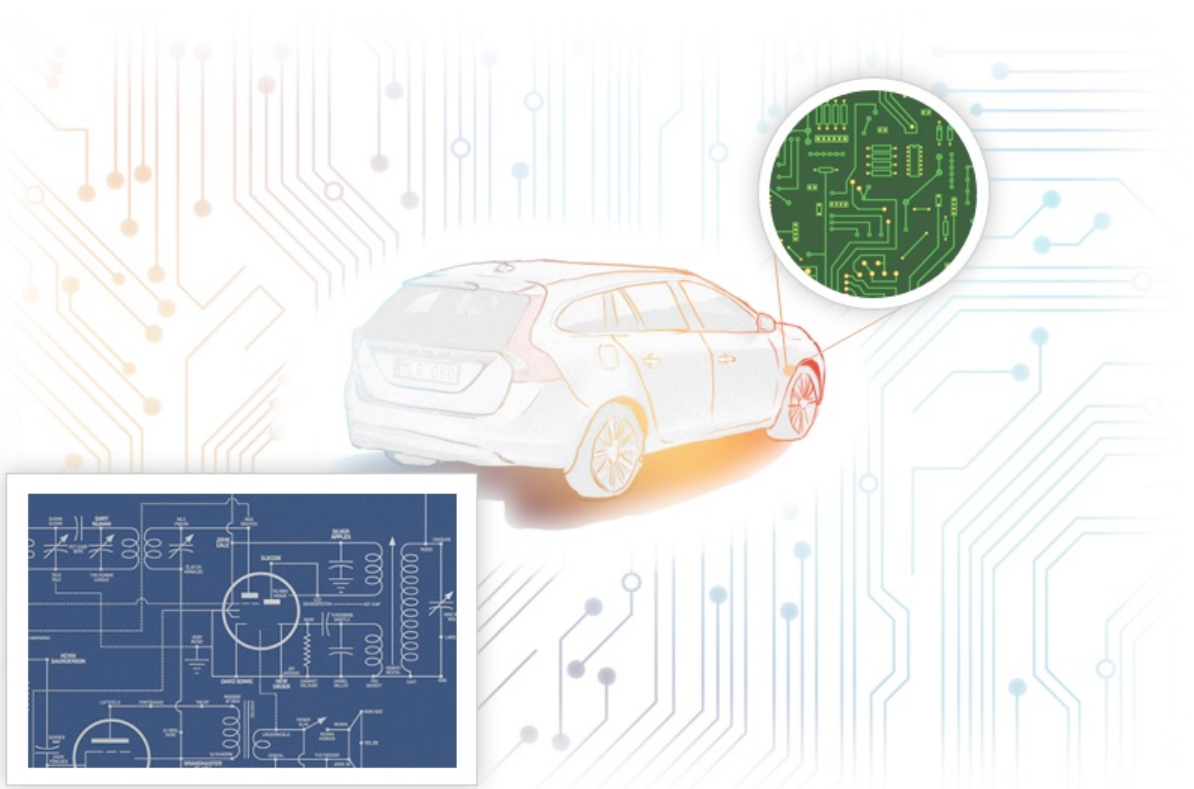




CHALMERS
UNIVERSITY OF TECHNOLOGY



UNIVERSITY OF GOTHENBURG



Visualization of Electrical Architectures In the Automotive Domain Based on the Needs of Stakeholders

Master's thesis in Software Engineering

FLORENCE MAYO

NATTAPON THATHONG

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2016

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a noncommercial purpose make it accessible on the Internet. The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

Visualization of Electrical Architectures In Automotive Domain Based On the Needs of Stakeholders

FLORENCE MAYO,
NATTAPON THATHONG

© FLORENCE MAYO, JUNE 2016.

© NATTAPON THATHONG, JUNE 2016.

Examiner: ERIC KNAUSS

Supervisor: MICHEL CHAUDRON, PATRIZIO PELLICCIONE,
TRUONG HO QUANG, ULF ELIASSON

Master's Thesis 2016:NN

Department of Computer Science and Engineering

Division of Software Engineering

Chalmers University of Technology and University of Gothenburg

SE-412 96 Gothenburg

Telephone +46 31 772 1000

Cover: Electrical Architecture (designed by: Juntima Nawilajaroen)

Typeset in L^AT_EX

Gothenburg, Sweden 2016

Abstract

The use of software in automotive engineering keeps on growing higher every year. This has an impact on a data stored in a database in such a way that a structure of data stored becomes complex due to hierarchy and hence difficult to understand. The benefit of visualizing a complex data structure of a system facilitates a quick learning of how a system work and a general understanding of where to locate a particular piece of data in a database. A large focus in the industries is placed more on innovating new features of a car by improving current software built in a car's system and also developing new software. There is rather less focus in visualizing the software which have already been implemented in a car's system since the visualization does not provide direct value. The purpose of this study is to provide an automated visualization of a current implementation of software data that is stored in a database and to understand the needs of different stakeholders who work with the database. The visualization is meant to cover the needs, which are functional and non-functional requirements, of different stakeholders interacting with a database to aid the understanding of a system and to facilitate decision making in their work. Our focus of the study has been to understand the field of automotive software engineering, its architecture, understanding how to get the needs of stakeholders, implementing an automated software visualization and finally gather more stakeholders' needs regarding the visualization of data. The visualization was done on a small sub-system of the car showing how the logical components were connected to one another via input and output signals, this covers a view in an automotive architecture called a logical view. On the later stage of the thesis, we interviewed other stakeholders to gather more needs towards automated visualization. One of their needs we gathered was that the stakeholders wanted to have another view which is named a physical view in the automotive industry. In addition to that, the stakeholders proposed to have an interactive way when visualizing the data, meaning that the visualization should provide the ability to view less or more details, to be able to filter the contents, to sort the contents and also to have the output that can easily be understood without cracking your brain.

Keywords: architecture, automated visualization, automotive, needs, stakeholders

Acknowledgements

We would like to express our gratitude to our supervisors Truong Ho Quang, Michel Chaudron, Ulf Eliasson, and Patricio Pelliccione, who have challenged us to think in a broader perspective so that we achieved the intended results. We are grateful for the opportunity to do a thesis with Volvo Car Group and that has helped us to get a wider understanding of what is taking place in automotive field. We appreciate the support that we obtained from the employees at the company and most of all, the support from Ulf who was our supervisor from the company. We are thankful for the support he gave us in getting familiar with the custom proprietary tool and also connected us with different stakeholders. We would like to express our gratitude to our examiner, Eric Knauss for supporting and guiding us in the process to make sure we follow the correct way of doing a master thesis work. We would also like to thank all participants who have contributed one way or another in this thesis.

Gothenburg, June 2016
Florence Mayo, Nattapon Thathong.

Contents

| | |
|---|-----------|
| List of abbreviations | xv |
| 1 Introduction | 1 |
| 1.1 Statement of the problem | 2 |
| 1.2 Purpose | 2 |
| 1.3 Research questions | 3 |
| 1.4 Contribution of the study | 3 |
| 1.5 Outline of the report | 4 |
| 2 Theoretical Framework | 5 |
| 2.1 Architectures in automotive domain | 5 |
| 2.1.1 Low- and high-level electrical architectures at VCG | 8 |
| 2.2 Software Architecture Visualization | 10 |
| 2.3 Model-Driven Software Engineering | 11 |
| 2.3.1 Modeling languages | 11 |
| 2.3.1.1 Meta-models, model instances, and semantics | 12 |
| 2.3.2 Model transformation | 13 |
| 2.4 Graphical notation of textual description | 14 |
| 2.5 Stakeholders | 16 |
| 2.5.1 Needs of stakeholders | 17 |
| 3 Methodology | 19 |
| 3.1 Focus selection | 20 |
| 3.2 Data collection | 21 |
| 3.2.1 Raw JSON data | 22 |
| 3.2.2 Interview data | 22 |
| 3.3 Data analysis and interpretation | 23 |
| 3.3.1 Meta-model of JSON data | 23 |
| 3.3.2 Optimization of JSON data | 24 |
| 3.3.3 Coding | 24 |
| 3.4 Take action | 25 |
| 4 Implementation | 27 |
| 4.1 Cycle 1: Creating automated visualization | 27 |
| 4.1.1 Focus selection | 27 |
| 4.1.2 Data collection | 29 |
| 4.1.3 Data analysis and interpretation | 29 |

| | | |
|----------|--|-----------|
| 4.1.3.1 | Analyzing raw JSON data | 29 |
| 4.1.3.2 | Optimizing raw JSON data | 33 |
| 4.1.4 | Take action | 34 |
| 4.1.4.1 | New meta-model and model instance | 34 |
| 4.1.4.2 | Automated visualization prototype | 36 |
| 4.1.4.3 | Final automated visualization prototype | 39 |
| 4.2 | Preliminary to cycle 2 | 41 |
| 4.2.1 | Selecting stakeholders | 41 |
| 4.2.2 | Preparation for interviews | 41 |
| 4.3 | Cycle 2: Identifying needs of stakeholders | 42 |
| 4.3.1 | Data collection | 42 |
| 4.3.2 | Data analysis and interpretation | 42 |
| 4.3.2.1 | Coding interview transcripts | 43 |
| 4.3.2.2 | Categorizing needs of stakeholders | 44 |
| 5 | Results | 45 |
| 5.1 | Automated visualization | 45 |
| 5.1.1 | Automated visualization prototype | 45 |
| 5.1.2 | Final automated visualization prototype | 46 |
| 5.2 | Coding results | 48 |
| 5.2.1 | Category: Personal info | 48 |
| 5.2.2 | Category: Use of the Database | 49 |
| 5.2.3 | Category: Specific task | 51 |
| 5.2.4 | Category: Automated visualization | 52 |
| 5.3 | Categories of needs of stakeholders | 55 |
| 5.3.1 | Dependencies | 55 |
| 5.3.2 | Clusters | 56 |
| 5.3.3 | Features | 56 |
| 6 | Discussion | 59 |
| 6.1 | Research questions | 59 |
| 6.2 | Use of the source code | 61 |
| 6.2.1 | Tools required | 61 |
| 6.2.1.1 | Open source projects | 61 |
| 6.2.1.2 | Eclipse software & plugins | 62 |
| 6.2.2 | Installation | 62 |
| 6.2.3 | Deliverables | 63 |
| 6.2.3.1 | Optimizing JSON document | 63 |
| 6.2.3.2 | Creating a meta-model and a model instance from JSON document | 63 |
| 6.2.3.3 | Generating a visualization | 63 |
| 7 | Validity Threats and Research Ethics | 65 |
| 7.1 | Validity Threats | 65 |
| 7.1.1 | Conclusion validity | 65 |
| 7.1.2 | Internal validity | 65 |
| 7.1.3 | Construct validity | 66 |

| | | |
|----------|---|-----------|
| 7.1.4 | External validity | 67 |
| 7.2 | Ethical consideration | 67 |
| 7.2.1 | Informed consent | 67 |
| 7.2.2 | Anonymity and confidentiality | 67 |
| 7.2.3 | Fraud | 68 |
| 8 | Conclusion and Future work | 69 |
| 8.1 | Conclusion | 69 |
| 8.1.1 | Sustainability | 70 |
| 8.2 | Future work | 71 |
| | Bibliography | 73 |
| A | Acceleo templates | I |
| A.1 | Acceleo template for the automated visualization prototype | I |
| A.2 | Acceleo template for the final version of the automated visualization | II |
| B | Interview questions and transcripts | V |
| B.1 | Interview questions | V |

List of Figures

| | | |
|------|---|----|
| 1.1 | An example of electrical architecture of the Volvo XC90 | 1 |
| 2.1 | Meta-model of UML-RT constructs for logical architecture. | 6 |
| 2.2 | Automotive development process. | 7 |
| 2.3 | Automotive architectural views. | 8 |
| 2.4 | The difference between architectural design and actual deployment. | 9 |
| 2.5 | Three main ingredients that comprise a modeling language. | 12 |
| 2.6 | sWML model's abstract syntax and graphical concrete syntax. | 12 |
| 2.7 | An example of M2T transformation. | 13 |
| 2.8 | UML class diagram rendered from the textual description. | 15 |
| 3.1 | Steps in action research. | 19 |
| 3.2 | Steps performed in the study. | 20 |
| 3.3 | Elicitation technique of stakeholder analysis. | 21 |
| 3.4 | A screenshot of Graphical Ecore Editor in Eclipse. | 24 |
| 4.1 | Overview of the steps performed in this study. | 27 |
| 4.2 | A screenshot of the Database. | 28 |
| 4.3 | An example of a visualization given by the software engineer. | 29 |
| 4.4 | Meta-model of the raw JSON document. | 31 |
| 4.5 | Meta-model of the optimized JSON document | 35 |
| 4.6 | Model instance from the optimized JSON document. | 36 |
| 5.1 | Automated visualization prototype. | 46 |
| 5.2 | Final version of the automated visualization. | 47 |
| 5.3 | Coding results of test engineer for Category: Personal info. | 48 |
| 5.4 | Coding results of software developer for Category: Personal info. | 49 |
| 5.5 | Coding results of system designer for Category: Personal info. | 49 |
| 5.6 | Coding results of test engineer for Category: Use of the Database. | 50 |
| 5.7 | Coding results of software developer for Category: Use of the Database. | 50 |
| 5.8 | Coding results of system designer for Category: Use of the Database. | 51 |
| 5.9 | Coding results of test engineer for Category: Specific task. | 51 |
| 5.10 | Coding results of software developer for Category: Specific task. | 52 |
| 5.11 | Coding results of system designer for Category: Specific task. | 52 |
| 5.12 | Coding results of test engineer for Category: Automated visualization. | 53 |
| 5.13 | Coding results of software developer for Category: Automated visualization. | 54 |

| | | |
|------|--|----|
| 5.14 | Coding results of system designer for Category: Automated visualization. | 54 |
| 5.15 | Needs in Dependencies group. | 55 |
| 5.16 | Needs in Clusters group. | 56 |
| 5.17 | Needs in Features group. | 57 |
| 8.1 | The sketch of a logical view and a physical view. | 71 |
| B.1 | List of pre-determined questions. | V |

List of Tables

| | | |
|-----|--|----|
| 2.1 | Some useful syntax for UML component diagrams. | 16 |
| 3.1 | Stakeholder who participated in cycle 1. | 21 |
| 3.2 | Stakeholders who participated in cycle 2. | 22 |

List of abbreviations

| | |
|-------------------|---|
| ADL | Architecture Description Language |
| API | Application Program Interface |
| CAN | Controller Area Network |
| DSL | Domain-Specific Language |
| ECU | Electrical Control Unit |
| EMF | Eclipse Modeling Framework |
| EPS | Electrical Power System |
| ER diagram | Entity-Relationship diagram |
| DVM | Design Verification Method |
| GPML, GML, or GPL | General-Purpose Modeling Language |
| HIT | The Hybrid Innovations for Trucks project |
| IBD | Internal Block Diagram |
| IDE | Integrated Development Environment |
| JSON | JavaScript Object Notation |
| LAC | Logical Architectural Component |
| LC | Logical Component |
| M2M | Model-to-model transformation |
| M2T | Model-to-text transformation |
| MDSE | Model-Driven Software Engineering |
| SRD | System Requirement Description |
| SWC | Software Composition |
| sWML | simple Web Modeling Language |
| SysML | Systems Modeling Language |
| UML | Unified Modeling Language |
| UML-RT | Unified Modeling Language for Real-Time |
| URI | Uniform Resource Identifier |
| VCG | Volvo Car Group |
| XMI | XML Metadata Interchange |
| XML | Extensible Markup Language |

1

Introduction

One of the biggest challenges in the automotive industry is to build vehicles that meet customer expectations. To overcome the challenge, more complex system of mechanical and electrical artifacts are designed and built in modern vehicles. Besides that, a huge number of software are also integrated in order to carry out certain tasks, resulting in a number of 70 of Electrical Control Unit (ECUs) deployed in a car [23]. The example is shown in Figure 1.1 which indicates the ECUs deployed to the Volvo car. For this reason, having the automated visualization of the complex system is of large importance which in turn can be used to verify that every artifact and the connections between them are in the right places.

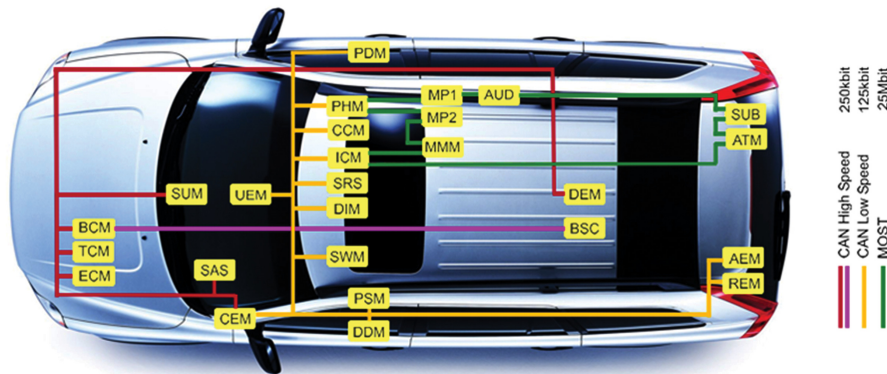


Figure 1.1: An example of electrical architecture of the Volvo XC90 [24].

