Structure of Structure Visualizer ....................................... 27
Figure 14 Decomposition of Structure Visualizer ........................................... 27
Figure 15 Design of Structure Visualizer ...................................................... 27
Figure 16 Design of Controller Structure Modeling Solution ....................... 28
Figure 17 Design of Controller Structure Modeling Language Structure ....... 29
Figure 18 Project Structure of Interface Modeling ......................................... 31
Figure 19 Decomposition of Interface Modeling ............................................ 31
Figure 20 Design of Interface Modeling ....................................................... 31
Figure 21 Extra Concepts for Visualization .................................................. 32
Figure 22 Enabling Text in RTF ..................................................................... 33
Figure 23 Structure Info Extractor Implementation Workflow ....................... 34
Figure 24 Prototype of Extracted Information in XML Format ..................... 35
Figure 25 Structure Visualizer Implementation Workflow ............................. 36
Figure 26 The Controller Model ................................................................. 37
Figure 27 A Subsystem Model ...................................................................... 37
Figure 28 The Controller Model with Relationship Displayed ..................... 38
Figure 29 A Build Interface Definition Model ............................................. 38
List of Tables

Table 1 Role Description and Contribution of End User ................................................................. 4
Table 2 Summary of Controller Structure Modeling Requirements ...................................................... 15
Table 3 Requirements: Extract Controller Structure Information ...................................................... 15
Table 4 Requirements: Visualize Controller Structure in Different Abstraction Levels ....................... 16
Table 5 Requirements: Analyze Extracted Information via Visualization .............................................. 16
Table 6 Requirements: Generate Document .......................................................................................... 16
Table 7 Summary of Interface Modeling Requirements ............................................................................ 16
Table 8 Requirements: Interface Modeling ........................................................................................... 17
Table 9 Summary of Usage Scenario Modeling Requirements ............................................................... 17
Table 10 Summary of Quality Requirements ......................................................................................... 17
Chapter 1  Introduction

1.1  Context and Scope

Océ Technologies concentrates on digital imaging, industrial printing, and collaborative business services. Since 2009, Canon and Océ have joined forces to create high-advanced printing products. As a result, customers can choose from one of the industry’s broadest range of products backed by best-in-class service and support organizations across the world. [1]

Figure 1 illustrates one printer product that Océ produces. The construction of this kind of products concerns multiple disciplines including software engineering, computer science, mechanical engineering, electrical engineering, chemical engineering, as well as user experience and user interface design. Correspondingly, the print-related software is built with the contributions from these fields. In addition, such a type of software is generally developed with high-technologies, by large groups, at different locations, and for decades of years. Moreover, the software normally contains millions of lines of code in order to satisfy functional and non-functional requirements.

According to software engineering literature, these factors have the possibilities to introduce challenges during the software’s development and maintenance. Not meeting these challenges may cause costly errors and delays, which consequently reduces the software’s development efficiency and degrades its quality. Therefore, the company launched several programs focusing on devising solutions to prevent the aforementioned risks.

This project belongs to one of these programs that targets bridging the architecting, engineering, and testing of the software’s development. As the first stage of the program, this project is to solve the difficulties in keeping the software’s architecture overview up-to-date, relating the architecture overview to the implementation, predicting the effect of changes at either the architecture overview level or the implementation level.

1.2  Preliminary Results

This project took Print System Controller (hereinafter referred to as “the controller”) as an example. The controller, provides an intuitive workflow for immediate productivity, minimal learning curves and easy
management of complex jobs. In principle, the controller contributes to lower training and running costs, higher productivity, fewer mistakes, and bigger profits.

This project provides a way of working that complements the current methods of developing and maintaining the controller. The proposed framework facilitates the analysis of software’s architecture and changes’ impact by modeling the software’s as-is architecture from different perspectives and with update-to-date information that is extracted from the code base. These models enable the controller’s development team to evaluate the controller’s architecture conveniently and properly as well as analyze changes’ impact on the controller timely and correctly. Moreover, the framework supports in verifying the controller’s functionalities and validating the controller’s design by defining usage scenarios. As a result, the productivity of the controller’s development team and the quality of the controller are together enhanced; additionally, the cost of both development and maintenance is reduced.

Moreover, the framework is applicable to other software if a piece of software is developed by the techniques that are comparable to the controller. Given a printing product, it requires various software and each piece of software is constructed in accordance with certain conventions which the controller follows. Since this framework can be adapted, it eventually ensures the print related software, with high-quality, to be built efficiently. Considering other domains, the methods of building the controller are also observable in the software of many domains. By virtue of the framework’s genericity, the improvement of development is also achievable by the controller’s similar software in those domains.

1.3 Report Outline

As illustrated in Figure 2, this report consists of five sections: Introduction, Conclusion, Analysis, Model, and Management. These sections are comprised of twelve chapters in total.

Introduction, Chapter 1, presents the project’s objective, which is to solve the problems that are related to the controller’s development process; additionally, summarizes the framework’s significance, which is to support the controller development team in architecture analysis and impact analysis; in the end, outlines the report’s structure.

Analysis contains four chapters. Chapter 2, Stakeholder Analysis, summarizes the main stakeholders’ responsibilities and influences during different phases of the project. Chapter 3, Problem Analysis, first describes the complexity of the controller domain and the controller development team’s development methodology and process; then elaborates the difficulties, which is caused by the team’s current way of working. Based on the discovered problems, the project’s goal and scope are defined. In the end, challenges and design opportunities are also discussed. Chapter 4, Domain Analysis, defines the objects that the framework models, describes the technology that is used for modeling, and demonstrates the decomposition of the controller’s code base. Chapter 5, Requirement Analysis, lists potential end users’ primary use cases of the framework as well as analyzes their functional and non-functional (quality) expectations.
Model focuses on the process of achieving the project’s goal and realizing the requirements elicited in analysis phase. In the Architecture chapter, Architecture Reasoning Model is adopted to formulate the solution’s architecture from three dimensions including Application, Construction, and Realization. The Design chapter illustrates applications’ interactions in addition to each application’s decomposition and internal dependencies. The Implementation chapter describes the workflow of realizing the applications with pre-defined construction and realization techniques; in addition, prototypes, in screenshot, are demonstrated. The Verification & Validation chapter summarizes bugs and deficiencies, which were found during the process; moreover, explains their causes and fixes.

Findings and Recommendations, Chapter 10, summarizes achievements that the project gains and future work that is suggested to be conducted.

The Management section is composed of two chapters. Chapter 11 addresses project management’s techniques and project execution’s dynamics. In addition, in Chapter 12, the author recapitulates the project process by retrospective analysis.
Chapter 2  Stakeholder Analysis

Stakeholder Analysis, described in this chapter, aims at the main stakeholders whose requirements were taken into consideration and who contributed domain knowledge in different phases of this project.

First, since the framework is to support the controller development team, members of the team were recognized as stakeholders as well as potential end users. These members act as various roles with assorted responsibilities for the controller’s development and maintenance. Given this project, six roles were regarded as being within the scope. In summary, their contributions were

1. Specifying use cases and requirements as well as evaluating specific deliveries.
2. Providing relevant domain knowledge, techniques, references, and contacts.

<table>
<thead>
<tr>
<th>End User</th>
<th>Role Description</th>
<th>Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controller Architect</td>
<td>The architect for the controller’s development</td>
<td>1 and 2</td>
</tr>
<tr>
<td>Feature Owner</td>
<td>The technical leader for a feature’s development</td>
<td>1 and 2</td>
</tr>
<tr>
<td>Function Designer</td>
<td>The designer for a feature’s development</td>
<td>1 and 2</td>
</tr>
<tr>
<td>Function Engineer</td>
<td>The engineer for a feature’s development</td>
<td>1 and 2</td>
</tr>
<tr>
<td>Product Tester</td>
<td>The tester for a product’s testing</td>
<td>2</td>
</tr>
<tr>
<td>Workflow Architect</td>
<td>The architect for the controller’s workflow</td>
<td>2</td>
</tr>
</tbody>
</table>

Two more stakeholders, also being known as supervisors, played an important part in conceiving the framework. They monitored the quality of the deliveries and the progress of the project as well as exert influence on making decisions together with the author. One of these supervisors belongs to the controller development team. Therefore, she can also give instructions from an end-user’s point of view. The other supervisor, who works in another team, is not only a domain expert but also experienced in analyzing the controller. Hence, he participated in the project by providing domain knowledge and background information of the controller.
Chapter 3  Problem Analysis

As introduced in Context and Scope, this project is the first stage of constructing a framework for bridging architecting, engineering, and testing during software development. In order to clarify the necessity of achieving the connections, an investigation of the company’s current way of working was carried out in the beginning of the project. On account of the recognized difficulties, project goals and scope were defined. Furthermore, possible challenges were indicated.

3.1  Research on the Status Quo

The investigation of the current situation in the company was carried out with two complementary methods: Stakeholder Interview and Documentation Analysis. The former provides practical insights that is based on rich experience and the latter ensures the objectivity of the results.

The controller, as a main piece of print-related software, inherits the complexity of the print domain. The controller’s development covers multiple disciplines. It has been developed with high-technologies, by a large team, at different locations, and for over two decades. Currently, the controller code base contains more than fifteen million lines of code.

In addition, as concluded from the stakeholder interviews and the document analysis, development of the controller is basically document-based. A set of controller development documents play various roles in different phases of the controller’s lifecycle. In practice, documents are not always active and reliable, which is one of the disadvantages of the document-based methodology. Additionally, knowledge sharing and decision making typically depend on oral communications. The information involved in the oral communications may be missing in the documents or in the code base. As a result, inconsistence between the documents and the code base enlarges. Consequently, the inconsistence increases the difficulty of testing because both the documents and the code base need to be referred to.

3.2  Problem Definition

3.2.1  Problems

According to the research results of the status quo, generally speaking, the methodology, which the controller development team is employing, facilitate the controller’s development progress. Therefore, the team is able to deliver products that satisfy customers’ need both functionally and non-functionally. However, with business growth, the team undergoes higher pressures of product quality and delivery time. This project focused on solving three difficulties: Architecture Analysis of the controller, Impact Analysis of changes, and Usage Scenario Linkage between definition and implementation.