This thesis is done at Volvo Car Group (VCG) which is a Swedish premium automotive manufacturer that produces modern vehicles to the world. At the company, two types of architectures for electrical system are used to handle complex systems of cars [10]. The two architectures have different abstraction levels, meaning that they serve different purposes. A high-level architecture (logical view) contains Logical Architectural Components (LAC) designed by software architects with a purpose of guiding and breaking down work to be developed during implementation phase. Development team creates a low-level architecture (design view) which represents actual structure of the electrical system. The low-level architecture also shows Logical Components (LC) which are broken down from LACs in the high-level architecture, it also shows connections between LCs, and signals they receive/send.

The low-level architecture is stored in a custom proprietary tool, which we will call it *the Database* in this thesis work. The Database stores the data such as documenta-

tions, the software components, the ECUs, the LCs, and the LACs. Using the stored data, the tool generates model shells to be implemented by software developers for in-house software development and also the tool generates requirement documents which has many functionalities to be implemented by suppliers. Moreover, it can also generate ECU-integration and network communication in the electrical system [11].

When a user interacts with the Database, he/she can search the information using small pop-up window. A user fills in the few details of the needed artifacts, after clicking on the search button, a result is displayed. The output depends on the user's specifications on the search pop-up window. Its limitation is that, you can have different variants handling, different version of the same artifacts allocated in different areas on the Database. As a result, all of this information is shown on a list which is a difficult task to know the right artifact unless you know exactly to where it is allocated. The architecture is enormous which makes it difficult for the stakeholders to get what they need with less time, having experience of using the Database is quite big support in this case, meaning someone must know where to locate a needed artifact otherwise it will take a long time to get the needed information. Thus, visualization of the low-level architecture is needed.

1.1 Statement of the problem

The problem that is addressed in this thesis is the difficulties in getting a needed information on using the search option on the Database which usually gives a user with a lot of information, sometimes unnecessary and not enough for a single search result. This in turn forces a user to keep searching for more information until he/she gets to the root source of what is needed in the first place. In most cases, only some parts of the data stored in the Database are relevant to the stakeholders and this depends on the individual's interest. In some cases, the stakeholders do not even know what they are looking for and this makes a task to locate what they want even more difficult.

1.2 Purpose

The purpose of this thesis is to create an automated visualization prototype of the electrical architecture from the data extracted from the Database using Unified Modeling Language (UML) notations. By referring to the previous research, there are results confirming that visualizing the architecture using the UML notations improves the communication among software engineers [9][12]. This is briefly summarized in section 2.2.

The thesis also aims to identify the needs of the stakeholders, the needs in this context are functional and non-functional requirements towards the automated visualization. The analysis of the needs of stakeholders will allow concluding on the

possible views/visualizations that can show the system at different levels of abstraction for different stakeholders.

In addition to the analysis of the needs of stakeholders and the creation of the automated visualization prototype, we also aim to report on what are the opinions of stakeholders towards the automated visualization of electrical architecture. In this particular context, we aim not only to consider the created automated visualization prototype but also the automated visualization of the whole electrical architecture at VCG.

1.3 Research questions

In order to address the problem mentioned, we have formulated two research questions. Both of these research questions will allow us to fulfill the purpose (Section 1.2) we have set in this thesis. The research questions are formulated as follows:

RQ 1. What are the needs of different stakeholders towards the automated visualization of the electrical architectures?

The first research question covers the identification of the needs of stakeholders towards visualization of the electrical architecture and the differences among them. The information needs concern details such as what are the artifacts (ECUs, LCs, etc.) that stakeholders want to be visualized and what to include and what not to include on the visualized output.

RQ 2. How does an automated visualization fulfil the needs of stakeholders?

The second research question covers how an automated visualization helps to fulfill the information needs of stakeholders. Since this study is newly introduced at VCG, we intend to learn on what the automated visualization of electrical architecture can contribute to both users and non users of the Database, the Database is the one that stores the electrical architecture of VCG (Section 1).

1.4 Contribution of the study

The findings of this thesis work aim to encourage the use of automated visualization, and the way of working with the electrical architecture in the automotive industry. Since it has been newly introduced, our report will provide supportive evidences about the benefits of applying automated visualization of data.

For the company, the data extracted from the Database and the needs that we discovered from interviewing different stakeholders will be a powerful input that will aid to provide a better automated visualization of the electrical architecture, which

in turn will cover the needs of different stakeholders. Our study also provides an opportunity for further research at the company to obtain greater results towards satisfying the needs of different stakeholders and most of all to improve the way of working with the Database. Moreover, with Model-Driven Software Engineering creating our thesis will prove to the stakeholders in the company that creating an automated visualization of the data from the Database is possible.

For us, as Master's students in Software Engineering, we had an opportunity to learn how automotive architects and stakeholders work with the architecture, and understood the needs of different stakeholders. We have also discovered how the automated visualization for the electrical architecture plays an important role in the automotive domain.

1.5 Outline of the report

This report is divided into 8 chapters. Chapter 1 introduces the background of the thesis work, statement of the problem, the purpose, research questions, and the significance of the study. Theory related to architecture in automotive domain, software architecture visualization, Model-driven Software Engineering, and graphical notation of textual description are be discussed in Chapter 2. Chapter 3 presents the scientific methods used in this work, while the implementations of the methods are explained in Chapter 4. The results of the study obtained are presented in Chapter 5. Chapter 6 presents the answers to the research questions mentioned in Section 1.3, and deliverables of the thesis work and open-source projects involved. Validity threats and research ethics are discussed in Chapter 7. The last chapter summarizes the study, our work and sustainability and what could be done in the future work.

2

Theoretical Framework

This chapter introduces theories related to this study, which cover the areas of architecture in automotive domain. The two types of electrical architectures that are used at VCG are also explained. Basic knowledge of Model-Driven Software Engineering is presented, including the difference types of stakeholders.

In this chapter, we have covered several applicable knowledge to our thesis. The thesis is done in automotive field, Section 2.1 introduces what is in this field and most important, since our thesis is based on the system architecture, that explains why we have discussed about the architecture in the automotive domain. The Section 2.1.1 explains the architecture at VCG which is what our automated visualization will be applied. In the Section 2.2, it is where we discuss about the previous research done on system's visualization, one was done on the component based system and the second visualization was about getting a reversed architecture from the source code of the system. The last two sections (Section 2.3 and Section 2.4) in this chapter cover the knowledge that we will apply to visualize the electrical architecture at VCG. This includes the discussion of different concepts in Model-Driven Software Engineering that will be of useful in getting the automated visualization of electrical architecture.

2.1 Architectures in automotive domain

Proper architecture is necessary for designing and building modern vehicles which are mostly driven by electronics and software. In this section, we explain some related works of how an architecture plays an important role in the automotive domain.

Beeck [23] developed a modeling approach for development of software for ECUs at BMW Group, which supports the development of logical and technical architectures (high- and low-level architectures, respectively). The approach was developed based on the notation Unified Modeling Language for Real-Time (UML-RT) aimed at compromising the complexity issue of developing, integrating, and maintaining software-intensive systems in vehicles. The logical architecture model was developed using UML-RT's capsule structure diagrams to represent graphical system view of automotive functions. The technical architectures separated into software and hardware architecture were developed using UML-RT's component diagrams (for software) and deployment diagrams (for hardware).

For the logical architecture, the author created a UML-RT constructs and used them to model the architecture. From the meta-model (Figure 2.1), capsules, ports, protocols, signals, and connectors notations were used for modeling architectural artifacts. Capsules represent functions, while ports and protocols model function interfaces. A port was used to specify a communication point of a capsule. Each port had associated protocol, which contained two sets of signals, export and import. Connectors represented channels between function interfaces.

For the technical architectures, the component notation was used to model software components. Hardware components such as ECUs and sensors were modelled using UML-RT nodes.

The author also reported some problems regarding the use of UML-RT. One of the problems was, the set of UML-RT diagram notations was quite restricted. It was difficult for ECU developers, who were familiar with non-object-oriented notations, to work with. In addition, the two protocols associated with ports did not meet requirements of some ECUs.

This paper gave us some suggestions that UML-RT is not the most suitable notations to be used in our automated visualization since not all developers know and are familiar with every notation.

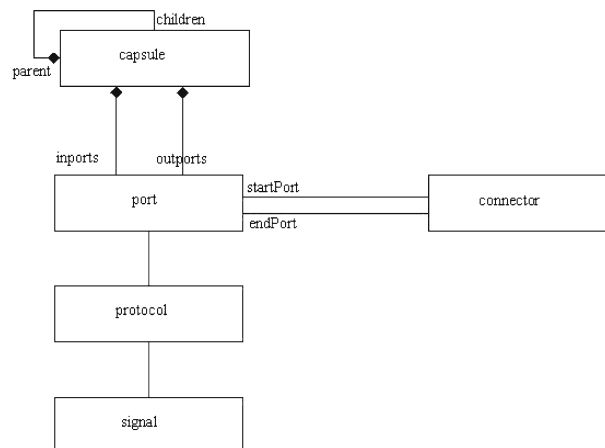


Figure 2.1: Meta-model of UML-RT constructs for logical architecture [23].

Grönniger et al. [14] developed an approach for modelling logical architecture of automotive systems using views (Figure 2.2). Function nets which are valid Systems Modeling Language (SysML) Internal Block Diagrams (IBD) were used to model complete automotive functions and views which described the environment and context of a certain aspect of function net. In addition to that, views could be used to model features in a self-contained way, and to specify consistency conditions for consistency between a view and a function net.

The authors claim that function nets and views could be used to describe and to explain scenarios of use-cases like how an automotive system reacts to external events or failures caused by subsystems. However, numerous models had to be created as well as references between these models. The authors states in their work that an investigation of existing model management strategies to handle number of models would be performed in the future.

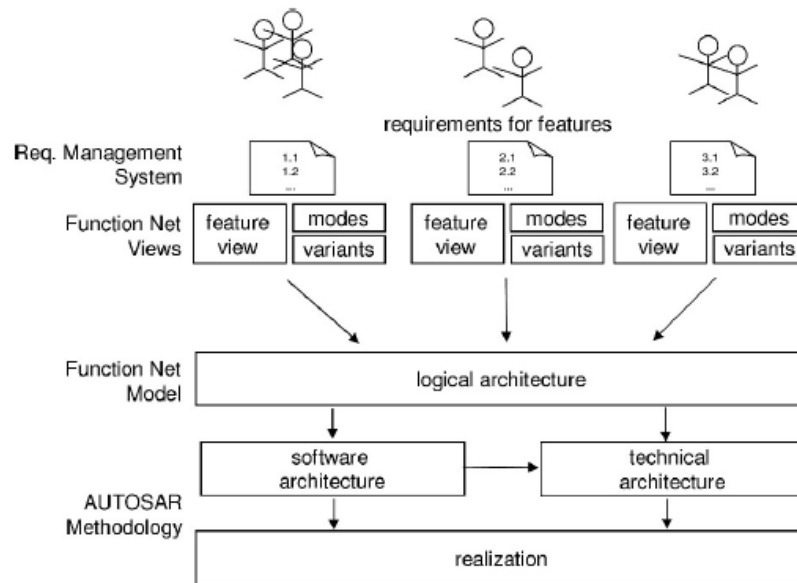


Figure 2.2: Automotive development process [14].

Dajsuren [8] presented a research which was part of Hybrid innovations of Trucks project and it covers the automotive Architecture Description Language (ADL) and quality of automotive software. This research had a role of identifying a proper ways of developing automotive software.

The author suggests that automotive software development enabled interaction between different engineering fields such as mechanical engineering, electrical engineering, and software engineering. ADL was said to be an effective way to manage such multi-disciplinary engineering information. It had been defined on the paper as ‘one of the approach to formalize the representation of the automotive systems and software architecture’. Examples of ADL used in automotive companies are, definition of AML for BMW company, EAST-ADL and TADL for Volvo, Fiat, and VW/Carmeq.

The different levels of the architecture had been mentioned as well, these include:

- *Feature view* that shows the number of features in a system,
- *Function view* that shows the number of functions or subsystems in a system. A single feature could include one or more functions,
- *Software view* that shows a detailed architecture. It shows components and blocks which represent the implementation of the functions specified in the

function view, and

- *Hardware view* that contains ECUs, sensors, actuators and Controller Area Network (CAN).

The author decided to use ADL language SysML¹ in modelling a functional view (Figure 2.3). MATLAB/Simulink had also been mentioned as one of the most popular graphical modelling language and a simulation tool for modelling software view.

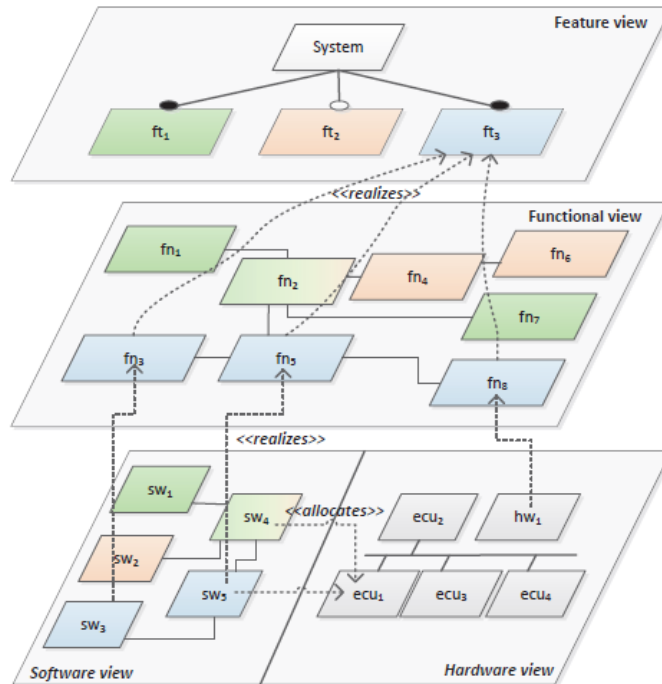


Figure 2.3: Automotive architectural views [8].

2.1.1 Low- and high-level electrical architectures at VCG

To have a better understanding of how low- and high-level electrical architectures have been constructed and used in VCG, we studied some researches that were conducted at the company.

Eliasson et al. [10] found that VCG and Volvo Group Truck Technology (VGTT) have two types of architecture: a high-level architecture and a working architecture (low-level architecture). The high-level architectures produced by high-level architects contain design decision, principles, rules, and pattern that should govern the overall system. The working architecture produced by low-level architects contains logical components which are broken down from the high-level architecture and more details. The study shows that the working architecture is always kept updated by developers as the product evolve, while the high-level architecture is only updated when the project has started. Because of this reason, the inconsistency between

¹<https://sysml.org>

the two architecture occurs. They also suggest that having two different groups of architects result to problems. High-level architects have a thought that the low-level architects are very focused on short-term solutions which makes them miss an overall picture of the system, while low-level architects see that another group lacks an understanding of current situation and is too focused on solutions that might be good in long run.

It has also been presented on the article by Eliasson et al. [11] that the software architecture in VCG is divided in 3 different views, the logical view, the design view and the deployment view. The logical view defines the intended architectural structure of the system, it has the logical architectural components (LACs) which represent a group of functions, LAC to LACs, logical view is in the form of UML model. The design view determines the actual structure of the system, LACs are broken down to LCs and each LC representing a single unit of functionality. The deployment view details how each group of functionality is deployed in different ECUs in the system. The logical view is done by system architecture group. The detailed design, design view is done by engineers in different sub-systems and components and the deployment view is done by both, each take part of it.

Eliasson et al. [11] suggest that the presence of the architecture technical debt at the design level of the VCG plays an important role on the efficiency of communication between components. In this paper, there is discovery of the ATD items (architectural violations) such as the misplaced LCs (logical components) and their impacts on the software development process. The ATD items together with the inputs from the stakeholders at VCG were used to assist in creating a visual tool which provides a better visualization of the ATD items and their interest. With this tool, the visualization is more comprehensive to the stakeholders (see Figure 2.4).

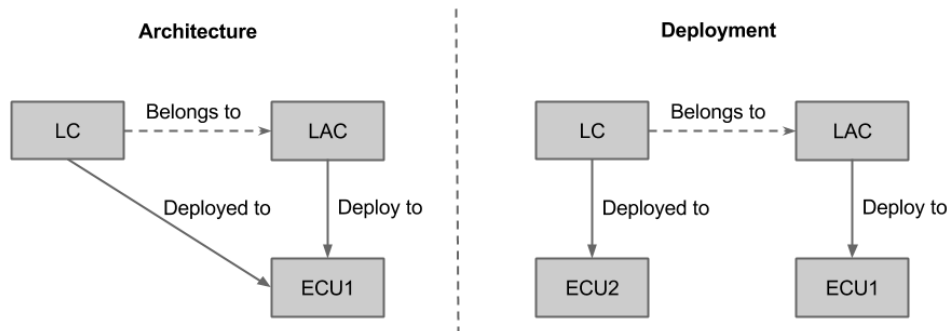


Figure 2.4: The difference between architectural design and actual deployment [11].

Our intention in this thesis is to find different information needs of stakeholders towards automatic visualization and make a prototype that could visualize the electrical architecture at VCG. But before that, we need to know of what makes the architecture, meaning the logical components, the communication between the components such as the inter-ECU communication, intra-ECU communication and the

intra-domain communication. Luckily, the two articles [10] and [11] were useful for that. So generally speaking, the articles somehow provided us with a systematic approach to the system.