Architecture Analysis and Impact Analysis

Architecture Analysis, in the context of this project, indicates evaluating the controller’s decomposed structure by virtue of promoting identification of risks early in the life-cycle, forcing a clear explication of the architecture, creating a documented basis for architectural decisions, and eventually resulting in improved architecture practices. Impact Analysis is a main procedure that happens iteratively when any changes are required for designing a feature or an interface. Whereas, accomplishing Architecture Analysis and Impact Analysis is complicated due to the complexity of the controller. Additionally,
qualities of the analysis’ results cannot always be guaranteed because of the current development methodology.

The information that the analysis needs must be update-to-date and correct. Theoretically, the knowledge can be obtained from the documents, oral communication, as well as the controller’s code base. However, for the reason that the documents may contain out-of-date or even wrong information, only the latter two can be utilized. Collecting knowledge through oral communication depends on participants coming from various domains and acting diverse roles. On the one hand, this activity costs much time and effort due to the team’s size. On the other hand, results of discussion are valuable but their reliability proves to be dependent on the nature of oral communication. Therefore, neither documents nor oral communication can ensure the timeliness and correctness of the information.

Since code base always reflects the controller’s actual status, extracting information from the code base is identified as the most reliable approach. Inspecting code base can be performed through a web-based search engine. It ensures that the information is timely because the search engine retrieves data from the latest code base. Whereas, due to the complexity of code base, manual inspection might leave out authentic information and waste time on checking “noises”.

**Usage Scenario Linkage**

Usage Scenario stands for a particular type of requirement in this project. A usage scenario describes the expected results of one function or a series of functions being executed under certain conditions. In other words, usage scenarios define the actions or algorithms that need to be implemented for specific functions. At present, the definition and implementation of usage scenarios are separately carried out as textual descriptions and executable code respectively. Hence, inconsistency between them may exist, which cause difficulties in tracing whether a usage scenario is implemented and whether an implementation is required.

**3.2.2. Project Goal**

The goal of this project is devising a framework that supports the controller development team in resolving the problems elaborated in 3.2.1. As a result, this framework facilitates the controller’s development and maintenance from the three aspects consisting of

1. Ensuring timeliness and correctness of the information for Architecture Analysis and Impact Analysis by extracting the information from the most up-to-date code base.
2. Enhancing readability and maintainability of the information by presenting the information in a proper way.
3. Increasing traceability of usage scenarios, which describe the controller’s actions or algorithms, by connecting their definition and implementation.

**3.2.3. Project Scope**

Figure 3 displays the reasoning process between the project’s goal and scope. As illustrated on the right side, the construction of the framework is arranged into three phases. The phases altogether are designated to solve the stated problems and eventually improve the controller’s development and maintenance.
On the grounds of their purposes and functions, which are explained in details in the following chapters, their priorities are assigned as follows:

1. Controller Structure Modeling: Must have
2. Interface Modeling: Should have
3. Usage Scenario Modeling: Could have

The first two phases were accomplished by the author in the company with the help of stakeholders. The third phase was assigned to a group of undergraduate students and completed by them at Eindhoven University of Technology (TU/e) under the author’s supervision.

3.3 Challenges

3.3.1 Compatibility with Current Way of Working

Although the controller development team is encountering difficulties introduced by the complexity of the controller domain in addition to the present methodology and process, the team’s productivity is satisfactory and the quality of team’s deliveries is high. Therefore, the solution that I devised was meant to complement the team’s current way of working rather than to replace it.

3.3.2 Technology Acceptance by End Users

The controller development team is utilizing various technologies, such as general-purpose programming languages, general-purpose integrated development environment, company-specific interface definition languages, company-specific interface code generator, code editors, and Microsoft office suite.

To resolve the defined problems, several brand new technologies were considered to be applicable. Since some technologies are pioneering, one challenge of the project is to prove these cutting-edge technologies can not only realize end users’ requirements but also be easy to learn and utilize. Interviews with end users were arranged in the interest of analyzing technology acceptance.
Chapter 4     Domain Analysis

This chapter is to provide an in-depth analysis of the domains in which the project resides. The analysis results provide directions for the Model section including Architecture, Design, Implementation, in addition to Verification and Validation. Three topics, Modeling Object, Modeling Technology, and Controller Code Base, are discussed in detail.

4.1     Modeling Object

4.1.1.     C4 Model

According to the results of Document Analysis, the controller’s architecting consists of outlining its structure and devising its behavior. This method of architecting is commonly accepted and utilized for software development. As a starting point, Architecture Analysis, in the context of this project, concentrates on modeling the structure of the controller.

Architecture Analysis requires not only timely and correct information but also a proper presentation that illustrates sound information and organizes the information systematically, as discussed in 3.2. The former was achieved by only extracting the information that the end users demand; the latter was accomplished by following an introduced concept “C4 Model.”

C4 model defines “a software system is made up of one or more containers, each of which contains one or more components, which in turn are implemented by one or more classes.” This concept is for the purpose of arranging and presenting structures in different levels of abstraction, which are Context, Containers, Components, and Classes, assuming a software system is developed in object-oriented programming languages (e.g. Java, C#, and C++).

By analyzing the controller’s code base, C4 Model is recognized to provide a proper classification method of abstraction levels for modeling the controller’s structure. Detailed research results of the controller’s code base is elaborated in 4.3 Controller Code Base.

4.1.2.     Interface Definition

As stated in the controller’s architecture document, the controller contains several types of interfaces, such as hardware interfaces, logical interfaces, and component interfaces. Considering their different purposes, component interfaces were taken into consideration in this project.

A component interface, as a shared boundary across components, is the contract among components and enables communication between them. Since components possibly cross platforms, Interface Definition (ID) is introduced for ensuring interface consistency in a language-independent way. From an interface definition, interfaces can be generated into different programming languages as needed.

Generally, interface definitions are devised ahead of implementation. Moreover, they do not belong to C4 Model because interface definitions are

1. Developed in an Interface Definition language that is for specifying operations, parameters to these operations, and data types;
2. Contracts among subsystems and required or provided by components;
3. The original resource to generate code in different programming languages.
This way of defining interfaces is widely acknowledged in software development. Therefore, Interface Definition is regarded as an extra modeling object besides the C4 abstraction levels.

### 4.1.3. Behavior-Driven Development

Linking usage scenarios’ definition and implementation originates from the concept behavior-driven development (BDD). As an extension of test-driven development, BDD

1. Offers more precise guidance on organizing the conversation between developers, testers, and domain experts. [6]
2. Uses simple domain-specific languages (DSLs) that can express the behavior and the expected outcomes of the behavior. [7]
3. Provides notations (in particular the given-when-then canvas) that are closer to natural language and have a shallower learning curve. [6]

In addition, the tools targeting DSLs generally support the automatic generation of technical and end user documentation from BDD “specifications”. [6]

This project introduces BDD for designing, implementing, and testing interfaces. By devising a DSL, the usage scenarios, which describe the application of interface definitions’ elements, are defined as structured statements. Then, these statements are generated into unit tests for verifying functionalities of the interface definitions as well as mocks for validating the roles of the interface definitions in a business workflow.

### 4.2 Modeling Technology

Based on the comparison results of a list of possible technologies, JetBrains MPS was decided to be the appropriate technology of choice for realizing the project’s goals. JetBrains MPS is a metaprogramming system for designing DSLs. It is a language workbench for defining new computer languages and an integrated development environment (IDE) for such languages. [8]

JetBrains MPS distinguishes several types of modules: solutions, languages, devkits, and generators. The modules below were utilized in the project.

- **Solution:** is a set of models holding code and is unified under a common name.
- **Language:** is a module that represents a reusable language and consists of several models.
- **Generator:** defines possible transformations of a language into something else, typically into another languages.

Additionally, JetBrains MPS introduces a concept “Language Aspects” [9] that allows users to describe different facets of a language by combining JetBrains MPS Base Language and Language Definition Languages. The following aspects were considered to be basic and key to the framework’s construction.

- **Structure:** describes the nodes and structure of the language AST (Abstract Syntax Tree). This is the only mandatory aspect of any language.
- **Editor:** describes how a language will be presented and edited in the editor.
- **Behavior:** describes the behavioral aspect of AST, e.g. AST methods.
- **Intentions:** describes intentions which are context dependent actions available when light bulb pops up or when the user presses Alt + Enter.
- **Plugin:** allows a language to integrate into MPS IDE.
JetBrains MPS integrates a series of plugins, such as mbeddr, Aveco, YouTrack, die modelwerkstatt, PeoPL, and Meta R. Among them, two libraries, Graphic Editors [10] and Documentation [11], that mbeddr provides were adopted for the purpose of visualizing the controller structure and the interface definitions as well as generating documents from models.

4.3 Controller Code Base

As discussed in 3.2, checking the up-to-date code base is recognized as the best approach to guarantee the timeliness and correctness of design knowledge. The controller, that this project targets, belongs to the print domain. However, characteristics of its code base can also be observed from software of other domains.

On account of in-depth research on the controller code base, most code is written in object-oriented languages including Java, C#, and C++. From the points of development and compilation, the controller exists in two forms that are Source and Build respectively.
4.3.1. Source

The controller can be decomposed hierarchically into the C4 Model based on the arrangement of folders and files composing Source.

1. Context: the controller itself that resides in the top-lever folder.
2. Containers: subsystems that are clustered as sub-folders of the controller folder.
   A subsystem is a logical grouping of components which together fulfill a high level responsibility. Moreover, a subsystem defines interfaces that cross subsystem boundaries.
3. Components: components that are grouped as sub-folders of its subsystem folder.
   A component is the smallest unit of (potential) reuse and developed in a certain programming language. Additionally, a component requires or provides one or more interfaces.
4. Classes: classes that are in the format of files and are gathered in the sub-folders of its component folder.

Besides these four abstraction levels, Source Interface Definition, which are defined by subsystems, are grouped into an isolated sub-folder. The sub-folder is in the same abstraction level as subsystems are, whereas it is not counted as a subsystem in accordance with the research results summarized in Interface Definition.
4.3.2. Build

Build Interface Definition, which is in the same format as Source Interface Definition, and Executable Interface, which is generated from build interface definitions, were also taken into consideration for modeling the controller’s structure. Build stores compiled code which is clustered into subsystem folders. Besides, build interface definitions and executable interfaces together are grouped into a separated folder. Build code in subsystem folders are binary and executable. However, this type of code contributes no added value. In contrast, the build interface definitions, which contain complete interface design, are the actual resource to generate executable interfaces in different programming languages. Moreover, the executable interfaces’ names, which follow specific programming language naming conventions, are the input to discover the usages of the interfaces.