2.2 Software Architecture Visualization

The emerging of component based software are said to have a repercussions on software visualization. On visualizing a component-based software product, one has to consider a visualization of component model, software components and of software assemblies. Favre and Cervantes [12] suggest to visualize a component model by describing it as a set of UML class diagram (meta-model). The meta-model will have all the necessary concepts to describe components without getting to the implementation details. The concept of meta-modelling can be considered as a basis in specifying a graphical notation of visualizing components. The graphical notation simplifies the understanding of a meta-model. On visualizing components, a specific notation must be used for each component technology. The authors also mentioned the usefulness of a graphical notation that it allows software engineers to communicate without the burden of speaking in technical terms.

Most software engineers think in low-level programming when designing a component model rather than a conceptual entities. Favre and Cervantes [12] mentioned the useful of having the visualization of the software products to software engineers since they designed the model ‘blindly’, so the visualization would help them to get a complete overview of the implementation. They also elaborated several options that we could apply in visualizing the data stored in the Database. Some challenges had been mentioned in visualizing complex components which might have many ports, so in this case, hiding ports with no connection and show only ports with connection could be of useful when it comes to a visualization process. Another challenge that was mentioned was the visualizing the components with complex connectors.

The visualization of the architecture was also done at Ericsson, where the task was to recover the architecture, it involved getting models from a source code. Darvas and Konnerth [9] mentioned on the advantages of having an automated visualization as it makes the architecture consistent with the current implementation found in a source code. In their work, they mentioned on grouping the ports using cable and port group pattern, it was the technique to merge the connectors to a single connector. The use of abstraction pattern could be useful in our case as the single component can have many ports and hence make the diagram not readable. In our case, we may be needed to find the corresponding metrics that will help to group the ports based on some similarities. Another thing that was mentioned in their work was the way of preserving the hidden information, this implies that once the artifacts are grouped, there will not be a way to see what is inside the grouped artifacts, so having a comment (text) next to the grouped artifacts can be useful to briefly describe what is inside of them.

2.3 Model-Driven Software Engineering

In creating a visualization of the electrical architectures, we applied Model-Driven Software Engineering methodology to our work. This section aims at giving the readers basic knowledge of the method.

In software development, models are used to depict software artifacts in software engineering activities throughout software development life cycle. A model itself is used as primary artifact representing more abstract view of a software to be built. In this context, the methodology is known as *Model-Driven Software Engineering* (MDSE), aiming at tackling the complexity problem caused by the large size of a software due to the needs of humans [2]. The goals of MDSE also include increasing the software development speed which can be done by transformations (Chapter 2.3.2), reducing cost in long-term, and supporting the reuse of model for repeatable processes.

Based on the book ‘*Allgemeine Modelltheorie (General Model Theory)*’ written by Stachowiak [20]², he describes that a model should have 3 fundamental properties: *reduction*, *mapping*, and *pragmatic*. Reduction property of a model is that it contains only details relevant to model creators and users. Generally speaking, the model does not include all details of its original. The mapping property means a model is always the model of something else i.e. its original. The pragmatic property of a model is the model can replace its original with respect to some purpose.

2.3.1 Modeling languages

Models and transformations need to be defined in some notation which in MDSE context it is called *modeling languages*. A modeling language is a tool that designers use for specifying definition of the concrete representation of a model for a software system [2]. It is comprised of three main ingredients: *abstract syntax*, *concrete syntax*, and *semantics* (see Figure 2.5).

A modeling language may consist of graphical representations, textual representations, or both. Modeling languages can be classified into two main categories: Domain-Specific Language (DSL) and General-Purpose Modeling Language (GPML, GML, or GPL). DSLs are the languages that are designed for specific domain, context, or company. The purpose of this type of languages is to help people to describe and explain things in a certain domain. In contrast, GPLs are the languages that are designed for general use. They lack specific features for a specific domain. An example of this type of languages is UML.

² <https://modelpractice.wordpress.com/2012/07/04/model-stachowiak/> (english translation)

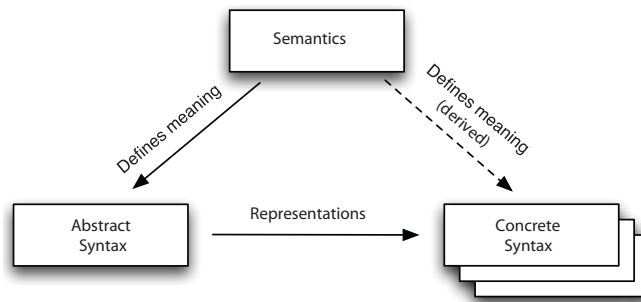


Figure 2.5: Three main ingredients that comprise a modeling language [2].

2.3.1.1 Meta-models, model instances, and semantics

As introduced in Section 2.3.1, a modeling language is comprised of abstract syntax, concrete syntax, and semantics (see Figure 2.5). An abstract syntax is defined using *meta-models* [2]. A meta-model is a precise definition of the parts and rules needed to create valid models [22], so to speak a type of model used to describe the model.

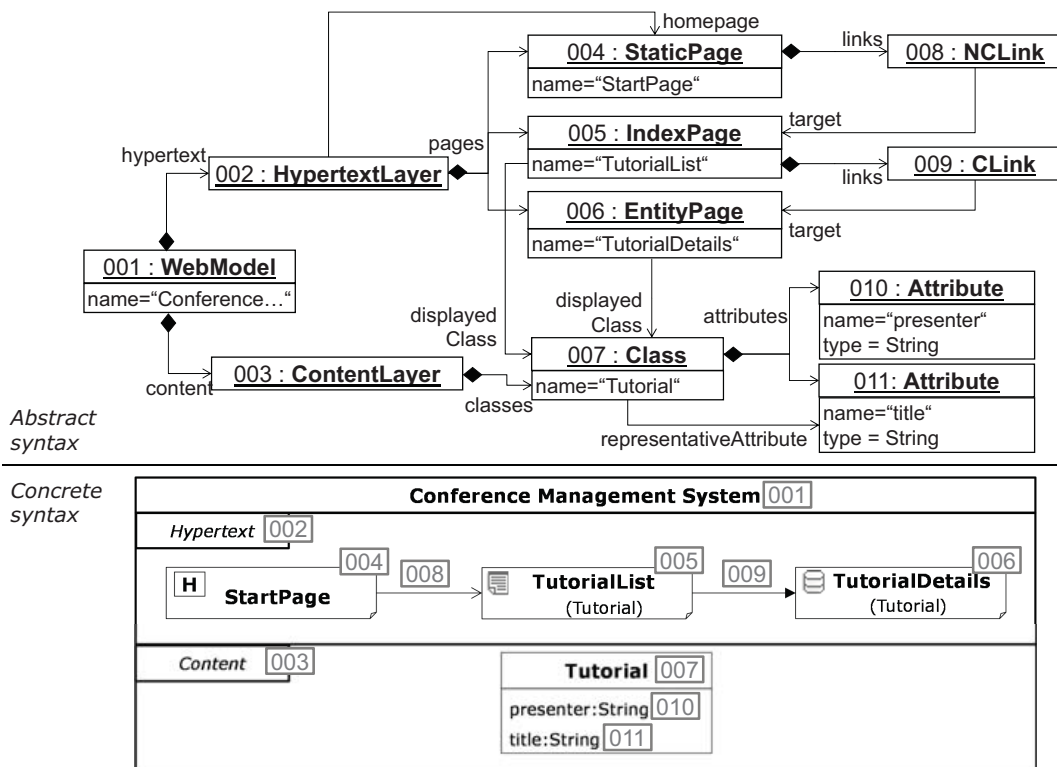


Figure 2.6: sWML model’s abstract syntax and graphical concrete syntax [2].

A concrete syntax is the concrete notation of a modeling language. It can be either a graphical or textual representation of *model instances*. The Figure 2.6 shows the abstract syntax and graphical concrete syntax of one of the well-known DSLs, simple Web Modeling Language (sWML).

The third ingredient of a modeling language is semantics. They define the meaning of abstract syntax and concrete syntax (indirectly). In software engineering, semantics are classified into two main categories: *static semantics* and *dynamic semantics*. The former specifies the allowed structure in a modeling language such as well-formedness and typing of meta-models, while the latter describes the execution behavior or run-time effect of a model [21].

2.3.2 Model transformation

Model transformation is another core concept of MDSE. It allows the mappings between different models. One example is transforming UML class diagram³ to Entity-relationship (ER) diagram⁴.

There are two kinds of model transformation, model-to-model (M2M) and model-to-text (M2T) transformation. M2M transformation takes in a source model as the input and the output is named a target model. M2T transformation takes in a source model as the input and the output is code(text). The model transformation can be done by using template-based approach or visitor-based approach [7]. In this thesis, we were mostly interested with M2T transformation and so we applied template-based approach in order to do a M2T transformation. The example of M2T transformation can be seen in Figure 2.7.

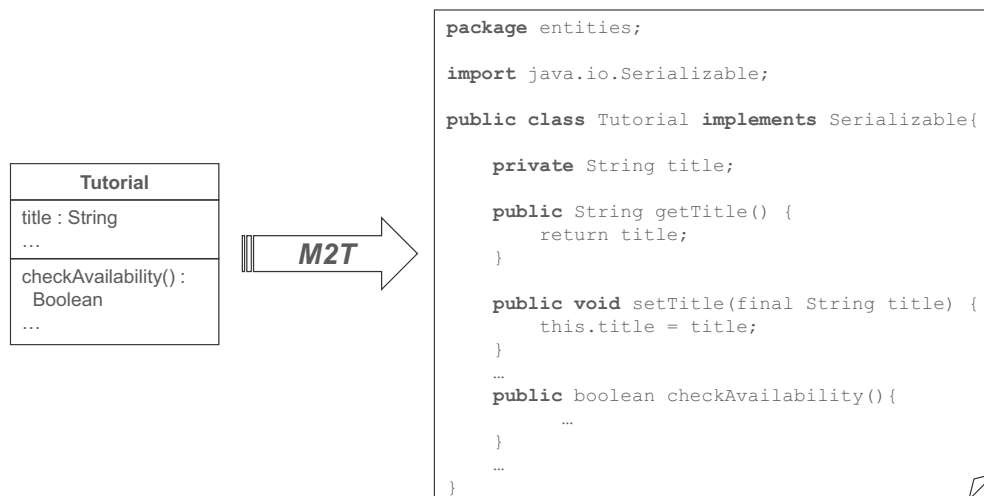


Figure 2.7: An example of M2T transformation [2].

We have applied M2T transformation because it increases the analyzability of the generated output(code) as one can go through line by line which in turn allows early error detection in the code. As mentioned already, M2T transformation is done by applying template based approach, another reason why we chose M2T transformation is that the template based approach allows an easy description of variables that

³A static structure diagram that represents the structure of a system showing classes, attributes, methods, and the relations among objects.

⁴A data model that describes data in business aspect.

depend on the model and the variables that does not depend on the model. The example of variables can be observed in Appendix A and Appendix B where the dependent variable of the models are subsystem, hasLC, hasPort, etc and also the variables such as the ones for styling on the visualized diagram are not dependent on the model.

To do a M2T transformation, a number of template engines are available. The template engines are capable of generating files with texts of different formats depending on how the generator file has been programmed. The content of a generated file(s) depends on the meta-model specified and also the model instance of a meta-model, this will be discussed in details in Section 4.1.4.2.

We used *Acceleo*⁵ as the template engine. It is an open source software and also available as a plugin in eclipse software. We have used Acceleo because of its ability to give a generated text from different sources such as UML models, ecore models and most of all a custom made meta-model which is something we have done in this thesis. The structure of an Acceleo template (see Listing 2.1) is composed of module, template, main, and generating files. Module in Line 1 is the part where Uniform Resource Identifiers (URIs) of the meta-models instantiating the models that you want to generate code from are parameterized. Template in Line 2 is where the name of the template and its parameters are specified. The parameters are declared in the convention `<name>:<type>`, where name is the parameter name belonging to type which is provided by meta-model. The code in Line 3 indicates the entry point of the generation. Line 4 is for specifying the name of generated file. The code written after that will be generate textual description.

```
1 [module moduleName('http://www.eclipse.org/emf/2002/Ecore' )/]
2 [template public genMyTemplate(aParam: EClass)]
3 [comment @main/]
4 [file ('filename.txt', false, 'UTF-8')]
5 ....
6 [/template]
```

Listing 2.1: An example of Acceleo template

2.4 Graphical notation of textual description

The output from M2T transformation is textual description. It is used for generating visualization based on a model instance, which conforms to its meta-model. To create a visualization of the electrical architectures, several tools such as PlantUML⁶ and Umple⁷ can be used.

⁵<https://eclipse.org/acceleo/>

⁶<http://plantuml.com/>

⁷<http://cruise.eecs.uottawa.ca/umple/>

In this thesis, we use the open-source PlantUML as a graph visualization tool. The language of PlantUML is well-formed and human-readable code from which the diagrams are rendered. The tool allows users to create UML diagrams from textual description written in its DSL and we would like to create an automated visualization using the standard UML notations.

We chose PlantUML because of its richness in symbolic expressions which are easier to understand, apart from that, there are many examples on their website which are easily applicable and easy to understand. The availability of a plugin in eclipse makes it better tool to use since you only need to set small configuration for you to see the graphical view of the transformed text.

```

1 @startuml
2
3 class Accommodation {
4   +Int Floor
5   +Int Wall
6   +Int Ceiling
7   +Int Door
8   +Int Window
9 }
10
11 class Apartment
12 class House
13
14 Accommodation <|-- Apartment : Inheritance
15 Accommodation <|-- House : Inheritance
16
17 @enduml

```

Listing 2.2: An example of textual description of a class diagram

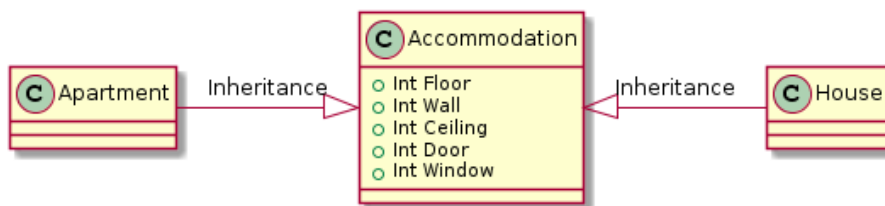


Figure 2.8: UML class diagram rendered from the textual description in Listing 2.2.

Different types of UML diagrams such as class diagrams can be generated. Figure 2.8 shows the UML class diagram rendered from the code in Listing 2.2. There are other different types of representations that can be applied in visualizing the data but instead we picked UML diagrams. This is due to its richness in visual elements which are easy to follow. UML diagrams also provide a standard for the software development and its widely used in both academic and industrial domain.

UML component diagrams can also be created from the textual description by using some specific syntax. Table 2.1 shows the examples of how to use PlantUML syntax. The use of them will be explained in Chapter 4.


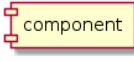
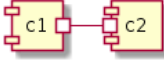



| Description | PlantUML syntax | Representation |
|---|--|---|
| Package | <code>package Package</code> |  |
| Component | <code>[component]</code> |  |
| Components, ports, and connection | <code>[c1] #-# [c2]</code> |  |
| Components, and ports with required/provided interfaces | <code>[c1] #- (0-# [c2]</code> |  |
| Rectangle | <code>rectangle Rectangle</code> |  |
| Component, required interface, and rectangle | <code>rectangle Rectangle [c1] -(Rectangle</code> |  |

Table 2.1: Some useful syntax for UML component diagrams.

With the combination and modification of the syntax, visualizing electrical architectures is possible.

2.5 Stakeholders

On referring to the article of Community Tool Box [4], there are three different kinds of stakeholders: primary stakeholders, secondary stakeholders and key stakeholders. On the later part of the study, we will interview different stakeholders. Some of the advantages of interviewing the stakeholders include getting more ideas, obtaining different perspectives, helping to avoid misunderstanding of the problem, and also increasing the chances of delivering a valuable output to the stakeholders.

The primary stakeholders in our case are the people who are directly interacting with the Database, also named beneficiaries and we, the students are the target. The automated visualization prototype can aid the beneficiaries in obtaining a simplified overview of a data found on the Database which in turn can provide a quick understanding of the needed artifacts and the interactions between them.

The secondary stakeholders are the ones who are indirectly affected by the Database. They are the ones who may be affected by the decision made by the stakeholders who interacts directly with the Database, which in our case will be due to the use of the obtained visualization. Our supervisors at the university can also be in a group of secondary stakeholders, they are not directly interacting with the obtained visualization but they also play a greater role in pushing forward our goals for delivering

a good visualization.

The key stakeholder is the person from the company and this is one of our supervisors. He plays a very important role in bringing efforts that could result to a better solution of the problem that we intend to resolve.

We, the students and as part of the primary stakeholders (targets), intend to provide a visualization output that could aid in quick understanding of the data stored on the Database and also provides the opportunity for further research depending on the final output that we will get in this report.

2.5.1 Needs of stakeholders

The needs of stakeholders can be varied and in different areas [4]. In this thesis we defined the ‘needs’ of stakeholders as functional and non-functional requirements towards automated visualization. Functional requirements cover the needs of stakeholders with respect to the information that they want to see as well as additional features in automated visualization. The information, in this context, is the data from the electrical architecture such as the artifacts (ECUs, LCs, etc.), details of signals (source and destination), and signal bus types. Non-functional requirements cover the needs in regards to software quality aspects such as usability and performance of the automated visualization. The classification of the needs of stakeholder will be explained in details later in Section 4.3.2.