In summary, controller’s structure can be modeled with JetBrains MPS and presented as four abstraction levels according to the C4 Model (Context, Containers, Components, and Classes), in addition to Interfaces. Among these levels, information of C4 originates from Source; knowledge of Interfaces integrates data gained from both Source and Build.
Chapter 5  Requirement Analysis

Requirement Analysis was conducted after the problems were clarified and the domain was investigated. Requirements were elicited with the method of Stakeholder Interview. This chapter first describes the primary use cases which end users put forward; additionally, summarizes the functional requirements and quality requirements that the project satisfied. These requirements were prioritized following MoSCoW method\textsuperscript{[12]}.

5.1  Primary Use Cases

Primary use cases that the desired framework can support were discussed with the end users in the beginning of the requirement analysis phase. Figure 6 illustrates the primary use cases of the project’s three phases. On the grounds of them, a list of requirements with different priorities were determined.

1. Four active actors (on the left side outside the boundary), who are Controller Architect, Function Designer, Function Responsible/Feature Owner, and Function Engineer, interact with the system. Among them, Controller Architect is mainly interested in Controller Structure Modeling, and the other three concern all three aspects of modeling.
2. Regarding to Controller Structure Modeling, end users expect to inspect controller structure including extracting controller structure information, visualizing controller structure in C4 model, analyzing extracted information via visualization, and generating document for distribution and maintenance.
3. As for Interface Modeling, end users require modeling to facilitate interface design by supporting editing interface definitions, importing interface definitions, and exporting interface definitions.
4. In order to model the structure and the interfaces, knowledge can be discovered in two entities (Controller Code Base and Company-Specific Interface Definition System) in addition to learning from the four active actors.
5. Considering Usage Scenario Modeling, end users need to be able to define usage scenarios in a normative way so that unit tests and mocks can be generated from the definitions. Unit tests and mocks are for the purpose of verifying executable interfaces and validating interface definitions respectively.
6. Knowledge of modeling usage scenarios can be found in the controller’s code base. In addition, it can be gained from the four active actors as well as Product Tester and Workflow Architect (on the right side outside the boundary).
5.2 Functional Requirement

The functional requirements concern behavioral requirements that specify how a proposed system processes and handles information. It details the features and rules that must be present to fully implement the functionalities desired. Table 2, Table 7, and Table 9 present the statistics of requirements which belong to each phase of the project respectively. Requirements that are derived from several primary use cases are counted only once. The complete requirement list is one deliverable of the project. However, it is not included in the report because of confidentiality.

In order to explain the construction of the framework, fifteen requirements are illustrated in Table 3, Table 4, Table 5, Table 6, and Table 8. The statements of these requirements indicate the capabilities or
behaviors required to the framework. The way that they were realized is clarified in the following chapters.

5.2.1. Controller Structure Modeling

Among the four primary use cases, “Extract Controller Structure Information” is fundamental because structure modeling is based on the information that is extracted from the code base. With the extracted information, structure can be visualized in different abstraction levels. Subsequently, on the ground of the visualization, users are able to analyze the information for specific purposes and generate document as needed.

As Table 2 lists, all requirements of the Extract Controller Structure Information use case are prioritized as “Must have”. Requirements derived from the last two primary use cases present the actual applications of the models instead of the modeling itself. They were suggested by end users and were realized in order to prove the advantage of conducting Controller Structure Modeling.

<table>
<thead>
<tr>
<th>Primary Use Case</th>
<th>Total No. of Req.</th>
<th>Must have</th>
<th>Should have</th>
<th>Could have</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extract Controller Structure Information</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Visualize Controller Structure in Different Abstraction Levels</td>
<td>17</td>
<td>14</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Analyze Extracted Information via Visualization</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Generate Document</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

In accordance with the C4 Model concept, extracting the controller structure information is comprised of extracting the information of the controller itself, its subsystems, subsystems’ components, and components’ classes individually. Six requirements that cover extracting information of the four levels respectively are listed in Table 3.

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Extracting Subsystem Name</td>
<td>Must have</td>
</tr>
<tr>
<td>1.19</td>
<td>Extracting Subsystem Required Interface Definitions</td>
<td>Must have</td>
</tr>
<tr>
<td>1.20</td>
<td>Extracting Subsystem Provided Interface Definitions</td>
<td>Must have</td>
</tr>
<tr>
<td>1.23</td>
<td>Extracting Component Required Interface Definitions</td>
<td>Must have</td>
</tr>
<tr>
<td>1.26</td>
<td>Extracting Class Provided Interface Definitions</td>
<td>Must have</td>
</tr>
<tr>
<td>1.4</td>
<td>Extracting Elements Defined in Interface Definitions</td>
<td>Must have</td>
</tr>
</tbody>
</table>

The extracted information is organized and presented hierarchically according to the C4 model. For the reason of usability, a lower level can be activated in its nearest higher level. For example, in the demonstration of the controller’s overview (Context Level), the visualization of one of the controller’s
subsystems (Container Level) can be opened. Additionally, Interface Definition, as one extra modeling object, needs to be connected with the four abstraction levels respectively in order that end users can check the interface definitions’ content conveniently.

Table 4 Requirements: Visualize Controller Structure in Different Abstraction Levels

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.16.1</td>
<td>Linking Context Level to Container Level</td>
<td>Must have</td>
</tr>
<tr>
<td>1.17.3</td>
<td>Linking to Interface Definition in All the Four Levels</td>
<td>Must have</td>
</tr>
</tbody>
</table>

Moreover, on the basis of the visualized information, end users are able to analyze the controller’s architecture from many perspectives. Two requirements that concern basic usage of the framework are selected for this use case.

Table 5 Requirements: Analyze Extracted Information via Visualization

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.15.4</td>
<td>Highlighting Specific Interfaces in Context Level</td>
<td>Must have</td>
</tr>
<tr>
<td>1.40</td>
<td>Checking Dependency Diff between Definition &amp; Implementation</td>
<td>Must have</td>
</tr>
</tbody>
</table>

Furthermore, the models are required to be presented as structured documents. As a starting point, this project concentrates on generating documents for the controller and container levels. The requirement displayed Table 6 indicates that besides models, complementary descriptions are also necessary for document generation.

Table 6 Requirements: Generate Document

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.37</td>
<td>Presenting Subsystem Responsibility in Doc Model</td>
<td>Should have</td>
</tr>
</tbody>
</table>

5.2.2. Interface Modeling

Comparing with Controller Structure Modeling, Interface Modeling has less requirements considering that interface models, which are in the company-specific format, have been used for many years. Modeling interfaces in this project basically requires transforming the existing format to a new format that is consistent with the structure as well as devising new editors that are more user-friendly. The primary use cases “Import Interface Definition” and “Export Interface Definition” are for the sake of transformation. “Edit Interface Definition” involves providing editors that facilitate designing and developing interfaces.

Table 7 Summary of Interface Modeling Requirements

<table>
<thead>
<tr>
<th>Primary Use Case</th>
<th>Total No. of Req.</th>
<th>Must have</th>
<th>Should have</th>
<th>Could have</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import Interface Definition</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Export Interface Definition</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Edit Interface Definition</td>
<td>10</td>
<td>9</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
In favor of accomplishing all the three primary use cases, the concepts of Interface Definition DSL must be defined in the first place. In addition, Editing Interface Definition Content in Graphical Editor allows end users to inspect how the relationship among elements are changed when they add, remove, or modify interface definition elements.

**Table 8 Requirements: Interface Modeling**

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Defining Company Interface Definition System Compliant DSL</td>
<td>Must have</td>
</tr>
<tr>
<td>2.5</td>
<td>Editing Interface Definition Content in Graphical Editor</td>
<td>Must have</td>
</tr>
<tr>
<td>2.9</td>
<td>Paring Existing Interface Definitions Based on the DSL</td>
<td>Must have</td>
</tr>
<tr>
<td>2.8</td>
<td>Generating Interface Definitions Based on the DSL</td>
<td>Must have</td>
</tr>
</tbody>
</table>

5.2.3. Usage Scenario Modeling

Requirement analysis of Usage Scenario Modeling was conducted roughly in the project because this phase was prioritized as “Could have” and was carried out by the undergraduate students at TU/e. The general requirements that were assigned to the students are summarized in Table 9. Based on these requirements, the students accomplished the assignment within ten weeks successfully. More importantly, the output proved that connecting usage scenario definition and implementation is feasible. The way that the students completed the development and the details of their deliverables are not included in this report.

**Table 9 Summary of Usage Scenario Modeling Requirements**

<table>
<thead>
<tr>
<th>Primary Use Case</th>
<th>Total No. of Req.</th>
<th>Must have</th>
<th>Should have</th>
<th>Could have</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define Usage Scenarios Normatively</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Generate Unit Tests</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GenerateMocks</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

5.3 Quality Requirement

The quality requirements specify the various operational parameters that define the environment in which the system exists. These are criteria which define Correctness, Completeness, Adaptability, and Changeability requirements.

**Table 10 Summary of Quality Requirements**

<table>
<thead>
<tr>
<th>Quality</th>
<th>Total No. of Req.</th>
<th>Must have</th>
<th>Should have</th>
<th>Could have</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctness</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Completeness</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Adaptability</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Changeability</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
1. Correctness

From the perspective of Controller Structure Modeling, correctness indicates that the models must accurately reflect the as-is architecture of the controller. As for Interface Modeling, correctness obligates that the content of interface definition must be the same as that of the original one in both cases of importing to the new format or exporting to the existing format.

2. Completeness

Completeness, in the context of Controller Structure Modeling, means the extracted information must cover all relevant input that was aforementioned in 4.3. About Interface Modeling, completeness signifies all valid interface definitions must be able to be imported, exported, and edited.

3. Adaptability

Since controller code base is organized as branches in the interest of product variety, modeling of both the controller structure and the interface must be applicable to any branches.

4. Changeability

Changeability focuses on Controller Structure Modeling and requires that the tool of extracting structure information should be able to be added with extra functionalities.