3

Methodology

This study was conducted using Action Research (AR) methodology which is one of the common primary approaches to research in software engineering. Since our study aims at providing an automated visualization of the electrical architecture for VCG, AR is an appropriate method as it emphasizes on providing practical value to an organization while contributing to acquisition of new theoretical knowledge [19].

The methodology consists of four basic steps [13] which are focus selection, data collection, data analysis and interpretation, and take action (shown in Figure 3.1). The general concept of the steps of AR methodology and how it was applied to this study are introduced in this chapter, while the implementation of the approach is presented in Chapter 4.

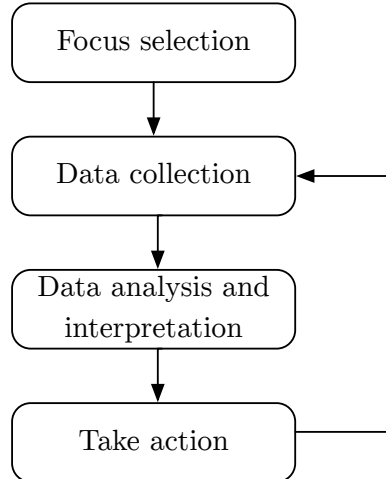


Figure 3.1: Steps in action research.

AR is cyclical approach, meaning that the process does not have to be terminated after the fourth step. In this thesis, we performed two cycles as shown in Figure 3.2. The reasons why we decided to perform only two cycles are: firstly, the accessibility of the data since we did not have full access to the Database, and secondly, due to time constraints. We started with selecting a focus and identifying a scope of visualization with the first stakeholder, a software engineer (see Table 3.2). Then, we collected the first set of data from the Database and analyzed them. We took action by developing a prototype of automated visualization and presented it to the

stakeholder in order to get some feedback for the improvement.

Once we finalized the automated visualization prototype, we began the new half cycle. We collected the second set of data, by conducting semi-structured interviews with other stakeholders who use the Database to work on their daily assignments. We recorded conversations during the interview sessions. The second data analysis was done by transcribing the tape records, and the transcripts were analyzed using a scientific method for qualitative data analysis called *Coding* [17] (will be described in Section 3.3.3). In the final part of the work, the results from the coding process were used for answering the research questions of this study. As it was mentioned earlier that due to unlimited amount of time and resources, we stopped in the step of data analysis and interpretation, the list of prioritized actions to be taken in the take action step of AR methodology have been presented in Section 8.2.

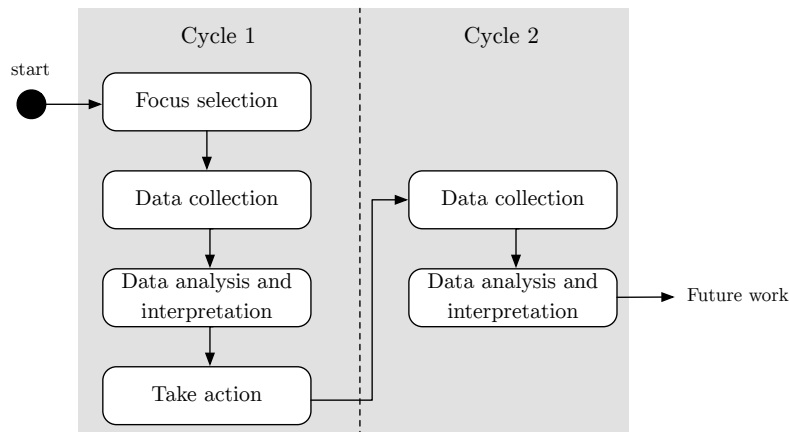


Figure 3.2: Steps performed in the study.

The implementation of the two cycles is presented in Section 4.1 and Section 4.3, respectively.

3.1 Focus selection

In order to identify the scope to visualize the Database, we applied the elicitation technique known as stakeholder analysis described by Lauesen [15]. In figure 3.3, taken from the book by Lauesen [15] (Figure 8.2 : Elicitation techniques), it can be seen that the technique is strongly applicable in a situation where the researcher wants to elicit on the goals and key issues, also to find out about the present problems.

| Techniques | Things to elicit | | | | | | | | | |
|----------------------|------------------|--|---------------------|-------------------------|----------------------|------------|---------------------|--------------|------------|--------------|
| | Present work | Present problems Goals and key issues | Future system ideas | Realistic possibilities | Consequences & risks | Commitment | Conflict resolution | Requirements | Priorities | Completeness |
| Stakeholder analysis | | | | | | | | | | |
| (Group) interview | | | | | | | | | | |
| Observation | | | | | | | | | | |
| Task demo | | | | | | | | | | |

Figure 3.3: Elicitation technique of stakeholder analysis [15].

In our case, we applied the technique on this first step of the AR methodology by conducting meetings with the stakeholder (Table 3.1).

| Stakeholder | Role | Cycle & step |
|-------------------|---------------------------|--------------------------|
| Software engineer | Develop software for ECUs | Cycle 1: focus selection |

Table 3.1: Stakeholder who participated in cycle 1.

The stakeholder was chosen by our supervisor at VCG due to the reason that he had been working with the Database for a long time and he had a good suggestion on which scope of the Database we would start to visualize. The first meeting was purposely to understand the problem at the company that we can address and identifying the scope to visualize. The second meeting was mainly to validate the first visualization (Figure 5.1) and also to get contacts of other stakeholders that we wanted to interview for the cycle 2 of the AR methodology. Through these meetings, we were also able to get a clear understanding of the Database. The technique had also helped us to come up with the scope to visualize and also helped us largely to understand the concepts used in automotive fields.

3.2 Data collection

Data used in this study were collected from two main sources, from the Database, and from interviewing stakeholders. This section explains how the data was collected from the data sources. The first part discusses the raw JSON data and gives an example of a JSON object structure. On the second part of this section, we introduced the applied approach to collect a meaningful data from interviewing the stakeholders.

3.2.1 Raw JSON data

To create an automated visualization, we extracted data from the Database. The best way that was suggested to retrieve data from the Database was to use an application program interface (API). At VCG, we were given a documentation that had a list of APIs and the description of how to use the APIs in order to get the data from the Database. In order to access the data, one has to be within the VCG network. Since we were not granted access, then we did most of the work outside the company area and that had forced us to use the censored data file. The use of API is more dynamic than visualizing a static file. However, this does not make the step to retrieve data from the Database less meaningful, it is still the same data but later on, a static file will need to be replaced with a specific API to get the data from the Database.

The data found in the static file retrieved from the Database was in JSON format. A simple example of JSON object is shown in the Listing 3.1 below:

```
1 {
2   "id": 1,
3   "name": "Master Thesis report",
4   "year": 2016,
5   "place" : "Sweden",
6   "examiner" : "Eric",
7   "Supervisors": ["Truong", "Ulf", "Michel", "Patrizio"]
8 }
```

Listing 3.1: An example of JSON object

3.2.2 Interview data

We conducted semi-structured interviews to collect data from stakeholders. This type of interview allowed us to prepare pre-determined and open-ended questions relevant to our study and the research questions. It allows interviews the freedom to express their views and opinions in their own ways. The list of stakeholders that we have interviewed, their roles at VCG, and also which step they took part in this study can be seen in Table 3.2.

| Stakeholder | Role | Cycle & step |
|--------------------|----------------------------|--------------------------|
| Test engineer | Involve in testing process | Cycle 2: data collection |
| Software developer | Develop functions for CEM | Cycle 2: data collection |
| System designer | Write system requirements | Cycle 2: data collection |

Table 3.2: Stakeholders who participated in cycle 2.

The stakeholders above were chosen by the software engineer that we had a meeting with in the cycle 1. His criteria of choosing was based on the roles of the stakeholders which cover the three main phases in software development process, which are

design, development, and testing phases.

To collect data effectively, we applied a goal-oriented data collection methodology developed by Basili [1]. We started off by setting the goals of data collection in order to find a pattern for collecting data. After establishing the goals, we created a list of pre-determined questions and categorized them based on the goals (see appendix B). The stakeholders were asked these questions in the interviews and a voice recorder was used to record all conversations. Then, we analyzed the collected data to extract essential pieces of information for answering the research questions.

3.3 Data analysis and interpretation

In this section, we will discuss the approach applied to analyze and interpret the collected data in both cycles of the applied AR methodology. The first part of the section covers the applied steps to analyse the data from the Database in cycle 1. The second part of the section explains the approach we applied on analysing the data from the interviews which is part of cycle 2.

3.3.1 Meta-model of JSON data

The retrieved JSON file from the Database was in the format of JSON and so we needed to identify what were the objects in it. To understand the raw JSON data from the Database, we firstly created a meta-model of the data in order to see the schema of the file which specifies the relationship among the artifacts.

We have used Eclipse¹ software in both this step of analyzing and interpreting the data and also in the take action step of the cycle 1. One can create manually a meta-model and a model instance by using Eclipse Modeling Framework (EMF) with an Eclipse plug-in called *EcoreTools*². EcoreTools is a complete environment including a Graphical Ecore Editor (Figure 3.4) for creating meta-models (Ecore) and model instances.

An alternative way of creating a meta-model and a model instance is to use *JSON discoverer*³. JSON discoverer is an open-source project developed by Javier Luis Cánovas Izquierdo and Jordi Cabot. It provides a feature that automatically discovers the implicit schema (meta-model) and a data model (model instance) of a JavaScript Object Notation (JSON) document.

Based on the paper written by the tool creators Javier Luis Cánovas Izquierdo and Jordi Cabot [3], the discovering process is a model-based process which is composed of three phases: pre-discovery phase, single-service discovery phase, and multi-service discovery phase. The first phase aims to extract the low-level a JSON model out of a JSON document. The second phase is to obtain the schema information of

¹<https://eclipse.org/>

²<http://www.eclipse.org/ecoretools/>

³<http://som-research.uoc.edu/tools/jsonDiscoverer/>

a JSON document. The last phase aims at obtaining common schema of more than one JSON documents. Since the data of the sub-system Visibility Control SPA that we extracted from the Database was in one single JSON document, multi-service discovery phase was not part of the work.

In our case, choosing JSON discoverer tool has been a good practice since data in the Database was extracted in JSON format. To create a meta-model and the model instance, we simply imported the JSON file as an input to the tool. The analysis of the obtained raw JSON file has been discussed in Section 4.1.3.1.

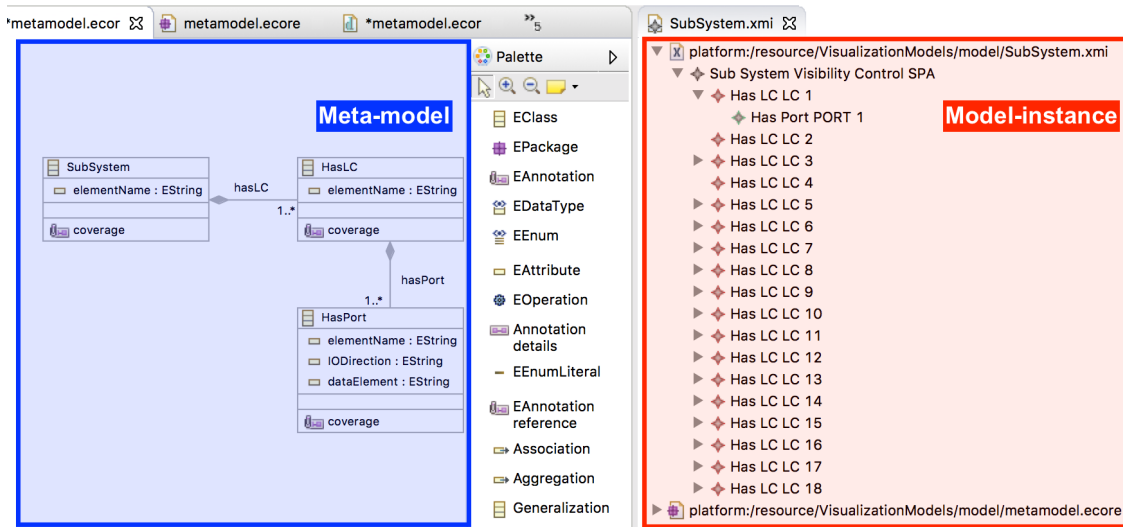


Figure 3.4: A screenshot of Graphical Ecore Editor in Eclipse. The blue area (left) is the editor where meta-model can be created. The red area (right) is the editor for creating model-instance.

3.3.2 Optimization of JSON data

The data retrieved from the Database had so many information that were somehow not needed in the visualization. The first step we did was to find a way to omit the data that was not needed and keep the one that we only needed to visualize. The optimized JSON content was applied again to the JSON discover to get the meta-model and the model instance. The detailed explanation of how this was performed is covered in Section 4.1.3.2.

3.3.3 Coding

Apart from the raw JSON data extracted from the Database which was done in cycle 1, another set of data was collected from interviewing stakeholders in cycle 2. We started the analysis of the data by listening to the tape records twice in order to understand the conversations and to get some key points that the interviewee emphasized during the interviews. The process continued by transcribing the tape records word by word manually. During this analysis, we would have applied voice recognition software but we decided not to use any due to the fact that the software

has to be trained for each voice, and it requires speakers to speak slowly making the interview inefficient.

The transcript files were then analyzed using coding method. To give a basic description what coding is, it is one of the methods used in qualitative data analysis especially the analysis of interview transcripts. It is the process of capturing essential words or phrases from set of data that give the same ideas, themes, and categories [17]. Before starting the coding process, a list of codes were created based on the goals of the pre-determined questions presented in Section 4.2.2. Once we had the list of codes, each of us began coding the transcripts files. Together, we then compared the results from coding to summarize the coded data. The list of codes and the results are presented in Section 4.3.2.1 and Section 5.2, respectively.

3.4 Take action

After the data collection in cycle 1, we developed an automated visualization from the data extracted from the Database using the concept in MDSE described in 2.3. A meta-model and model-instance were constructed. An Aceleo template was created for a M2T transformation. A textual description generated from the template was the input of PlantUML to create an automated visualization. The implementation of this step has been elaborated in Section 4.1.4.

Due to the issue of time and the workload of the report, the take action step of cycle 2 is reported on the Section 8.2, with the list of prioritized stakeholders needs which have also been discussed on Section 6.1.

4

Implementation

This thesis work was conducted using AR research methodology introduced in Chapter 3. This section presents how steps of the methodology were performed in each cycle as shown in Figure 4.1, starting with the steps in cycle 1 which are focus section, data collection, data analysis and interpretation, and take action. Actions preliminary to cycle 2 consisting of how we selected stakeholders and prepared for interviews are explained in Section 4.2. The implementation of cycle 2 is then presented in Section 4.3.

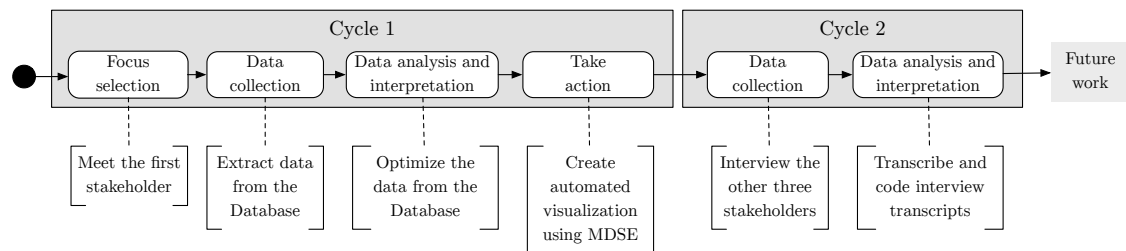


Figure 4.1: Overview of the steps performed in this study.

4.1 Cycle 1: Creating automated visualization

This section explains the first four steps shown in Figure 4.1. The first step is about identifying the scope of visualization which is being covered in section 4.1.1. In the section 4.1.2, we discuss a bit about the implementation of data collection, the section 4.1.3 covers implementation of data analysis and interpretation and the last step of the cycle 1 is presented in section 4.1.4.

4.1.1 Focus selection

At VCG, the Database is being used for storing data of the vehicles such as requirement documents, software components, ECUs, LCs, and LACs. These data are structured similar to directory structure¹ of an operating system, comprising folders and sub-folders, also shown in figure 4.2.

To understand how the Database has been used in development process, we arranged a meeting with a software developer who works with one of the biggest and impor-

¹The organization of files into hierarchy of folders.

tant ECUs within a car, which we call it *the Node* in this report. In the meeting, he explained that searching for an artifact in the Database was a big challenge because of the lack of visualization features in the tool. One of the examples that he gave us to explain the challenge was, considering a problem with LC where a fault signal from a port of one of its associated LCs was sent to. To solve the problem, the in-house developers responsible for this task must look for the port using a search text-area field in the Database. The tool then returned a list of more than 20 ports of the associated LCs, which the developers will need to check them one by one unless if they already knew by experience which port they should look for. Thus, having a visualization of the signals sent/received among LCs would be an advantage in this aspect.