The requirements (summarized in Requirement Analysis) are the input defined by stakeholders (discussed in Stakeholder Analysis) for devising a solution for the problems (identified in Problem Analysis) in the defined domains (analyzed in Domain Analysis). The process of developing the framework is elaborated in the next chapters that are Architecture, Design, and Implementation.
Chapter 6  Architecture

Architecture concerns an overview that gives direction to design, implementation, verification and validation, and evolution of the framework. Architecture Reasoning Model (ARM) [3] provides a guideline of analyzing the software architecture of a system as well as making decisions from three dimensions including Application, Construction, and Realization.

The framework’s architecture is illustrated in Figure 7. Three types of resource, Controller Development Documents, Controller Code Base, and Company-specific Interface Definition Language, are necessary for constructing the framework. Their contributions to each phase are elaborated in 6.1. Since Usage Scenario Modeling was completed outside the company, this phase is presented as a dashed box.

![Figure 7 Architecture of the Framework](image)

6.1  Application

Application indicates the functionalities that a solution need to provide. These functionalities originate from primary use cases and they were devised on the grounds of functional requirements determined in Requirement Analysis. As illustrated in Figure 8, the solution was divided into three phases which are Controller Structure Modeling, Interface Modeling, and Usage Scenario Modeling.
6.1.1. Controller Structure Modeling

Two subsystems, “Structure Info Extractor” and “Structure Visualizer”, were designated to realize functional requirements related to modeling the controller’s structure. As their names imply, “Structure Info Extractor” (divided into three sub-components) is for the purpose of extracting information from the controller code base and “Structure Visualizer” is to demonstrate the extracted information.
following the C4 Model. The output of the former subsystem is the input of the later one. Moreover, responsibilities of subsystems are extracted from the controller development documents.

6.1.2. Interface Modeling

![Interface Modeling Diagram]

“Interface Definition DSL” enables a new format of Interface Definition, which was accomplished by utilizing DSL. This DSL must be compatible with the company-specific interface definition system so that the exported interface definitions can be used for generating executable interfaces. “Interface Definition Visualization” supports users to edit the content of Interface Definition textually and graphically, which is regarded to be more user-friendly. Generating executable interfaces from interface definitions is presented as dashed boxes and dashed lines because it is carried out by the company-specific interface definition system. Including this generation progress is to illustrate the relationship with the framework’s applications.

6.1.3. Usage Scenario Modeling

![Usage Scenario Modeling Diagram]

On the grounds of BDD, a DSL is the foundation of describing the behaviors and expected outcomes of usage scenarios. From the defined usage scenarios, unit tests and mocks, in different programming languages, are generated for verification and validation of interfaces.
6.1.4. Relationship among These Phases

As described in the requirement 1.17.3 (Table 4), “Structure Visualizer” provides the connection between the C4 models and interface models. Additionally, regarding to the requirement 1.15.4 (Table 5), “Structure Visualizer” displays only the subsystems that have specific interface definitions as their provided or required interfaces. Both requirements concern the interaction between Controller Structure Modeling and Interface Modeling.

As mentioned in 4.1.3, this project employed BDD for verifying the functionalities of implemented interface definitions and validating interface definitions’ roles in business workflows. Each usage scenario has its target interface definitions. Moreover, defining a usage scenario depends on the elements belonging to its target interface definitions. Therefore, Interface Modeling and Usage Scenario Modeling also have communications.

6.2 Construction

Construction expresses design and configuration of existing or to-be-acquired software and the connections between them. Main aspects, which need to be contemplated, are composed of existing infrastructure and available technology. These two aspects ensure feasibility of the proposed solution.

6.2.1. Existing Infrastructure

As discussed in Domain Analysis, Controller Code Base is the most important input in both modeling phases. In order to retrieve up-to-date information, the Structure Info Extractor application must be connected to the code base server. In addition, with the purpose of enabling Adaptability, Structure Info Extractor needs to be located in the server where the application can access all branches.

Besides the code base, the Interface Definition system, which the company is currently employing, restricts especially Interface Modeling. The application Interface Definition DSL needs to support all valid elements that Interface Definition requires. In addition, the elements’ types, relationship, and constraints must be consistent with those are defined in Océ’s Interface Definition system.

6.2.2. Available Technology

JetBrains MPS and JetBrains MPS Plugin

As explained in Modeling Technology, JetBrains MPS was utilized for devising DSLs for Controller Structure Modeling and Interface Modeling. Furthermore, mbeddr, a JetBrains MPS plugin, was employed with the purpose of visualizing the extracted information and interface definitions as UML-like diagrams as well as presenting the models as structured documents.

Programming Language

Given the quality requirements, design patterns were adapted to building the framework. Accordingly, programming language to implement the framework must be object-oriented, which support design patterns in good order. In addition, concepts’ behaviors (a language aspect in JetBrains MPS) also need to be developed in a certain programming language. Three programming languages (Java, C#, and C++) were considered to be applicable. Given that I have more experience in Java, Java was selected for implementing not only the Structure Info Extractor but also the concepts’ behaviors. Before this project, the controller development team executed a Python script to search components’ dependencies. The
functions defined in the script were re-written in Java in order to better integrate them to the “Structure Info Extractor” application.

Others

Four other technologies were also employed

- Enterprise Architect: organizing models created in the project’s life cycle for various purposes, including domain analysis, requirement analysis, design, in addition to verification and validation.
- Microsoft Visio: modeling the architecture of the project’s solution.
- Microsoft Excel: managing the project’s milestones and risks as well as tracking the process
- Git: controlling versions of the project’s output

6.3 Realization

Realization dimension focuses on development approaches and principles to apply. The development approach must conform to the capabilities of the controller development team and the principles must be consistent with which the controller development team follows. Additionally, the decisions reached in the other two dimensions also have impact on determining realization techniques.

6.3.1 Development Approach

Development Process

Agile development process was employed in the project’s entire lifecycle. Because this approach advocates adaptive planning, evolutionary development, early delivery, and continuous improvement as well as encourages rapid and flexible response to change.

Implementation Strategy

As aforementioned in Construction, an existing Python script fulfilled one functional requirement. However, due to Python’s limitation, the script needed to be re-written in Java programming language. Besides functions of the script, the others of the applications, which were discussed in Application, were all developed from the scratch.

Team Composition

Besides stakeholders (including end users), several other groups were also involved, who were supervisors, experts, and undergraduate students. The supervisors guided me by providing instructions of constructing the solution and reviewing the project’s deliveries. The experts, who are from the university, the company, and the field, helped me by granting theoretical and practical suggestions. The undergraduate students conducted a project which originated from my project. Their delivery that was related to visualization was adapted to my solution. Additionally, their output proves the concept of Usage Scenario Modeling is applicable.

Verification and Validation

Verification includes all the activities associated with producing a high-quality solution. In this project, verification tasks consisted of specification analysis, design inspection, and output review.

Validation is intended to ensure the solution to meet the operational needs of the users. In this project, reviewing by end users were organized several times for collecting their feedback.
6.3.2. General Principles

Java Code Conventions

The Structure Info Extractor application and concepts’ behaviors and intentions in JetBrains MPS were written in Java. In order to standardize the code, the *Code Conventions for the Java Programming Language*[^13] document was employed since the document contains commonly-accepted constrains and guidelines for Java developers.

Model-Based Development

Models were created for domain analysis, requirement analysis, architecture reasoning, application design (partially), DSL design, in addition to verification and validation. These models facilitated communicating as well as making decisions with stakeholders.

Model-Driven Development

Model-Driven Development was mainly adopted to the development of “Structure Info Extractor”. This method helps ensure the consistency between design and implementation of this application.
Chapter 7  Design

The previous chapter, Architecture, presents a high-level overview of the framework. This chapter elaborates the process of devising the applications which fit the existing infrastructure, utilizing the available technologies, as well as following the development approach and the general principles. The design decisions reached in this chapter serve as guidelines of the subsequent implementation.

7.1  Controller Structure Modeling

7.1.1.  Structure Info Extractor

Structure Info Extract is designated to extract the up-to-data design knowledge from the controller code base. It was decomposed into three components, which are Structure Explorer, Interface Name Translator, and Interface Usage Finder, as illustrated in Figure 12. During design, Codebase Configuration was also introduced. Relationship among these four components is demonstrated in Figure 12.

![Figure 12 Design of Structure Info Extractor](image)

**Structure Info Extractor – Codebase Configuration**

Codebase Configuration performs as the foundation of Structure Info Extractor. The component depends on config.properties and layer_mapping.csv. The former allows end users to define a specific branch, including both source and build, from which Structure Info Extractor extracts information. The latter indicates each subsystem’s matching virtual layer. Besides, the component provides an interface
named CodeBaseConfig. Through this interface, three returned data types are accessed by Structure Explorer and Interface Name Translator.

Structure Info Extractor – Interface Name Translator

Interface Name Translator is to translate element names that conform to company-specific interface definition naming conventions into names that are programming language compliant. In accordance with the categories of the executable interfaces, names of three interface definition elements were translated into Java, C#, and C++ names in the project. Moreover, this component implements the InterfaceNameTranslator interface that enables Interface Usage Finder to retrieve the translated names.

In consideration of Changeability (a quality requirement), Abstract Factory Pattern was employed so that Interface Name Translator has the ability of translating extra ID elements’ names into additional programming language specific names.

Structure Info Extractor – Interface Usage Finder

Interface Usage Finder searches the translated names in Source of the controller code base. Since the translated names are grouped by programming languages, the search process is also on the grounds of a programming language that a component employs. Additionally, search when the programming language matches improves the execution performance. Search results are categorized per interface definition and are accessible through the InterfaceUsagefinder interface. InterfaceNameTranslator and InterfaceUsageFinder together are the main components for realizing the requirement 1.26 and 1.4 (Table 3).

Given the quality requirement, Changeability, Strategy Pattern was introduced. When a new programming language is required, an algorithm of searching the translated names in the new programming language can be added without affecting other algorithms.