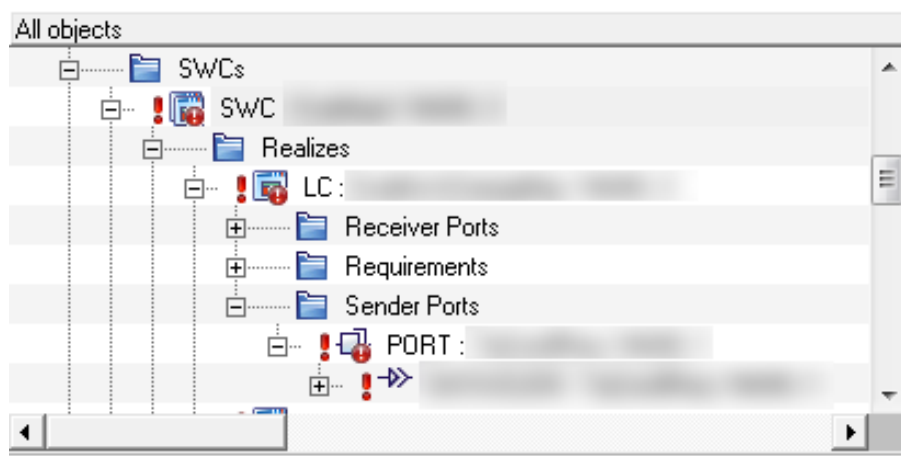


Figure 4.2: A screenshot of the Database where the data is stored hierarchically.

The discussion with the software engineer was for approximately two hours and it was then we figured that we needed to visualize the logical view in the Database which will include LCs, ports and data elements. He also proposed to visualize the physical view in the Database which would include the ECUs.

The relationship between ECU and LC can be elaborated as follows : first of all, an ECU can have one to many software compositions (SWCs). The SWC is simply a group of LCs. The LC can have zero to many ports and each port has one data element, it is the data element that connects one LC to another through ports. For example, LC 1 with PORT 1 can be linked to LC 2 with PORT 2 only if PORT 1 and PORT 2 have the same data element.

In the discussion with the software engineer, we decided to visualize a sub-system. An example of a simple sub-system can be observed from the Figure 4.3 on the top left, there is a small box named SUBSYSTEM and inside of it, there are two ECUs, one ECU has two LCs and another ECU has one LC. The LCs are connected together via ports. The LCs inside the sub-system are also connected to other LCs outside the sub-system but our scope is only for LCs inside a single sub-system. For this report, we visualized a specific sub-system which we gave it a name

Visibility Control, this sub-system has 18 LCs and 179 ports at the moment.

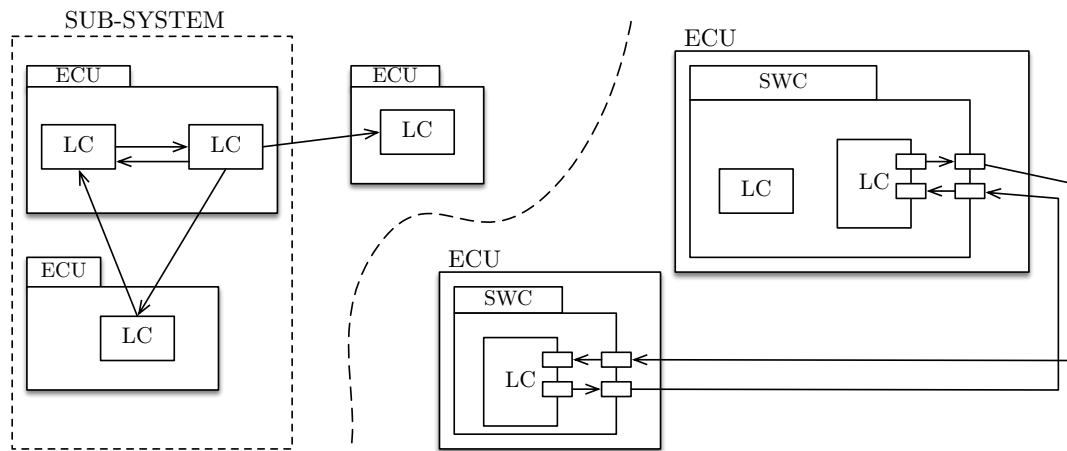


Figure 4.3: An example of a visualization given by the software engineer.

4.1.2 Data collection

The data of the sub-system *Visibility Control* was extracted from the Database with help from our industrial supervisor. The data were also censored due to security policy of the company and we had no access to the Database by all means.

4.1.3 Data analysis and interpretation

In this section, we started by analysing the raw JSON data, the process is explained in section 4.1.3.1. The next part of this section covers the step we did on optimizing the raw JSON data, it is presented in section 4.1.3.2.

4.1.3.1 Analyzing raw JSON data

On this sub-section, we had to understand the meaning of the JSON objects found on the retrieved data and their relations with one another. The static file had approximately 45 thousands line of code. The beginning of the static file that was retrieved from the Database is shown in the Listing 4.1 below:

```

1 {
2   "elementName": "Visibility Control",
3   "name": "Visibility Control",
4   "state": "In work",
5   "classType": "GSUBSYSTEM",
6   "variant": "MAIN",
7   "domain": "VCC_EEDM",
8   "creationUser": "Anonymous",
9   "className": "SUBSYSTEM",
10  "contentRelations": [
11    {
12      "destination": {

```

4. Implementation

```
13         "elementName": "LTC 1",
14         "name": "Name for LTC 1",
15         "classType": "GLTC",
16         "variant": "MAIN",
17         "domain": "VCC_EEDM",
18         "creationUser": "Anonymous",
19         "className": "LTC",
20         "state": "Frozen",
21         "modificationUser": "Anonymous",
22         "modificationDate": "2013-06-25T05:13:36+0000",
23         "attributes": {
24             "MaximumLatency": "250",
25             "Access control": "Access allowed",
26             "NominalLatency": "-1",
27             "RefinedConstraints": [],
28             .....
29         },
30         "creationDate": "2013-05-02T11:48:25+0000",
31         "id": "1937212",
32         "elementId": "1169278",
33         "revision": 0
34     },
35     "contentAttributes": {}
36 },
37 .....
38 }
```

Listing 4.1: The section found in the beginning of the visualized static file

From the Listing 4.1 shown above, the variable `elementName` that appears in Line 1 of the file specifies the name of the sub-system that is visualized which has the value `Visibility Control`. The variable `className` that appears in Line 9 determines the class of the static file that is visualized, it has the value `SUBSYSTEM`.

In order to explain the relationships between JSON objects found in the retrieved file, we created a meta-model for the purpose of explaining the contents of the file, see Figure 4.4.

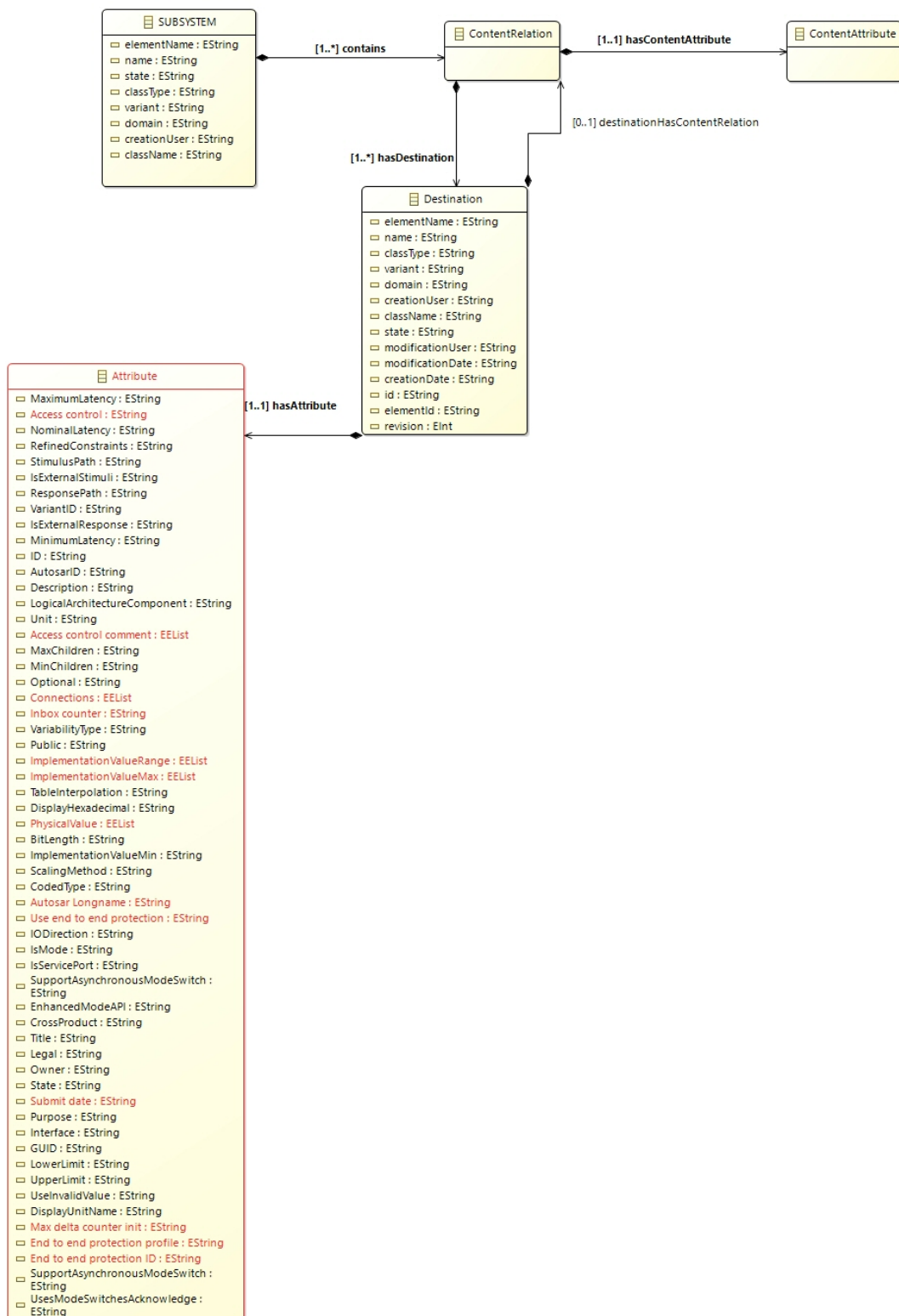


Figure 4.4: Meta-model of the raw JSON document to explain the content of the retrieved data from the Database.

4. Implementation

The meta-model shown in the Figure 4.4 has sum up most of the variables found in the static file. The classes that we needed the most are the `contentRelations`, `destination` and `attributes`. It can be noticed that the class `contentRelations` has an association of one to many with the class `destination` and also the class `destination` has an association of zero to one with the class `contentRelations`. Also, the class `destination` has an association of one to one with the class `attribute`.

The objects of the class `destination` could be LC, port, data-element, data-type, REQSET, etc. In order to elaborate this, we will give an explanation of how it appears in the JSON file :

1. a LC which is a JSON object of a class `destination` has a JSON object of a class `contentRelations`, let's name it CR1,
2. a JSON object CR1 can have an array of JSON objects of a class `destinations`, a port being one of the JSON object of a class `destination` in that array,
3. a port has a JSON object of a class `contentRelations`, let's name it CR2,
4. a JSON object CR2 can have an array of JSON objects of a class `destinations`, a data-element being one of the JSON object of a class `destination` in that array,
5. a data-element has a JSON object of a class `contentRelations`, let's name it CR3,
6. a JSON object CR3 can have an array of JSON objects of a class `destinations`, a data-type being one of the object of a class `destination`, etc.

The Listing 4.2 below presents the LC 1, its port and its data element, some data have been omitted to allow the possibilities to see how the LC, port, data-element and data-type are related:

```
1 {
2   "destination": {
3     "elementName": "LC 1",
4     "name": "Name for LC 1",
5     .....,
6     "contentRelations": [
7       {
8         "destination": {
9           "elementName": "PORT 1",
10          "name": "Name for PORT 1",
11          .....,
12          "contentRelations": [
13            {
14              "destination": {
15                "elementName": "DATA-ELEM 1",
16                "name": "Name for DATA-ELEM 1",
17                ....
18                "contentRelations": [
19                  {
```



```

20         "destination": {
21         "elementName": "DATA-TYPE 1",
22         "name": "Name for DATA-TYPE 1",
                .....
23     "attributes": {
24         "IODirection": "REQUIRE",
25         ...
26 }\\

```

Listing 4.2: A sample part of a JSON file showing the relationship between LC, port and data-element

A JSON object with the name `attributes` can be noticed from the Listing 4.2 at line 145, inside this object, there is a variable with the name `IODirection`. The variable `IODirection` determines whether a port provides to another port or a port requires another port. The principle in this context of `require` and `provide` is that it explains about a dependency between ports and when it comes to visualize the connection between LCs, we took a look at this variable to determine on how to map the ports in a visualized component diagram.

4.1.3.2 Optimizing raw JSON data

As it can be observed in the listing 4.1, the raw JSON data had lots of information that was necessary to be filtered. What we did was to write an algorithm that is able to read the original JSON data and get only the data that we used in the visualization. The optimized data included the names of LCs, the ports for each LCs, the name of the ports, the `IODirection` of the port which determines the connection between the ports and the data-element of the ports.

To explain how the algorithm works, we will write down the steps that we followed:

1. Read from a original JSON file
2. Get the name of the sub-system
3. Introduce loop 1 that gets all the LCs of the sub-system.
4. Introduce loop 2 inside loop 1 that gets all the ports of a single LC
5. Get the name of the port
6. Get the name of the data-element
7. Get the value of `IODirection` of the port
8. End of the loop 2, End of loop 1
9. Write an optimized content to a new file

The result of the algorithm that does a JSON optimization appears in the listing4.3 below :

```

1 {
2   "elementName": "Visibility Control",
3   "hasLC": [
4     {
5       "elementName": "LC 1",
6       "hasPort": [
7         {

```

```
8         "elementName": "PORT 1",
9         "IODirection": "REQUIRE",
10        "dataElement": "DATA-ELEM 1"
11    }
12  ]
13 },
14 {
15   "elementName": "LC 2",
16   "hasPort": []
17 },
18 {
19   "elementName": "LC 3",
20   "hasPort": [
21     {
22       "elementName": "PORT 2",
23       "IODirection": "REQUIRE",
24       "dataElement": "DATA-ELEM 2"
25     },
26     {
27       "elementName": "PORT 3",
28       "IODirection": "REQUIRE",
29       "dataElement": "DATA-ELEM 3"
30     }
31   ]
32 }
```

Listing 4.3: A sample part of the optimized JSON content

4.1.4 Take action

This is the last step of the cycle 1, it has three sections. In Section 4.1.4.1, we explained on the steps we came up with meta-model and the model instance. The first automated visualization prototype is discussed in Section 4.1.4.2 and the final automated visualization prototype is discussed in Section 4.1.4.3.

4.1.4.1 New meta-model and model instance

Once the raw JSON data is optimized (Listing 4.3), we created a meta-model and a model instance using the open-source tool JSON discoverer. With the optimized version of the raw JSON file, JSON discover provided us a very-simple-but-efficient meta-model. From Figure 4.5, the meta-model comprises three classes `SubSystem`, `HasLC`, and `HasPort`. Each class has at least one attribute (property). `SubSystem` has an attribute `elementName` which represents the name of sub-system. The class `HasLC` has one attribute `elementName` which represents the name of LC. The class `HasPort` has three attributes namely `elementName` containing the name of port, `IODirection` specifying the type of port (`PROVIDE` or `REQUIRE`), and `dataElement` keeping the name of port. All attributes are of type `String`.

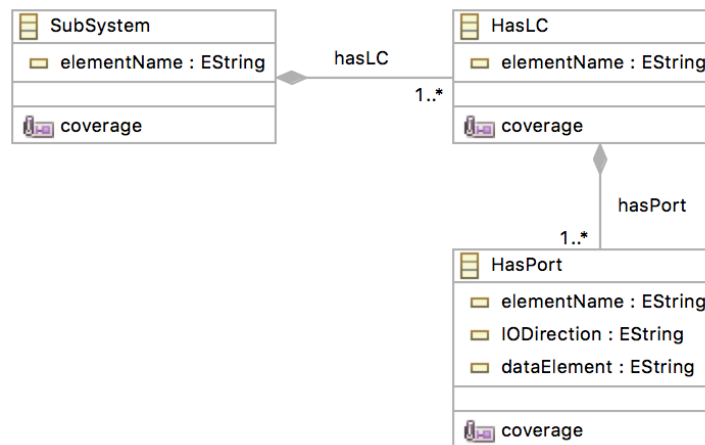


Figure 4.5: Meta-model of the optimized JSON document discovered by JSON Discoverer tool.

The meta-model also has two class relationships: `hasLC` and `hasPort`. The first relationship is a composition meaning that the class `HasLC` will be destroyed if the class `SubSystem` is destroyed. The same type of relationship is also applied to the relationship between the class `HasLC` and `HasPort`. Both relationships have multiplicity `1..*` according to the optimized JSON document. Note that the annotations `coverage` included in all classes are resulted from the the JSON grammar rules the authors of JSON discoverer created to guide the generation of the JSON meta-model [3].

As it has been mentioned that JSON discoverer also provides the discovery of data model (model instance). Using the same optimized JSON document, we obtained a model instance in XML² Metadata Interchange (XMI) file. A part of the file can be seen in Listing 4.4.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <discoD:SubSystem
3     xmi:version="2.0"
4     xmlns:xmi="http://www.omg.org/XMI"
5     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6     xmlns:discoD="http://jsonDiscoverer/discovered/SubSystem"
7     xsi:schemaLocation="http://jsonDiscoverer/discovered/SubSystem
8         metamodel.ecore" elementName="Visibility Control">
9     <hasLC elementName="LC 1">
10        <hasPort elementName="PORT 1" IODirection="REQUIRE" dataElement
11            ="DATA-ELEM 1"/>
12    </hasLC>
13    ...
14 </discoD:SubSystem>
  
```

Listing 4.4: Data model or model instance discovered by JSON discoverer

²Extensible Markup Language

4. Implementation

Using the editor from the EcoreTools Eclipse plug-in, the model instance and the properties of each class are visualized as seen in Figure 4.6. From the model instance (left), it can be seen that the attribute `elementName` of the class `SubSystem` is "Visibility Control". It has 18 children (`HasLC`), `LC 1`, `LC 2`, `LC 3`, ..., `LC 18`. Each child has its own `HasPort`, `PORT 1` belongs to `LC 1`, for example. The properties of each class are shown on the right side of the figure. Note that the model instance was constructed conforming to the meta-model in Figure 4.5.

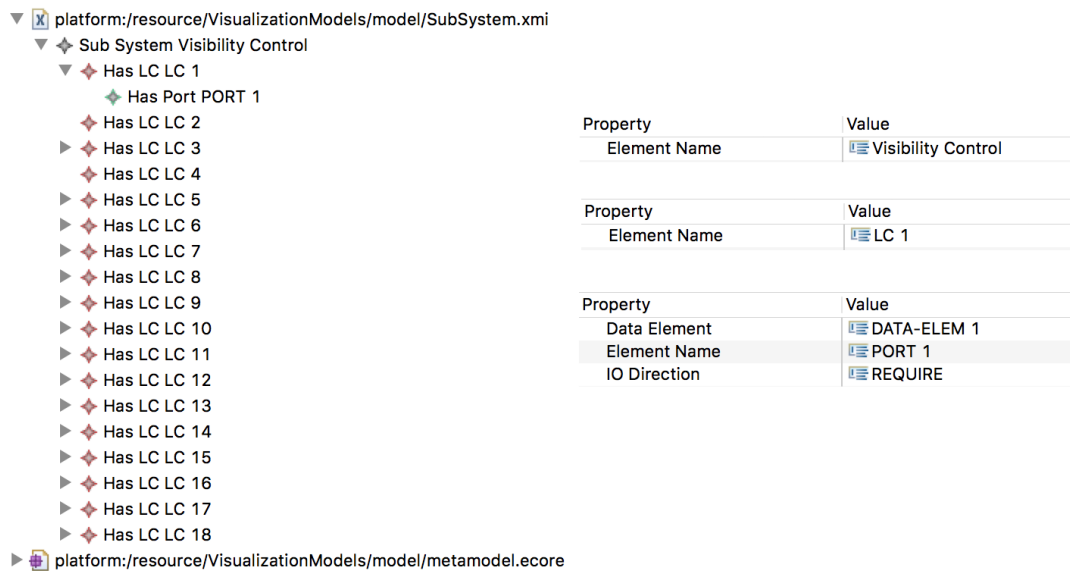


Figure 4.6: Model instance resulted from the optimized JSON document.

4.1.4.2 Automated visualization prototype

Up to this point, we already had the meta-model and the model instance from the optimized JSON document. The next step was to do M2T transformation, which could be done by create an Acceleo template for textual description. The complete template is included in Appendix A Section A.1. Only important pieces of code will be explained in this section.

Defining sub-system

```
1 ...
2 [template public generateElement(subsystem : SubSystem)]
3 ...
4 artifact [subsystem.elementName.replaceAll(' ', '')/] {
5 ...
6 }
```

Listing 4.5: Defining artifact for the sub-system

The sub-system `Visibility Control` is visualized as an artifact following by its name. To be able to get the name, we first created a `SubSystem`-type

parameter, namely `subsystem`. With this parameter, we could access its attribute `elementName`. A `replace` operation is used to remove white space in the value because PlantUML does not support name with white space. Thus, the name printed in the visualization is `Visibility Control`.

Defining LCs

```

1 ...
2 [for (lc:HasLC | subsystem.hasLC)]
3 ['[/][lc.elementName.replace(' ', '')/][''/]
4 [/for]
5 ...

```

Listing 4.6: Defining component for the LCs

Inside the artifact, a `for`-loop statement is implemented for retrieving all LCs belonging to the sub-system. The `for`-loop has a parameter `lc` belonging to the class `HasLC` which can be accessed via the class relationship `hasLC`. The name of the LCs can be obtained by `lc.elementName`, and the `replace` operation is used as well to remove white space.

Defining ports and connections

```

1 ...
2 [for (lc:HasLC | subsystem.hasLC)]
3 [for (port:HasPort | lc.hasPort)]
4 [for (lc2:HasLC | subsystem.hasLC)]
5 [for (port2:HasPort | lc2.hasPort)]
6 [if ((port2.dataElement = port.dataElement) and not (port2.
   elementName = port.elementName) and not (lc.elementName = lc2.
   elementName)) and (lc2.elementName.replace(' ', '').substring(3).
   toInteger() > lc.elementName.replace(' ', '').substring(3).
   toInteger()))]
7 [if not (port.IODirection = port2.IODirection)]
8 [lc.elementName.replace(' ', '')/] "[port.elementName.substring(6)
  /]['-'/][port.dataElement.substring(11)/]" [if (port.IODirection
  = 'REQUIRE')]#--([elseif (port.IODirection = 'PROVIDE')]#--0[/if
  ][if (port2.IODirection = 'REQUIRE')]--#[elseif (port2.
  IODirection = 'PROVIDE')]0--#[/if] "[port2.elementName.substring
  (6)/]['-'/][port2.dataElement.substring(11)/]" [lc2.elementName.
  replace(' ', '')/]
9 [elseif (port.IODirection = port2.IODirection)]
10 [/if]
11 [/if]
12 [/for]
13 [/for]
14 [/for]
15 [/for]
16 ...

```

Listing 4.7: Defining ports and connections

4. Implementation

Ports and connections in textual description has to be in the same line, in the following syntax:

$$A_1 \text{ "B}_1\text{-C}_1\text{" \#-(0-\# "B}_2\text{-C}_2\text{" A}_2$$

Assume that A is the name of a LC, B is the number part of a port name, for example, 1 is the number part of PORT 1, and C is the number part of data element of the port. The connection between two ports was defined based on the type of a port (attribute `IODirection`), `\#-(` for REQUIRE port and `\#-0` for PROVIDE port.

To obtain the name of all LCs and their ports, we created a new for-loop looping through `HasLC` and an inner for-loop to get port(s) of each LC looping through `HasPort`. This results in obtaining $A_1 \text{ "B}_1\text{-C}_1\text{"}$. After that, we created another two for-loops inside the inner for-loop in order to search for each LC's pair, resulting in obtaining $\text{"B}_2\text{-C}_2\text{" A}_2$. An if-statement is used to pair two ports by checking if their attributes `dataElement` have the same value, and validate that the two LCs are not the same LC. For the connection between two ports, two if-statements are created, one to check the type of each port, and another to pair between two ports having different types. This results in having two representations of connections as follow:

- REQUIRE-PROVIDE, `\#-(0-\#`
- PROVIDE-REQUIRE, `\#-0)-\#`

From the template, Acceleo generated a textual description that could be visualized using PlantUML. Listing 4.8 shows a piece of it.

```
1 @startuml
2
3 skinparam nodesep 80
4 skinparam ranksep 80
5
6 artifact Visibility Control {
7 [LC1]
8 [LC2]
9 [LC3]
10 ...
11 LC1 "1-1" #--(0--# "165-1" LC18
12 LC3 "2-2" #--(0--# "144-2" LC17
13 LC3 "4-4" #--(0--# "116-4" LC10
14 ...
15 }
16
17 @enduml
```

Listing 4.8: A piece of textual representation generated from Acceleo template engine

The result from the textual description is presented in Chapter 5 Section 5.1.1.

The automated visualization prototype of the sub-system `Visibility Control` can be seen in Section 5.1.1.

4.1.4.3 Final automated visualization prototype

Once the automated visualization prototype was created, we arranged the second meeting with the software engineer for presenting the prototype and getting some feedback regarding the prototype. We found out that the most important information that should be included in the visualization was the `dataElement` properties of the class `HasPort`, not `elementName`. Because of this reason, we improved the automated visualization by introducing a representation for data element as well as redefining sub-system, and ports and connections.

Defining data element

```

1 ...
2 [for (lc:HasLC | subsystem.hasLC)]
3 [for (port:HasPort | lc.hasPort)]
4 rectangle [port.dataElement.replace('-', '').replace(' ', '')/]
5 [/for]
6 [/for]
7 ...

```

Listing 4.9: Defining rectangle for the data element

`Rectangle` was used for representing all data elements provided/required by ports. Two for-loop statements was created. The first one is for retrieving all LCs belonging to the sub-system. The second one is for retrieving all ports belonging to each LC. The data element can be obtained by `port.dataElement`, and two replace operations are used as well to remove dash symbol and white space.

Redefining sub-system

```

1 ...
2 package "[subsystem.elementName.replaceAll(' ', '')]"
3 ...
4 }
5 ...

```

Listing 4.10: Redefining package for the sub-system

Sub-system was redefined from Section 4.1.4.2. Instead of using `artifact`, `rectangle` is used to represent the sub-system.

Redefining ports and connections

```

1 ...
2 [for (lc:HasLC | subsystem.hasLC)]
3 [for (port:HasPort | lc.hasPort)]
4 [if (port.IODirection = 'PROVIDE')]
5 ['[/][lc.elementName.replace(' ', '')/][']'/' -O [port.dataElement.
   replace('-', '').replace(' ', '')/]
6 [/if]
7 [if (port.IODirection = 'REQUIRE')]
8 ['[/][lc.elementName.replace(' ', '')/][']'/' -( [port.dataElement.
   replace('-', '').replace(' ', '')/]
9 [/if]
10 [/for]
11 [/for]
12 ...

```

Listing 4.11: Redefining ports and connection

Ports and connection syntax was redefined as follow:

$$A - (D$$

Assume that A is the name of a LC, and D is the data element. The connection between a port and its data element was defined based on the type of a port (attribute `IODirection`), `- (` for REQUIRE port and `-O` for PROVIDE port.

From the template, Acceleo generated a textual description for the final version of the automated visualization. Listing 4.12 shows a piece of it.