Structure Info Extractor – Structure Explorer

Structure Explorer resides on the top level of Structure Info Extractor. It requires data from the other three components and provides the StructureExplorer interface to Client. This component explores Source of the controller code base and extracts valid design knowledge for each abstraction level interactively. The output of the component is StructureInfo.xml that stores the retrieved information. For instance, the information that the requirement 1.1, 1.19, 1.20, and 1.23 demand were retrieved as follows:

- A subsystem name is the name of the subsystem folder.
- A subsystem’s required interface definitions are the combination of its components’ required interface definitions and are not provided by the subsystem’s components.
- A subsystem’s provided interface definitions are the combination of its components’ provided interface definitions.
- A component’s required interface definitions are defined in the component’s project file.

7.1.2. Structure Visualizer

Structure Visualizer is for the primary use cases Visualize Controller Structure in Different Abstraction Levels, Analyze Extracted Information via Visualization, and Generate Document. The design of the application (Figure 14) was conducted aligning with JetBrains MPS project Structure (Figure 13). The
relationship among controller structure models and the four language aspects were formulated based on the configuration of JetBrains MPS (Figure 15).

**Figure 13 Project Structure of Structure Visualizer**

**Figure 14 Decomposition of Structure Visualizer**

**Figure 15 Design of Structure Visualizer**

**Solution**

Solution is a set of models holding code and unified under a common name. The controller structure models are organized in accordance with the Source file system of the controller code base and illustrated by the composite aggregation notation (a binary association decorated with a filled black diamond at the aggregate end \(^{[14]}\)). For example, the notation between Controller and Subsystem
indicates that a controller contains one or more subsystems; if the controller is deleted, all its subsystems are deleted too.

Additionally, build interface definition models are created because a build interface definition contains a complete set of elements as explained in 4.3.2. The arrangement of controller structure models and interface definition models is determined by the design of the controller structure modeling language.

Moreover, DocumentModel is provided by the mbeddr plugin for the purpose of generating documents. As a starting point, only the controller and the subsystem were designed to contain doc models in this project. These models were utilized to satisfy the requirement 1.37 (Table 6).

Figure 16 Design of Controller Structure Modeling Solution

Language – Structure

Structure describes the nodes and structure of the language AST. Structure is also regarded as the metamodel of a DSL. This is the only mandatory aspect. To support creating the models designed in Solution, Controller Structure Modeling DSL is structured as displayed in Figure 17.

- Considering importing the extracted information to the controller structure models and displaying them following the C4 Model, a generalized concept StructureConcept was devised. Therefore, the information that the requirements listed in Table 3 demand can be imported to the corresponding models and displayed in any levels as necessary.
- In view of the requirement 1.16.1 (Table 4), two design decisions were made. One is introducing a generalized concept StructureReference; the other is assigning controller, subsystem, component, and class as root models. These two combined enable the linkage between two directly-connected abstraction levels. For example, in the Controller model, users can choose to check one of its subsystems by activating a Subsystem model.
Concerning the requirement 1.17.3 (Table 4), interfaceDefinition concept is imported from Interface Modeling language. Therefore, end users can select a build interface definition to analyze in any abstraction levels.

In order to realize the requirement 1.15.4 (Table 5), the QueryData concept is devised for storing an end user’s selection of interfaces. For example, subsystems with specific interfaces as their required or provided interfaces can be highlighted in the Controller model (the context level).

The explanations of other concepts can be found in the design document (one of the company deliverables), which is not included in the report.

Language – Editor

Editor describes how a language is presented and edited in the editor. Two editors were designed for presenting controller structure graphically and textually respectively. Among them, the graphical editor was designated as the default one. Additionally, Concept Editor Hint allows end users to choose the editor which they want to employ.

1. Graphical editor: the controller’s structure is illustrated as UML-like diagrams. The C4 concepts are in the shape of box. Their interfaces, including required interfaces and provided interfaces, are in lollipop and socket notations.

2. Textual editor: Content is displayed as plain text. A model in one abstraction level displays its own attributes and owning models that are in the next abstraction level.

Language – Behavior

Behavior describes the behavioral aspect of concepts. Behaviors were devised for all specialized StructureConcept, their reference concepts, and the interface concept in order to support end users in...
analyzing the extracted information. For instance, cooperating with intentions and editors, they realize
the functionalities described in the requirement 1.15.4 (Table 5).

Language – Intentions

The Intentions aspect describes intentions which are context dependent actions available when light
bulb pops up or when the user presses Alt + Enter. Together with other language aspects, the following
intentions were conceived to realize the requirements listed in Table 4, Table 5, and Table 6.

1. importStructureXML: importing StructureInfo.xml as controller structure models as well as the
descriptions of the subsystems’ responsibilities as document models.
2. importBuildID: importing build interface definitions as interface definition models.
3. selectControllerInterface: highlighting specific interfaces and their requiring/providing subsys-
tems in the Controller model. The interfaces are selected by end users from an interface list that contains all
names of the interface definitions belonging to the controller.
4. checkDifference: checking the difference between the interface definitions that are defined in a
component’s project file and the interface definitions that are actually provided/required by the
component.

The functionality of selectControllerInterface is also provided by the selectInterfaceAllLevel intension.
However, selectInterfaceAllLevel restricts the selection of the interface definitions to the container level.
It means the interface list, from which end users can select interfaces, contains only the names of the
interface definitions belonging to a specific subsystem. More importantly, this intention enables
highlighting specific interfaces in other three abstraction levels related to a subsystem including

   a. Highlighting specific interfaces and their requiring/providing components in the subsystem
      model.
   b. Highlighting specific interfaces and their requiring/providing classes in the subsystem’s
      composed component models.
   c. Displaying specific interfaces in the subsystem’s composed class models if a class requires or
      provides the interfaces.

Highlighting specific interfaces in all the four abstraction levels is not applicable to the entire controller
due to the big size of the controller as well as the potential risk of JetBrains MPS crash.

One more intention resetSelected was devised for the sake of resetting all the controller structure
models to the status when they are originally imported and created.

7.2 Interface Modeling

As Figure 10 presented, Interface Modeling contains two applications that are Interface Definition DSL
and Interface Definition Visualization. In JetBrains MPS, they were reorganized following JetBrains MPS
project structure (Figure 18 and Figure 19). Accordingly, the design of Interface Modeling was
formulated in accordance with the project structure (Figure 20).
The models, in the context of Interface Modeling, are the imported source interface definitions because the source interface definitions are allowed to be edited for designing interfaces. The source interface definitions in the original format are parsed based on a new DSL and are presented in a different format. Moreover, the models are editable including modifying or removing the exiting elements in addition to adding new elements. The following two constrains on the content of the source interface definitions must be satisfied:

1. If end users do not edit a model, the content of the model must be the same as that of the original source interface definition.
2. If end users edit a model, the content of the model must be the same as that of the generated source interface definition.
7.2.2. Language

Language - Structure

As the only mandatory aspect, structure is the infrastructure of Interface Definition DSL as well as the foundation of Solution and Generator.

Concerning the requirement 2.1 (Table 8), constructing the structure of the DSL was basically transforming the company’s current rules of specifying interface definition. Therefore, all valid elements must be available; additionally elements’ types, relationship, and constraints must be correct and complete.

Two interface concepts were utilized by virtue of simplicity.

- INamedConcept: concepts with “name” as a property implement the pre-defined interface.
- IDocumentable: concepts with “doc” as a child element implement the user-defined interface.

Furthermore, extra concepts (Figure 21) were devised in favor of realizing the requirement 2.5 (Table 8).

- IHasType: concepts with “type” as a property implement the user-defined interface.
- IDataType: The type of the property “type” was assigned as IDataType in the interest of supporting the visualization of elements’ relationship.
- PrimitiveType: PrimitiveType implements IDataType and represents primitive data types of the property “type”.
- Reference and ReferenceConcept: Reference implements IDataType and refers to ReferenceConcept. ReferenceConcept is the target of Reference and represents user-defined types of the property “type”.

Moreover, with the purpose of displaying text in Rich Text Format, a pre-defined concept Text, which is provided by de.slisson.mps.richtext (a dependency), was applied to three concepts which are doc, pre, and post (Figure 22).
Two styles of editors were devised for different purposes of usages. By defining Concept Editor Hint, users are able to choose a presentation style.

1. Concise Style: the content of Interface Definition is displayed as plain text. Additionally, tags and makeups that are used to hold the company-specific format were removed. Editor in concise style is set as default.

2. Graphical Style: Interface Definition is illustrated as UML-like diagrams. Not only the content but also the relationship among elements is demonstrated.

Language – Behavior

One method was defined as Reference’s behavior that ensures returning the target of a reference concept. The reference-target relationship is actually the relationship among elements. In addition, PrimitiveType owns the behavior of returning the name a primitive data type. Besides, doc, pre, and post concepts possess a method respectively that is used for normalizing its text.

Language – Intentions

Functionalities of the intentions enable end users to import source interface definitions, in the original format, to JetBrains MPS as source interface definition models. Combined with the language structure, editor, and behavior, the intentions realized the requirement 2.9 (Table 8). The importing application was decomposed based on the language structure.

7.2.3. Generator

Generator defines possible transformations of a language into something else, typically into another languages. Concerning the requirement 2.8 (Table 8), one generator was conceived to transform the new-formatted interface definitions back to the original format. Subsequently, the generated interface definition serves as the input to generating executable interfaces in multiple programming languages.
Chapter 8  Implementation

The previous two chapters, Architecture and Design, explained how the discovered problems can be solved. This chapter focuses on the realization process and illustrates a prototype of the framework.

8.1  Controller Structure Modeling

8.1.1  Structure Info Extractor

Workflow

Structure Info Extractor (SIE) was implemented in accordance with the model-driven methodology. Java code was generated from class diagrams as well as reversed back to class diagrams (Figure 23). This procedure ensures the consistency between design and implementation.

![Figure 23 Structure Info Extractor Implementation Workflow]

- From Class Diagram to Java Classes
  In class diagrams, attributes and operations of each class were defined. In addition, classes’ relationship, such as composition and inheritance, was also devised. Then, from the class diagrams, Java classes were generated with these designed attributes and operations in addition to their relationship.

- From Java Classes to Class Diagram
  To realize required functionalities, algorithms/actions were completed in Java Classes. During this process, extra attributes and operations were introduced. Hence, by reversing the Java classes, class diagrams were updated.