```

1 @startuml
2 ...
3 package "Visibility Control"{
4 rectangle DATAELEM1
5 rectangle DATAELEM2
6 rectangle DATAELEM3
7 ...
8 [LC1] - ( DATAELEM1
9 [LC3] - ( DATAELEM2
10 [LC3] - ( DATAELEM3
11 ...
12 }
13 @enduml

```

Listing 4.12: A piece of textual representation generated from Acceleo template engine for the final version of the automated visualization

The result from the textual description is presented in Chapter 5 Section 5.1.2.

4.2 Preliminary to cycle 2

4.2.1 Selecting stakeholders

We organized a second meeting with the software engineer (introduced in Section 4.1.1) and he gave us some feedback to work on. The obtained feedback lead us to the second visualization. In the meeting, we explained to him our next step which was having interviews with other stakeholders to get more requirements and different perspective on the visualization of the Database. Since the visualization was based on a single sub-system `Visibility Control`, we preferred to interview the stakeholders who were familiar with the sub-system and the entire Database as a whole. As a result, he gave us 4 contacts, 2 of the stakeholders were working at the System level and the remaining 2 were working at the sub-system level. Luckily, we were able to complete 3 interviews which included one stakeholder from sub-system level and the remaining 2 stakeholders were from the system level. We did not had a chance to interview the last person.

The first stakeholder had a position of test engineer at the company. She has been working at the company for 3 years and she has been using the Database at least 2 days a week. Her responsibilities have been to:

- Specify test cases and test procedures for each requirement in the test domain for system level and all functional level
- Generate test report on Database, DVM (design verification method)
- Upload the test results to Database

The second stakeholder was previously a system designer at the sub-system level but he is now a software developer at the company. (Note that this stakeholder is not the same person in Section 4.1.1). He has been working at the company for approximately 4 years and he has been using the Database several times a week. His responsibilities have been to:

- Responsible for different functions (locking and visibility)
- Work with the Node, in-house software development

The third stakeholder had the position of system designer at the company. He has been working at the company for 2 years and he has been using daily the Database. His responsibilities have been to:

- Define signaling within a system, and connections to other system.
- Write system level requirements
- Write functional realization requirements (depending on the size of a system)

4.2.2 Preparation for interviews

After we completed with the design of the automated visualization prototype. We conducted three semi-structured interviews. During each interview session, an interviewee was asked many questions from the list in sequence, but he/she was not

limited to those pre-determined questions. Some questions straying from the list were asked if the interviewers found it appropriated. All conversions in the interviews were recorded which were then transcribed and analyzed later to answer the research questions. Below is the list of the applied steps when formulating questions for interviews:

1. We established the goals of the interviews. The goals were:
 - To get to know the interviewee, the position and responsibilities
 - To get an understanding of how the interviewee use the Database
 - To identify the needs of the interviewee towards the automated visualization
 - To get the opinion of the interviewee towards the automated visualization prototype
2. We developed the list of questions of interests.
 - We, as the interviewers, developed a list of pre-determined questions aiming at answering the two research questions in Section 1.3 and their opinions on the automated visualization prototype. The list of questions can be seen in Appendix B.
3. We designed a template which had a list of questions to be asked during the interview.
 - It was said in the paper [1] that we needed to select someone with enough knowledge in the area to review the interview questions. As for the validation of the template, we had a support from our supervisor Truong to make sure we had the correct questions and properly formulated.

4.3 Cycle 2: Identifying needs of stakeholders

The second cycle of AR methodology has two steps, the first step involved data collection presented in Section 4.3.1. The second step of this cycle covers the data analysis and interpretation, this is presented in Section 4.3.2.

4.3.1 Data collection

As mentioned in Section 3.2.2, we applied semi-structured interview to get the needs from stakeholders. This step was done soon after we completed the selection of group of stakeholders which has been explained in Section 4.2.1 and setting up the interview questions explained in Section 4.2.2.

4.3.2 Data analysis and interpretation

The last step we followed in this AR methodology was to analyze the data and interpret it. This is firstly covered in Section 4.3.2.1 where we have discussed on the way we have done the coding on the interview transcripts. The second section of this part is about categorizing the needs of stakeholders that have been obtained from the coded interview transcripts, this is discussed in Section 4.3.2.2.

4.3.2.1 Coding interview transcripts

After another set of data from interviewing the stakeholders were collected. We proceeded to the data analysis by listening to the tapes recorded during each interview twice in order to get familiar with the conversations. The tape records were then transcribed manually and word by word. We also performed validation of the transcripts according to Basili's methodology mentioned in Section 3.2.2 by reading through the files and listening to the records at the same time.

The next step of the data analysis was to perform the coding on the transcripts to capture the key phrases/sentences spoken by the interviewees. We decided to use a software for qualitative data analysis called *NVivo* to help handling the coding process. In the beginning of the process, we created a list of codes and divided them into four categories corresponding to the goals of the pre-determined interview questions, as follow:

1. **Category: Personal info**
 - (a) code: NAME
 - (b) code: POSITION
 - (c) code: RESPONSIBILITIES
2. **Category: Use of the Database**
 - (a) code: DURATION
 - (b) code: FREQUENCY
 - (c) code: ADVANTAGES
 - (d) code: CHALLENGES
3. **Category: Specific task**
 - (a) code: TASK DESCRIPTION
 - (b) code: INFORMATION NEEDED
 - (c) code: SEEKING FOR INFORMATION
4. **Category: Automated visualization**
 - (a) code: OPINIONS
 - (b) code: NEEDS

The first category consists of three codes, NAME for the name, POSITION for the position, and RESPONSIBILITIES for the responsibilities of each interviewee. The second category consists of four codes, emphasizing the use of the Database. It basically covers how long the interviewees have been using the Database, how often they use it, the good things and the problem that they face when using the Database. The third category has three codes. The codes aim at inquiring the most recent task that the interviewees were assigned to, including information that they needed and how they looked for it in the Database. The fourth category consists of two codes, capturing their opinions on the automated visualization prototype and the needs of them towards the automated visualization.

The transcripts were coded twice by each of us, we compared the results together in order to check if something was missing, or miscoded. The results from the comparison were then finalized as can be seen in Section 5.2.

4.3.2.2 Categorizing needs of stakeholders

After the data from the stakeholders were analyzed, it was then put into categories. We came up with 3 categories which were **Dependencies**, **Clusters** and **Features**. Note that these categories depend on the code of NEEDS which we have presented in section 4.3.2.1.

The first category which is Dependencies relies more on the connection between artifacts of car. The artifact could be requirements, ECUs, LCs, and SWCs. The key idea of having this category is that the stakeholders insisted on being to trace the connection between one artifact of a car to another artifact and that is why we have this category.

The second category which is Clusters concerns grouping artifacts to show their parent artifact. One example could be considered on several LCs which belongs to a certain ECU. In this example, the idea is to indicate to which ECU this group of LCs belong. The answer that, they all belong to a single ECU. The same principle applies to a group of requirements which belongs to a single LC, a group of ECUs which belongs to a single SWC, also a group of ECUs which belongs to a single subsystem.

The last category which is Functional Requirements is based on the additional things that the stakeholders wanted to see in the visualization. This can be for example new requirement or something relating to improve the attractiveness of the visualization output.

5

Results

This Section presents both the prototype and the final version of the automated visualization which are the results from the textual descriptions generated from Aceleo templates. The results from coding are presented here in Section 5.2. Section 5.3 presents the results from analyzing the needs of stakeholders aimed to answer the research questions. The results from coding the transcripts and categories of needs of the stakeholders are also presented here. In addition to that, this section also presents how to use the tools both developed during this thesis work and the open-source projects.

5.1 Automated visualization

5.1.1 Automated visualization prototype

Figure 5.1 is Automated visualization prototype of the sub-system `Visibility Control`. The sub-system has 18 LCs represented by component and 179 ports represented by small squares attached to each LC according to the PlantUML syntax presented in Table 2.1 in Chapter 2. The lines show the connections between the ports on different LCs represented by `-o`, or `-()`. The LCs which do not have ports nor connection are the ones missing the connection with the rest of the LCs. These are LC2, LC4, LC16 and LC16. Some of the LCs seems to be having a lot of ports and overlapping of data in it, these are LC3, LC6, LC10, LC17, LC18. The reason to that it is because the ports of these LCs connect to many other ports.

However, there are some limitation of PlantUML. One of the limitations is that the tool does not allow users to move any components once they are rendered from textual description. This makes quite a big impact to our visualization since the sub-system has a large number of artifacts.

The automated visualization prototype gives an overview of how the artifacts in the sub-system are connected. However, the prototype has to be improved. Another limitation that can be seen from Figure 5.1 is, there are some overlapping texts in the visualization which makes it difficult to read. These texts are quite important because they represent the data element sent to/received among ports.

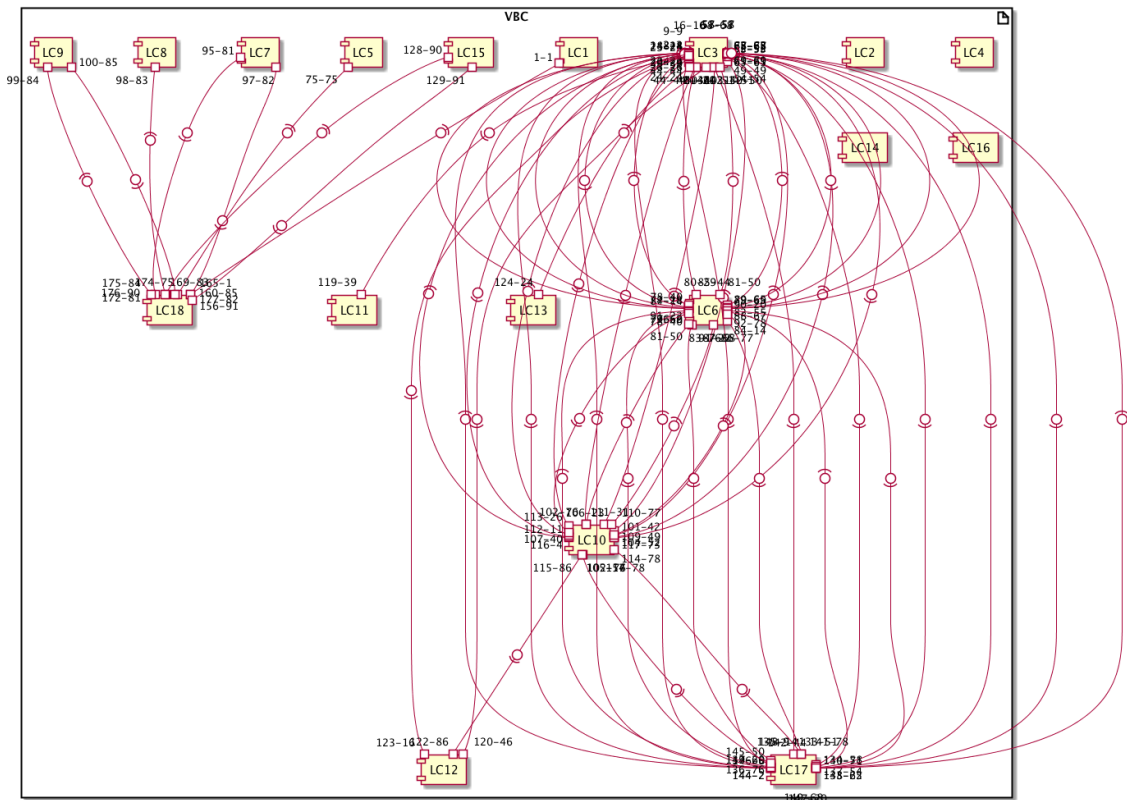


Figure 5.1: Automated visualization prototype.

5.1.2 Final automated visualization prototype

After the automated visualization prototype was presented to the Software Developer who gave us the scope, we took his feedback into account and improved the visualization. Figure 5.2 is the results after the improvement. The data used were the same set. The name of the ports were omitted, and we emphasized on the representation of the data element.