Use Case

<table>
<thead>
<tr>
<th>Primary Use Case</th>
<th>Extract Controller Structure Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Goal</td>
<td>Extract the structure information of a specific feature branch of the controller</td>
</tr>
</tbody>
</table>
| Pre-conditions   | 1. Eclipse (neon.2) is installed.  
  2. The Java project is open in Eclipse. |
| Requirements     | 1.1, 1.19, 1.20, 1.23, 1.26, 1.4        |
Scenarios

1. **EU (End User)**
   Specify values of the properties defined in the configuration file `config.properties` in the resources folder.

2. **EU**
   Run TestDrive.java in the package com.oce.strctureexplorer as Java Application.

3. **SIE**
   Start to extract information and need about twenty three minutes to finish.

4. **SIE**
   Generate StructureInfo.xml (Figure 24) to the Source path.

![Prototype of Extracted Information in XML Format]

Figure 24 Prototype of Extracted Information in XML Format

8.1.2. **Structure Visualizer**

**Workflow**

Structure Visualizer (SV) was implemented by JetBrains MPS. As designed, Language (including structure, editor, behavior, and intentions) and Solution were completed adhering to the workflow displayed in Figure 25. As the foundation of a DSL, structure must be constructed first. Then the other language aspects were completed for specific concepts. Based on the DSL, models were created. The models’ content was imported from the StructureInfo.xml (the output of Structure Info Extractor). Additionally, demonstration of the content contributed to improving the language aspects as needed.
Use Case

<table>
<thead>
<tr>
<th>Primary Use Case</th>
<th>Visualize Controller Structure in Different Abstraction Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User Goal</strong></td>
<td>Active a subsystem model in the controller model to analyze the subsystem’s decomposition</td>
</tr>
</tbody>
</table>
| **Pre-conditions** | 1. JetBrains MPS is installed.  
2. The mbeddr plugin is installed.  
3. The JetBrains MPS project is launched.  
4. Controller structure models are created by executing the intension “Import Controller Structure XML”.  
5. Build interface definition models are created by executing the intension “Import ID XML from Build”. |
| **Requirements** | 1.16.1 |
| **Scenarios**    | |  
1. **EU** Double-click the controller model.  
2. **SV** The controller model is open (Figure 26).  
3. **EU** Move the cursor to the name label of a subsystem.  
4. **EU** Press Ctrl and left-click the mouse.  
5. **SV** A subsystem model is open (Figure 27). |
### Primary Use Case
Visualize Controller Structure in Different Abstraction Levels

#### User Goal
Activate a build interface definition model in the controller model to analyze the content of the build interface definition.

#### Pre-conditions
1. JetBrains MPS is installed.
2. The mbeddr plugin is installed.
3. The JetBrains MPS project is launched.
4. Controller structure models are created by executing the intension “Import Controller Structure XML”.
5. Build interface definition models are created by executing the intension “Import ID XML from Build”.

#### Requirements
1.17.3

#### Scenarios
1. **EU** Double-click the controller model.
2. **SV** The controller model is open (Figure 26).
3. **EU** Execute the intension “Display Relationship of Structure & Interface in Context Level”.
4. **SV** Demonstrate the relationship of structure symbols and build interface definition symbols (Figure 28).
5. **EU** Move the cursor to the name label of an interface definition.
### 6. EU
Press Ctrl and left-click the mouse.

### 7. SV
A build interface definition model is open (Figure 29).

---

<table>
<thead>
<tr>
<th>Primary Use Case</th>
<th>Analyze Extracted Information via Visualization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User Goal</strong></td>
<td>Select specific interfaces definitions to analyze changes of these interface definitions may have impacts on which subsystems</td>
</tr>
</tbody>
</table>
| **Pre-conditions** | 1. JetBrains MPS is installed.  
2. The mbeddr plugin is installed.  
3. The JetBrains MPS project is launched.  
4. Controller structure models are created by executing the intension “Import Controller Structure XML”.  
5. Build interface definition models are created by executing the intension “Import ID XML from Build”. |
<p>| <strong>Requirements</strong> | 1.15.4 |
| <strong>Scenarios</strong> | |
| 1. EU | Double-click the controller model. |
| 2. SV | The controller model is open (Figure 26). |
| 3. EU | Execute the intension “Display Relationship of Structure &amp; Interface in Context Level”. |
| 4. SV | Demonstrate the relationship of structure symbols and build interface definition symbols (Figure 28). |</p>
<table>
<thead>
<tr>
<th>Primary Use Case</th>
<th>Analyze Extracted Information via Visualization</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Goal</td>
<td>Check difference between the dependencies that are defined and that are actually required by a component</td>
</tr>
</tbody>
</table>
| Pre-conditions        | 1. JetBrains MPS is installed.  
                      | 2. The mbeddr plugin is installed.  
                      | 3. The JetBrains MPS project is launched.  
                      | 4. Controller structure models are created by executing the intension “Import Controller Structure XML”.  
                      | 5. Build interface definition models are created by executing the intension “Import ID XML from Build”. |
| Requirements          | 1.40 |
| Scenarios             | |
| 1. EU                 | Double-click the controller model. |
| 2. SV                 | The controller model is open (Figure 26). |
| 3. EU                 | Execute the intension “Check Difference Between Declared and Actually Used Interface Definitions”. |
| 4. SV                 | Generate the results as a txt file per subsystem (Figure 30). |

![Figure 30 Dependency Difference of subsys1](image)

<table>
<thead>
<tr>
<th>Primary Use Case</th>
<th>Generate Document</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Goal</td>
<td>Check a subsystem’s responsibilities</td>
</tr>
</tbody>
</table>
| Pre-conditions        | 1. JetBrains MPS is installed.  
                      | 2. The mbeddr plugin is installed.  
                      | 3. The JetBrains MPS project is launched.  
                      | 4. Doc models are created by executing the intension “Import Controller Structure XML”. |
| Requirements          | 1.37 |
| Scenarios             | |
| 1. EU                 | Double-click a doc model. |
| 2. SV                 | The doc model is open (Figure 31). |
8.2 Interface Modeling

Workflow

Interface Modeling (IM) was implemented in JetBrains MPS. As designed, Language (including structure, editor, behavior, and intentions), Solution, and Generator were completed adhering to the workflow illustrated in Figure 32.

The same as the Structure Visualizer, structure was constructed first. Then the other language aspects were completed for specific concepts. Models were created based on the DSL and used for improving language aspects. Moreover, generator was produced for the purpose of generating source interface definitions when any changes were operated by end users.

Use Case

<table>
<thead>
<tr>
<th>Primary Use Case</th>
<th>Import Interface Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User Goal</strong></td>
<td>Import an existing source interface definitions to check its content</td>
</tr>
<tr>
<td><strong>Pre-conditions</strong></td>
<td>1. JetBrains MPS is installed. 2. The mbeddr plugin is installed.</td>
</tr>
</tbody>
</table>
3. The JetBrains MPS project is launched.
4. An empty interface definition model is created.

**Requirements**

2.1, 2.9

**Scenarios**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. EU</td>
<td>Double-click the no-name interface definition model.</td>
</tr>
<tr>
<td>2. IM</td>
<td>The no-name interface definition model is open (Figure 33).</td>
</tr>
<tr>
<td>3. EU</td>
<td>Execute the intension “Import from Interface Definition File”.</td>
</tr>
<tr>
<td>4. EU</td>
<td>Select a source interface definition and open the file.</td>
</tr>
<tr>
<td>5. SV</td>
<td>The content of the selected file is imported and displayed in the text editor in concise-style (Figure 34).</td>
</tr>
</tbody>
</table>

---

**Primary Use Case**

**Edit Interface Definition**

**User Goal**

Edit source interface definitions in graphical editor and check the changes’ impacts on the elements’ relationship

**Pre-conditions**

1. JetBrains MPS is installed.
2. The mbeddr plugin is installed.
3. The JetBrains MPS project is launched.
4. A source interface definition model is imported.

<table>
<thead>
<tr>
<th>Requirements</th>
<th>2.5</th>
</tr>
</thead>
</table>

### Scenarios

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. EU</td>
<td>Double-click the interface definition model.</td>
</tr>
<tr>
<td>2. IM</td>
<td>The interface definition model is open (Figure 34).</td>
</tr>
<tr>
<td>3. EU</td>
<td>Right-click any space of the model and choose “Push Editor Hints”.</td>
</tr>
<tr>
<td>4. EU</td>
<td>On the pop-up window, check “Use custom hints”.</td>
</tr>
<tr>
<td>5. EU</td>
<td>In the “com.oce.controller.InterfaceDefinition” group, check “GUI: overview” and click “OK” to confirm.</td>
</tr>
<tr>
<td>6. SV</td>
<td>The content of the interface definition and the elements’ relationship are displayed as a UML-like diagram in the graphical editor (Figure 35).</td>
</tr>
<tr>
<td>7. EU</td>
<td>Activate “Diagram Palette” (Figure 36).</td>
</tr>
<tr>
<td>8. EU</td>
<td>Edit the content by modifying the existing elements or add new elements.</td>
</tr>
<tr>
<td>9. SV</td>
<td>The content is changed and the elements’ relationship is changed accordingly when applicable (Figure 37).</td>
</tr>
</tbody>
</table>

### Primary Use Case

#### Export Interface Definition

<table>
<thead>
<tr>
<th>User Goal</th>
<th>Generate source interface definitions that are in the company-specific interface definition format.</th>
</tr>
</thead>
</table>
| Pre-conditions | 1. JetBrains MPS is installed.  
2. The mbeddr plugin is installed.  
3. The JetBrains MPS project is launched.  
4. A source interface definition model is imported. |
| Requirements | 2.8 |

### Scenarios

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. EU</td>
<td>Double-click the interface definition model.</td>
</tr>
<tr>
<td>2. IM</td>
<td>The interface definition model is open (Figure 34).</td>
</tr>
<tr>
<td>3. EU</td>
<td>Right-click the Model root and choose “Rebuild Model”.</td>
</tr>
<tr>
<td>4. IM</td>
<td>The interface definition is generated to its original format.</td>
</tr>
</tbody>
</table>
Interface Definition: ID5.3

Figure 35: an Interface Definition Displayed in Graphical Editor
Chapter 9  Verification & Validation

Chapter 6 to Chapter 8 elaborated the architecture, design, and implementation of the framework. This chapter explains the process and techniques of ensuring that the framework realized the devised functionalities properly (verification) as well as satisfied the stated requirements certainly (validation).

9.1  Controller Structure Modeling

Figure 38 Verification & Validation of Controller Structure Modeling

9.1.1  Verification

Verification of Controller Structure Modeling includes manual checking and end user reviewing. These two techniques focus on Structure Info Extractor and Structure Visualizer respectively as displayed in Figure 38.

Execution results of Structure Info Extraction’s functions were checked manually during the development the application, although conventionally functions can be verified through unit testing. Unit testing was considered to be unsuitable, because

1. Algorithms/actions of the application, including exploring folders, reading files, converting strings, and regular expressions, are simple.
2. Errors, if any, are mainly caused by personal preference of writing code instead of the algorithms/actions themselves.

In addition, design of the functions were documented in details. Therefore, whether the functions have correct algorithms/actions and whether the functions cover all possible cases of writing code can be examined through the design document.

Regarding to Structure Visualizer, verification was achieved by end user reviewing and carried out in two steps. First, recorded videos of the prototype were sent to the end users. These videos allow the end
users to gain an overview and a basic understanding of the prototype’s functionalities. Sequentially, the end users were invited to operate the prototype. During this process, the end users checked the correctness and completeness of the extracted information. Comparing to plain text, the visualized extracted information is more readable and understandable. Moreover, the end users own the knowledge from various aspects of the controller. Their expertise ensures the reviewing results’ reliability.

9.1.2. Validation

Validation of Controller Structure Modeling consists of requirement tracing and end user reviewing. Requirement tracing allows the end users to confirm that their requirements were implemented. Requirement tracing was conducted to components. The tracing results of the requirements listed in 5.2.1 were achieved as illustrated in Figure 39 and Figure 40. Moreover, by operating the prototype, the end users validated whether the functionalities meet their needs. Feedback and also additional expectations were collected during their reviewing.

Figure 39 Requirement Tracing Results of Structure Info Extractor
Figure 40 Requirement Tracing Results of Structure Visualizer
9.2 Interface Modeling

9.2.1 Verification

Four techniques were utilized to verify the DSL that was devised for Interface Modeling. Expert reviewing is compulsory to check the DSL’s structure, because properties of and relationship among the concepts defined in the DSL must be consistent with the company-specific interface definition system. End user reviewing, validating generated file, and comparing content were adopted by virtue of checking the correctness of imported content. Among them, the latter two were essential since they were conducted with particular tools which guarantees results’ accuracy.

Based on the expert reviewing, revision of the structure was performed. The following differences between the DSL structure and the company-specific interface definition system were approved by the experts:

1. Properties of several elements were removed since they were either not applicable to the elements or not generated to executable interfaces.
2. Default value of an element was inconsistent because the value in the company-specific interface definition system was incorrect.
3. The type of a property was changed for the sake of supporting the visualization of elements’ relationship.

The other language aspects were revised, as necessary, in view of the results achieved with all the four verification techniques. Two bugs of generator were detected and fixed. One was caused by changing the type of a property and the other was about special characters written in the original source interface
definition. Moreover, a set of decisions were carried out in order to provide a unified format of generated files.

1. Properties of several elements are optional and have no default values. The generator generates these properties only if their values are defined explicitly.
2. Properties of several elements are optional and have default values. When they are not defined explicitly, the generator generates these properties with their default values. If they are defined with specific values, the generator generates these properties with their defined values.
3. Only properties of the root elements are generated even the original source interface definitions may include extra data since the data is only supplementary information.

9.2.2. Validation

Requirement tracing and end use reviewing were also applicable to Interface Modeling. In accordance with the component-level design, displayed in Figure 20, the tracing results of the requirement listed in 5.2.2 were achieved as demonstrated in Figure 42. The end user reviewing focused on collecting user experience feedbacks of editing elements through graphical and textual editors.

![Figure 42 Requirement Tracing Results of Interface Modeling](image-url)
Chapter 10  Results & Recommendations

Among the previous chapters, Chapter 2 to Chapter 5 explained why to construct the framework as well as what the stakeholders required; then Chapter 6 to Chapter 9 elaborated how the framework was constructed and the stakeholders’ requirements were realized. This chapter summarizes the achievements and the added values to the stakeholder. Moreover, recommendations are provided for improving the controller’s architecture and further developing the framework.

10.1 Results

The project consists of three phases: the Controller Structure Modeling and Interface Modeling phases were carried out in the company; while the Usage Scenario Modeling phase was completed by a group of undergraduate students at TU/e.

Considering the first two phases, all the must-have functional requirements, some should-have functional requirements, and all the quality requirements were satisfied, according to the requirement tracing results. Functionally, the end users are able to

1. Extract the controller’s structure information from the controller code base as needed.
2. Visualize the information in four abstraction levels (the C4 Model) to obtain an overview of the controller’s structure.
3. Import and visualize the controller’s interface definitions to examine the relationship among elements of the interface definitions.
4. Analyze the controller’s architecture on the grounds of the C4 models.
5. Evaluate the usage of the build interface definitions based on the C4 models and the build interface definition models.
6. Edit the source interface definitions in both a concise-style textual editor and a UML-like graphical editor.
7. Generate the modified source interface definitions into the company-specific interface definition format.
8. Investigate impacts on the controller’s architecture and elements’ relationship when changes are applied to the interface definitions.

Given the third phase, two primary use cases were satisfied by the students successfully. Functionally, the end users are able to

1. Define usage scenarios of interface definitions following pre-defined rules.
2. Generate unit tests to verify the functionalities of the interface definitions.

Moreover, regarding to the quality requirements, the following accomplishments were gained:

1. Completeness: the extracted information covers all the four abstraction levels and contains the complete knowledge that is considered to be essential for architecture analysis.
2. Correctness: the extracted information correctly reflects the current status of the code base; the visualization of the controller structure demonstrates correct relationship and statistics; the imported interface definitions have the correct content; and the generated interface definitions have the correct format and content.
3. Adaptability: the framework is applicable to any branches of the controller code base by simply modifying the Branch property in the configuration file (config.properties).

4. Changeability: the framework is modifiable given that the framework’s entire development process is well documented, especially requirement analysis, architecture, and design.

Completeness and Correctness were evaluated during verifying and validating the output of Controller Structure Modeling and Interface Modeling. Adaptability and Changeability are related to one of the design criteria which constrained the project. The evaluation matrix and results are elaborated in 12.2.2.

In conclusion, the framework supports the controller development team in obtaining a clear explication of the controller’s architecture, identifying risks early in the controller’s development life-cycle, establishing a documented basis for architectural decisions, and eventually achieving an enhanced architecture practice.

10.2 Recommendations

Recommendations encompass both the usage and evolvement of the framework. The former facilitates revising the controller’s architecture by presenting additional facts; and the latter offers instructions of developing the framework in order to enable more functionalities.

Besides the extracted information, an extra log is also generated during the process of exploring the code base. This log contains possible deficiencies of the controller’s architecture, such as inconsistent names, unconventional formats of interface definitions, and inappropriate definition of dependencies. Hence, the supplementary log supports the end users in inspecting potential risks that may be caused by these deficiencies.

For further developing the framework, the following aspects are recommended to be considered:

1. This framework contains two languages (Structure Definition and Interface Definition) that were devised in JetBrains MPS for Controller Structure Modeling and Interface Modeling. Currently, the structure aspect is regarded as being well-defined after constantly reviewing and revising. In order to support more functionalities, the other language aspects need further modification.

2. Regarding to the primary use case Analyze Extracted Information via Visualization, the available functionalities were implemented by creating functions in the behavior and intentions aspects. At present, the potential end users are only able to inspect certain statistics and execute pre-defined queries, which limits the inspection of the extracted information. Therefore, a query language is suggested with the purpose of supporting the end users in customizing their usage of the framework.

3. The framework, which was constructed in the project, resides in the print domain and targets a print controller system. Whereas, on account of its architecture and design, the framework is also applicable to other software if the software is developed with similar techniques. Since some software systems, which compose a printing product, are indeed comparable to the controller, adapting the framework to those software systems can also benefit their development and maintenance.

4. Given the functionalities provided by Usage Scenario Modeling, integrating the deliverables to Controller Structure Modeling and Interface Modeling is also recommended. With those functionalities, the framework realized the connection between architecting and testing as well as engineering and testing.
Chapter 11  Project Management

This chapter first summarizes the project management aspects. Among these aspects, Risk Management and Milestone Management are explained in detail respectively. Risk Management was conducted to avoid or resolve potential issues in advance; additionally, Milestone Management was adopted to control the project progress in order to ensure that the project can be accomplished successfully.

11.1 Management Aspects

The management of this project concerns five aspects: Meeting Management, Sprint Management, Session Management, Risk Management, and Milestone Management. Meeting Management regulates how a meeting needs to be arranged in addition to what needs to be delivered before and after a meeting. Sprint Management and Session Management are for the development of the framework. Since the agile methodology was employed, the development process was divided into nine sessions; and each session was comprised of several one-week sprints. Sprint Management and Session Management altogether define the tasks to be completed in a sprint/session and track the tasks’ status during a sprint/session. Risk Management describes the risks of the project and their status’ changing process during the project. Milestone Management elaborates the dynamics of the project progress.

11.2 Risk Management

Risks of the project were managed qualitatively using the method Probability and Impact Matrix. As the framework’s development was conducted iteratively, each risk was evaluated repetitively in the beginning of a session by re-grading its probability of occurrence and its impact on the project if it occurs. In addition, extra risks were added during the project because of practical reasons. Figure 43 and Figure 44 demonstrated the risks’ initial status and one of their intermediate status respectively.

![Figure 43 Probability and Impact Matrix of Session 2](image-url)
Originally, twelve categories of issues, which may cause project failure, were taken into account. Among them, Requirement Issues, Planning, in addition to Architecture and Design were considered to have high probability of occurrence and high impact if they happen. In contrast, Team Issues was not identified as a risk. The reasons are as follows:

1. Since requirements guide and constrain the functionalities and qualities of the framework, analyzing requirements comprehensively and correctly is essential to the project’s success. Additionally, in the beginning of the project, the stakeholders provided miscellaneous requirements without a well-defined target. Hence, the possibility of not organizing these requirements properly was believed to be high.

2. Planning being regarded as an issue was concluded from the lessons learnt in the first session of the project. During Session 1, several tasks were scheduled. However, the tasks’ time was estimated positively and their priorities were not arranged properly. Therefore, Planning was an issue in the early stage of the project.

3. Architecture and Design was believed to affect the project’s success because the author was lack of experience in constructing such a framework. In addition, at the time of analyzing the risks, the architecture was not completed and the design was not devised yet. Hence, the Architecture and Design risk required more attention initially.

4. Team Issues was not taken into consideration at that moment because the project was planned to be carried out by the author individually.

As the project continued, the status of the twelve risks were evolved, especially to the left part of the matrix by virtue of proactive and continuous review by the supervisors. Whereas, among them, the status of Team issues was changed contrarily. Additionally, two more risks were added that are Unavailable Code Base and Inappropriate Technology.
1. The Team issues risk was regarded as being more influential because of the involvement of a group of undergraduate students. From Session 5, the students started to participate in the project and took responsibilities for Usage Scenario Modeling. However, as originally determined, this phase had the lowest priority and was planned to be conducted after the Controller Structure Modeling and Interface Modeling phases. Due to the alteration of the plan, the currently undergoing tasks were interrupted; moreover, extra time and effort were required to supervise the students.

2. The two extra risks were counted considering the specific situations of the project. Since extracting information from the controller’s code base is the fundamental functionality of the framework, the availability of the code base must be guaranteed. The Inappropriate Technology risk indicates particularly the uncertainties of JetBrains MPS. As the technology was decided to be employed in all the three phases, JetBrains MPS had high influence on the project.

11.3 Milestone Management

Milestone Trend Analysis (MTA)\(^{[16]}\) was employed in order to monitor and analyze the project progress. During the project, a number of milestones were defined. The milestones may be achieved in advance, on time, delayed, or even cancelled from the project plan. Since MTA was constantly updated on a sprint basis, the dynamics of each milestone was recorded. This information supports in documenting the project progress clearly and precisely; additionally, it enables a simple interpretation and, if necessary, a timely correction.

![Figure 45 Project Plan on 21 July 2017](image-url)
Figure 45 illustrates the status of the project’s milestones and their changing trends. On the basis of the timeline, the following matters need to be mentioned:

1. In the first three sessions, a number of milestones were delayed due to being lack of experience.
2. During Session 4, Session 5, and Session 6, milestones’ planning was changed significantly because of two reasons. The first is the participation of the students. On the one hand, some undergoing tasks were interrupted due to unexpected extra time on supervising the students; on the other hand, some tasks were planned to be delayed deliberately in view of reusing the students’ deliverables. The second reason is the evolvement of requirements. Since the framework was constructed iteratively, feedback was also collected iteratively. Moreover, based on the domes, stakeholders formulated a clearer view of the project’s goals and accordingly clearer expectations of the framework.
3. From Session 7 to Session 9, milestones that were related to the Usage Scenario Modeling phase were combined. The new milestone was to integrate the students’ output to the report given the success of their assignment and the adjustment of the project scope.
Chapter 12 Reflection

In this chapter, difficulties that the author encountered and solved in the project are described. In addition, the definition and evaluation of the design criteria are also discussed.

12.1 Difficulties & Solutions

Besides the traced risks, some other problems also happened during this period of time and were successfully resolved with the help of the supervisors, the stakeholders, as well as both the university colleagues and the company colleagues.

1. In the beginning of the project, due to ambiguous project goals, a large number of documents, concepts, and technologies were proposed to be investigated. Therefore, reading the documents, understanding the concepts, and experimenting the technologies costed much time in the early stage. Fortunately, the knowledge learnt in this phase contributed significantly to the Architecture, Design, Implementation, as well as Verification and Validation phases. Therefore, the project was still completed on time.

2. Based on the investigation results, JetBrains MPS was decided to be utilized. However, the technology is young and advanced so that tutorials or examples are difficult to obtain, comparing with other matured technologies. Therefore, continuous efforts were invested for learning the technology and inspecting its possibilities. Moreover, JetBrains MPS does not support in implementing certain functionalities of the framework directly, alternative solutions were figured out to compensate its insufficiencies.

3. Because of the controller’s massive lines of code, exploring the entire code base and extracting its structure information requested plenty of time, which were the results of the first several demos. In order to improve the performance, a series of revisions and experiments were conducted to the Structure Info Extractor application.

4. Due to the performance limitation of JetBrains MPS, executing some queries, which involves the whole set of the controller’s structure information, may lead to system crash. Hence, such queries were designed to only be performed on each abstraction level individually.

12.2 Design Criteria

Considering the project’s goals and scope as well as the functional requirements and quality requirements, three design opportunities were identified to be applicable. They are Impact, Genericity, and Documentation. In contrast, Complexity and Methodical approach were not taken into account.

12.2.1 Impact

Impact criteria is relevant in a way of measuring the influence of the project’s output in the company. As explained in 3.3, this framework is required to complement rather than replace the controller development team’s the current way of working. Therefore, whether the framework will be employed in the team’s daily work depends on the stakeholders’ satisfaction of the framework. The evaluation was carried out by interviewing the stakeholders.
Metrics

1. Perceived usefulness: the degree to which a person believes that using a particular system would enhance his or her job performance.\[^{17}\]
   a. Very Satisfied
   b. Satisfied
   c. OK
   d. Dissatisfied
   e. Very Dissatisfied

2. Perceived ease of use: the degree to which a person believes that using a particular system would be free from effort.\[^{17}\]
   a. Very Satisfied
   b. Satisfied
   c. OK
   d. Dissatisfied
   e. Very Dissatisfied

12.2.2. Genericity

Genericity criteria is meant to evaluate whether the framework can be employed to other context including different feature branches, diverse interface definitions, and additional software. As a pilot, this project focused on a specific branch of the controller code base and a particular interface definition. Therefore, the framework must be appropriate for this feature branch and this interface definition. However, the controller code base is composed of a quantity of feature branches and a number of interface definitions. Hence, the framework is required be applicable to all the feature branches and interface definitions. In addition, since the problems discovered from the controller may also exist in other software not only of the printing products but also of the products in other domains, the framework is expected to be suitable for the controller’s similar software too.

Metrics

1. Feature Branch: Scale a-e for grading the lines of code that need to be modified in order to execute the framework against other feature branches.
   a. 80% -- 100% Modification
   b. 60% -- 80% Modification
   c. 40% -- 60% Modification
   d. 20% -- 40% Modification
   e. 0% -- 20% Modification

2. Interface Definition: Scale a-e for grading the lines of code that need to be modified in order to execute the framework against other interface definitions.
   a. 80% -- 100% Modification
   b. 60% -- 80% Modification
   c. 40% -- 60% Modification
   d. 20% -- 40% Modification
   e. 0% -- 20% Modification

3. Similar Software: Scale a-e for grading the lines of code that need to be modified in order to execute the framework against other software.
12.2.3. Documentation

Documentation is a supplementary design criteria that is for the purpose of ensuring Impact and Genericity. From Impact point of view, documenting stakeholder interviews (type 1), problem analysis (type 2), and requirement analysis (type 3) helps the stakeholders to trace back their motivation of the project as well as evaluate whether the framework resolves the problems. Additionally, User Guide (type 4) is also necessary for ensuring the utilization of the framework. Regarding to genericity, for the sake of adapting the framework to other context and extending the framework’s usages, architecture details (type 5) and design details (type 6) are the most important materials to refer to.

Metrics

1. Completeness: deliverables contain a certain collection of documents that stakeholders can gain the information as they need.
   1. Complete: including all the six types
   2. Adequate: including type 3, type 4, type 5, and type 6
   3. Incomplete: not including one of type 3 to type 6
2. Readability: the documents contain essential text and diagrams so that a reader can understand the content simply. The evaluation is conducted by interviewing the stakeholders.
   a. Very Satisfied
   b. Satisfied
   c. OK
   d. Dissatisfied
   e. Very Dissatisfied

The two criteria, Complexity and Methodical Approach, are considered to be inapplicable to be project because of the following reasons:

1. Complexity, in the context of the project, concerns how to develop and how to use the framework. High complexity of these two aspects may degrade the framework’s usability and extensibility; and consequently, reduce the probability of achieving the Impact and Genericity criteria.
2. The development of the framework was completed with a set of new concepts and state-of-the-art technologies, which did not follow any rules of choosing techniques. In addition, the appearance of the models are based on the stakeholder’s preference instead of any modeling standards.

12.3 Evaluation of Design Criteria

As defined in 12.2, three design criteria, Impact, Genericity, and Documentation, are applicable to the project. The evaluation was carried out based on their metrics respectively.

Regarding to Impact, about ten stakeholders participated in the stakeholder interview and provided their assessment of the project’s output. In general, they are satisfied with the achieved results and
recognized the framework to be useful and easy to use. At the same time, they offered some suggestions of further improving the framework, which were summarized in 10.2.

Genericity concerns that the framework can be adapted to other feature branches, interface definitions, and similar software. First, adopting the framework to various interface definitions requires no effort thanks to the employment of the domain specific language technology. Additionally, utilizing the framework to different feature branches just needs to modify at most five lines of code in a configuration file. Furthermore, the effort of applying the framework to comparable software depends on the similarity between the controller and the target software. However, theoretically, the adaption is considered to be possible.

Considering Documentation, the collection of delivered documents is regarded as being complete since it consists of stakeholder interviews (type 1), problem analysis (type 2), requirement analysis (type 3) user guide (type 4), architecture details (type 5), and design details (type 6). In addition, stakeholders are also satisfied with the documents’ content.
Reference


About the Author

Yinghui Wu completed her bachelor’s study in Software Engineering at Xiamen University. After that she was recommended to a master’s program in Software Engineering at Peking University and received her master’s degree in 2012. Her Master thesis entitled “Research on Workflow Model Suitable for Localization Engineering Business” was to specify the most appropriate workflow models that can be used to describe L10n engineering business. After she graduated she worked for about 3 years as a Senior Software Engineer at Nokia. She participated in the development and maintenance of Nokia S30 and Asha products. Furthermore, as a technical leader, she was in charge of the overall localization process of Nokia X series with technical and managerial tasks. During this two-year PDEng program in TU/e, she acquired cutting-edge knowledge and down-to-earth skills by attending advanced lectures, practicing in industrial projects, and tutoring undergraduate students.