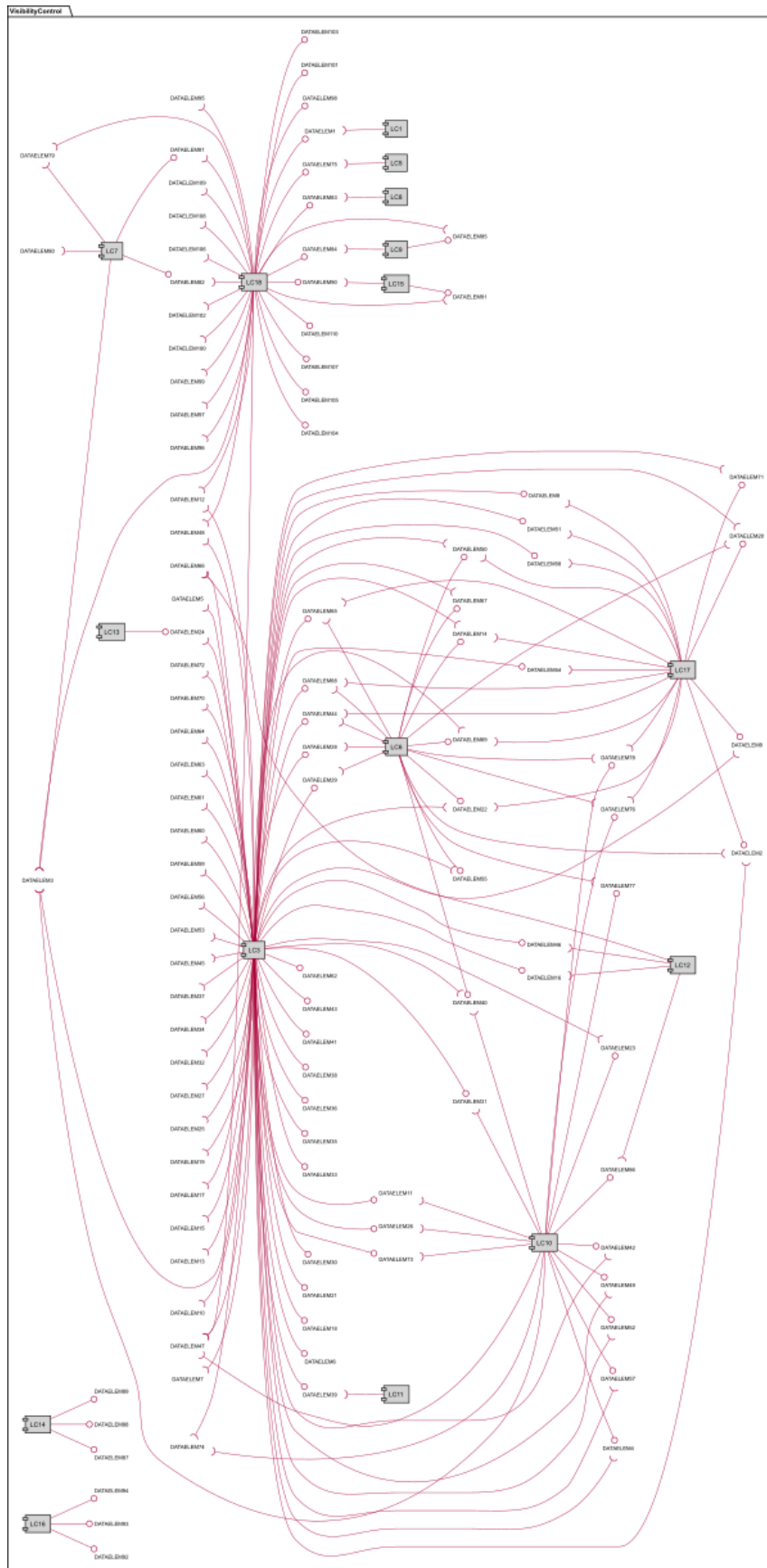


Figure 5.2: Final version of the automated visualization.

5.2 Coding results

The data collected from interviewing the stakeholders were transcribed and analyzed using the coding method were explained in Section 4.3.2.1. In this section, the results from the coding are presented by diagrams for four coding categories in the following sub-sections. Each diagram consists of the three objects: grey rectangle represents coding category, oval represents code, and floating object represents coded data from the transcripts.

5.2.1 Category: Personal info

Diagrams in Figure 5.3, 5.4, and 5.5 illustrate the codes and their coded data under the category Personal info of the stakeholders (interviewees). As it has been introduced, the category has three codes: NAME is used for data in the transcripts that provide the name of the stakeholders, POSITION states the their positions, and RESPONSIBILITIES is for the data that describe the responsibilities of their positions. Note that the interviewee's names are omitted due to anonymity reason.

The first stakeholder (Figure 5.3) works as a test engineer and is responsible for specifying test cases and test procedures for each requirement. After executing test cases, she uploads the results to the Database. In addition to that, she also generates Design Verification Method (DVM) report from the Database.

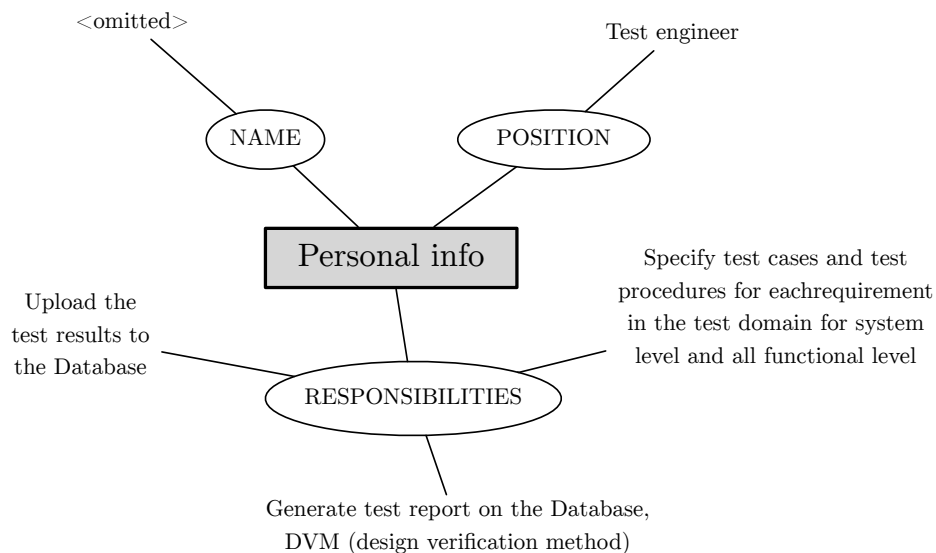


Figure 5.3: Coding results of test engineer for Category: Personal info.

The second stakeholder (Figure 5.4) that we interviewed works as a software developer. His main responsibility for the position is to develop software for different functions in-house.

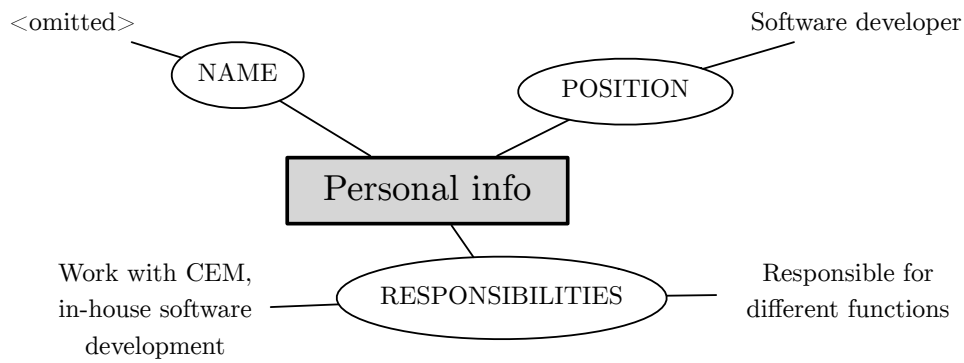


Figure 5.4: Coding results of software developer for Category: Personal info.

The third stakeholder (Figure 5.5) works as a system designer. His responsibilities are to write function realization requirements, system requirements, including defining signals and their connections within a certain sub-system.

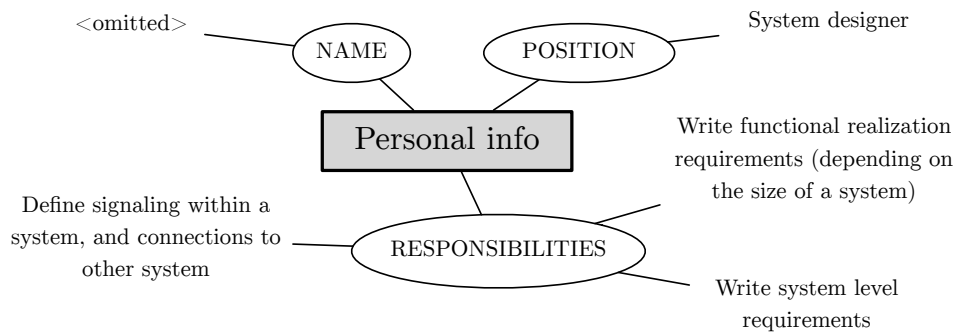


Figure 5.5: Coding results of system designer for Category: Personal info.

5.2.2 Category: Use of the Database

Diagrams in Figure 5.6, 5.7, and 5.8 illustrate the codes and their coded data under the category Use of the Database. The category has four codes: DURATION is used for data in the transcripts that describe how long the stakeholders have used the Database, FREQUENCY is for the data that provides information on the frequency the stakeholders use the Database in a week, ADVANTAGES explains the advantages of the Database they envision, and CHALLENGES is used for the data that describe disadvantages/challenges the stakeholder have faced when using the Database.

From the Figure 5.6, the test engineer has used the Database for 3 years, at least two days a week. From her point of view, advantage of the Database is that it provides statistical data such as the number of test cases that fail or pass, and the number of test verification. Still, she finds the Database difficult to interact with, especially when filling in information in the tool. It cannot show which requirements are for specific software releases. These are the data that she needs to see.

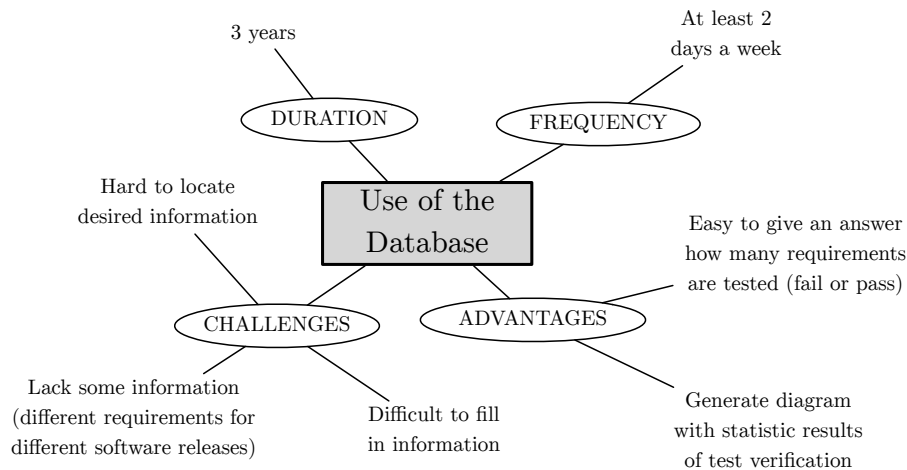


Figure 5.6: Coding results of test engineer for Category: Use of the Database.

From the Figure 5.7, the software developer has used the Database since the beginning of his work at VCG for approximately 4 years, and several times a week. He said that the tool has some advantages, for example, it provides him some information he needs such as requirements on each signals. But, the Database is quite slow from time to time. The tool does not provide the visualization of the connections and signals among ECUs. Sometimes, the information such as revisions of each artifact is not up-to-date or it is updated before the implementation, which causes confusion. In addition to that, he finds it hard to handle variants of artifacts.

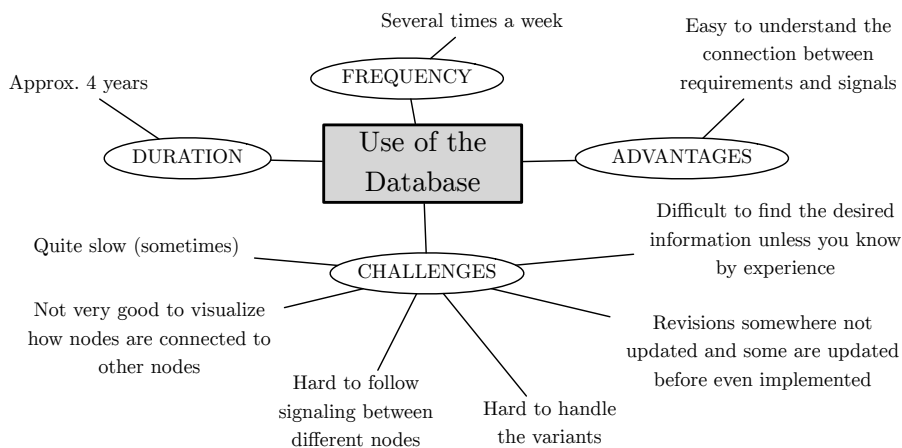


Figure 5.7: Coding results of software developer for Category: Use of the Database.

From the Figure 5.8, the system designer has used the Database for 2 years on daily basis. The advantages he sees when using the tool are that, it is the place where the data are organized, and the data in the tool can be used as a reference when communicating with other people both in the same and other departments. However, the Database is slow from time to time and it does not have user-friendly interface. Some background information of why an artifact is created is missing.

Furthermore, information is misplaced sometimes due to human error. Some useful features such as a feature to log deviations, and a feature to specify accepted/denied requirements are not available. He also added that there are failures occurred once per week.

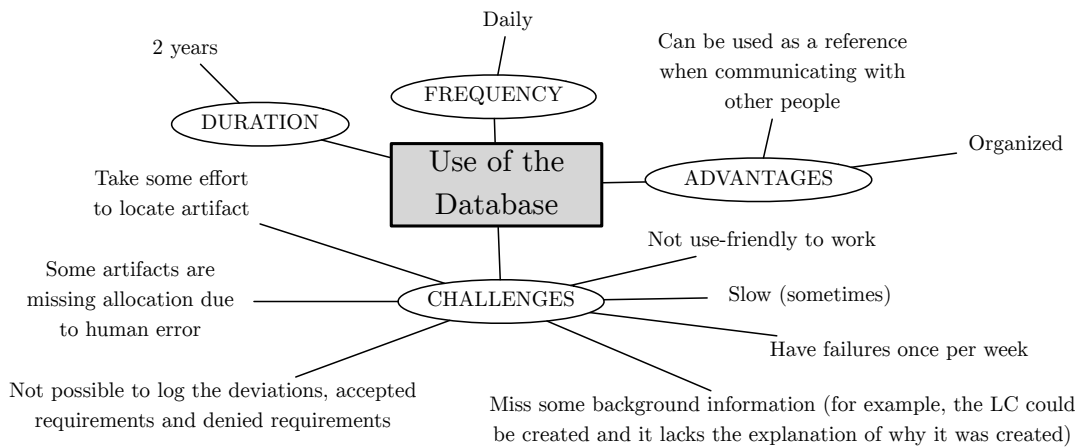


Figure 5.8: Coding results of system designer for Category: Use of the Database.

5.2.3 Category: Specific task

The codes and their coded data under the category Specific Task are shown in the diagrams in Figure 5.9, 5.10, and 5.11. The category has three codes: TASK DESCRIPTION is used for the data in the transcripts that describe the details about the most recent task of the stakeholders, INFORMATION NEEDED specifies what information they need to complete the task, and SEEKING FOR INFORMATION explains how the stakeholders look for those information.

The test engineer explained us her most recent task (see Figure 5.9), which was to write test procedure of a sub-system based on System Requirement Description (SRD). In order to write test procedure, she needed to know the requirements of each LC under ECUs in the sub-system. These requirements were stored in the Database, only she had to locate them using the search function provided by the tool.

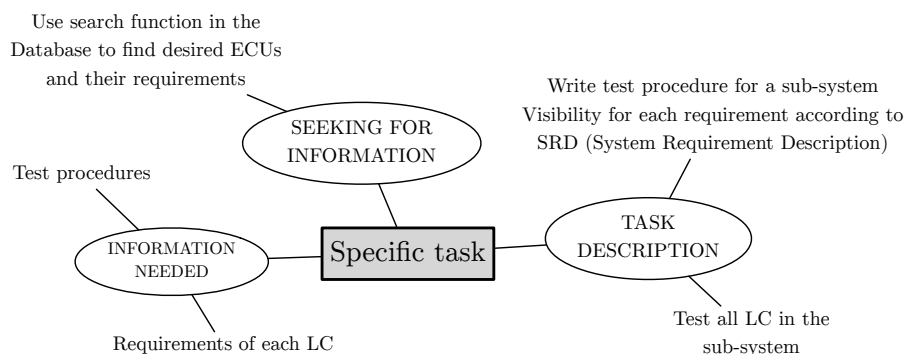


Figure 5.9: Coding results of test engineer for Category: Specific task.

The most recent task that the software developer used as an example was, to add signals and ports from new requirements to the existing LCs and ECUs (see Figure 5.10). To be able to complete the task, he needed to know the name of the ECU where the LCs belong to, and in which SWC. The way to find the information was to contact the person responsible for the specific artifact, or else he had to know by his experience where the artifact was located in the Database.

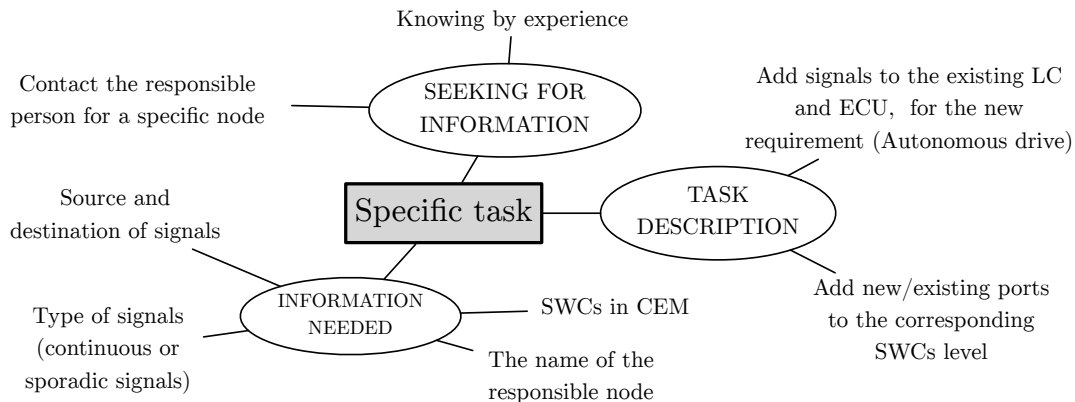


Figure 5.10: Coding results of software developer for Category: Specific task.

The most recent task that the system designer worked on was to design a system for autonomous drive by adding variants, and functional requirements into the system (see Figure 5.11). The information that he needed was the requirements on the system, including the corresponding LACs, LCs, SWCs, signals, sub-system, and variants. To get hold of this information, he needed to talk to people responsible for the artifacts, and to look for the artifacts in the Database using search function.

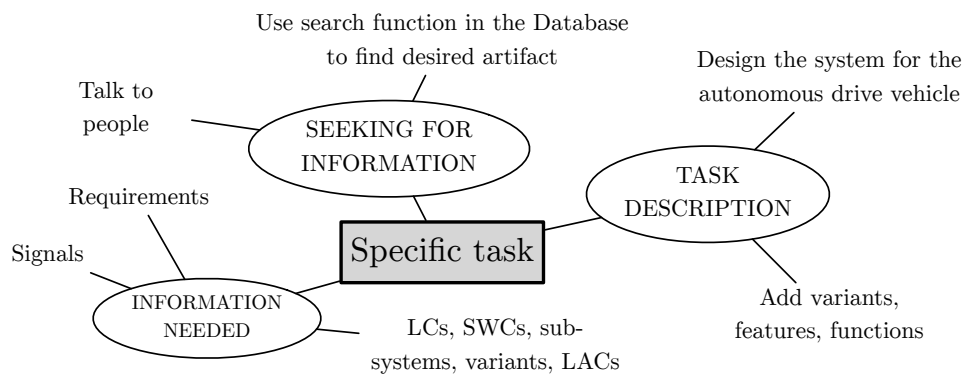


Figure 5.11: Coding results of system designer for Category: Specific task.

5.2.4 Category: Automated visualization

Diagrams in Figure 5.12, 5.13, and 5.14 illustrate the codes and their coded data under the category Automated visualization. The category has two codes: OPINIONS presents the stakeholders' opinions on the automated visualization in general, and NEEDS is the functional and non-functional requirements that the stakeholders

want for the improvement of the automated visualization.

The test engineer also expressed her opinions on the automated visualization (see Figure 5.12). She says that it is easy to understand how the sub-system `Visibility Control` works, and how everything is connected. The visualization can provide her some information that she needs, and it is faster than looking for it in the Database.

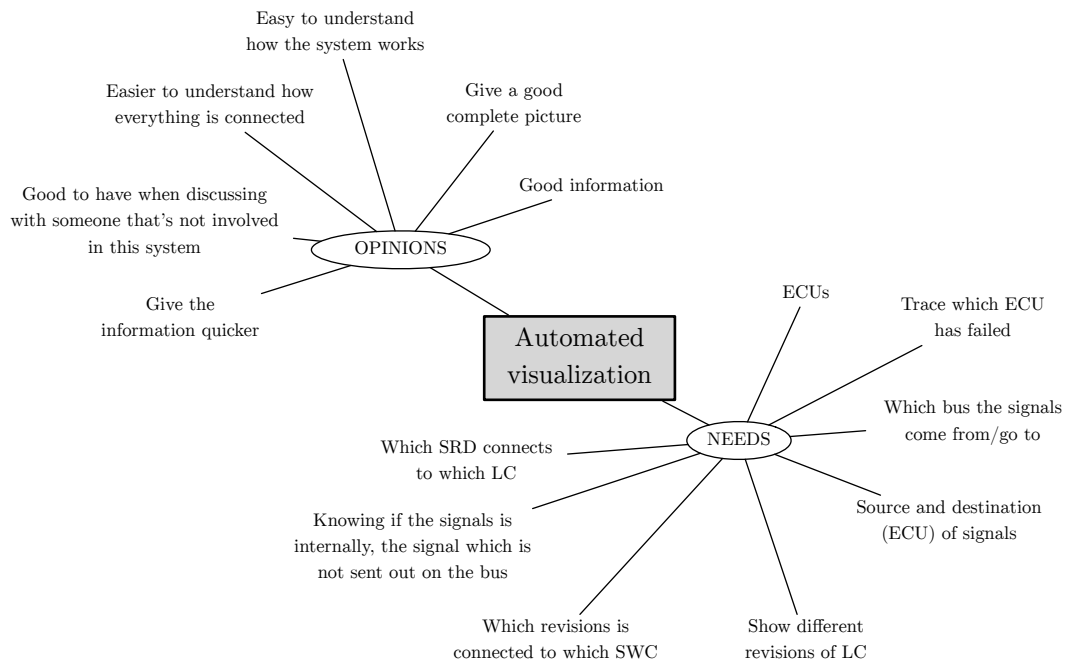


Figure 5.12: Coding results of test engineer for Category: Automated visualization.

However, the test engineer needs the automated visualization to combine physical view (ECUs) and the logical view together. It should be possible to see the source, destination of the signals, and the bus where the signals are sent to/received from ECUs. On top of that, showing which SRD relates to which LC, and which revision is connected to which SWC would be useful information for her.

The software developer gave us his opinions on the automated visualization (see Figure 5.13). He says that it looks nice. The visualization has good information on connections among artifact in the sub-system which makes it easy to understand how the system works. He thinks that it might the visualization will be difficult to deal with if the sub-system is bigger.

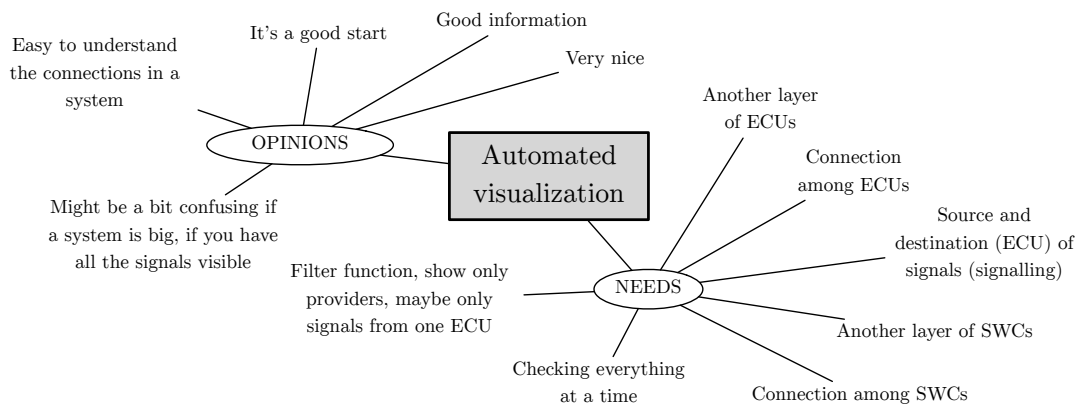


Figure 5.13: Coding results of software developer for Category: Automated visualization.

The software developer also specified his needs towards the automated visualization. Similar to what the test engineer mentioned, another layer of ECUs and SWCs would be beneficial. Being able to know the source and destination of signals sent to/received from ECUs are useful. In addition to that, some features such as filter function filtering only information that he needs to see would make the visualization more efficient.

The software designer gave us his opinions on the automated visualization (see Figure 5.14). He says, it is beneficial for him. But, it still lacks of information. Similar to the software developer said, the visualization might be difficult to deal with if the sub-system is big. On top of that, it should be more user-friendly and intuitive.

He specified his needs towards the automated visualization. Adding another view of LACs could be more beneficial for him. Similar to the test engineer’s needs, some other important information such as buses the signals are sent to/received from, is missing. Moreover, being able to sort and filter information shown on the visualization, and to distinguish buses by different colors would be more efficient.

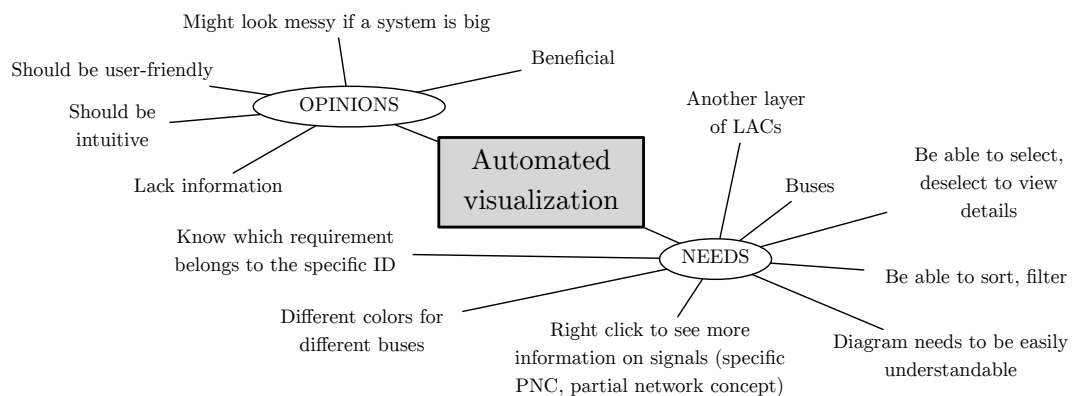


Figure 5.14: Coding results of system designer for Category: Automated visualization.

It is obvious that some needs are the needs towards the automated visualization in common, and some needs are specific to individual's role and responsibilities. The common need is that, for example, they want to see which type of bus the signals are sent on. The needs specific to each stakeholder is, for example, the test engineer wants to see the number of failed ECUs, but this is not in the interest of the software developer and the system designer since testing is not their responsibilities. Similarly, the system designer wants the visualization to show the connections among LACs which is another abstract level, but the test engineer and the software developer did not mention about it as it is not useful for them.

5.3 Categories of needs of stakeholders

In the section 4.3.2.2, we introduced the categories of the analyzed needs of stakeholders which were the dependencies, clusters and features. Again, in the section 5.2.4, we have reported the results of the code of NEEDS, specifically shown in figure 5.12, figure 5.13 and figure 5.14. So, in this section, we have categorized the needs of stakeholders by considering what has been reported in the CODE of NEEDS. These categories will aid us to answer the first research question.

5.3.1 Dependencies

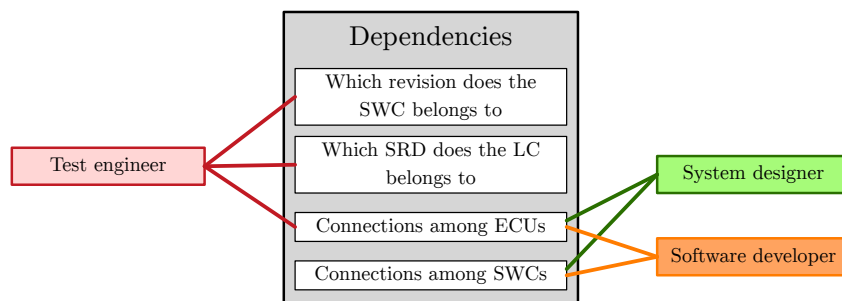


Figure 5.15: Needs in Dependencies group.

If you take a look on figure 5.15, you will notice that under the category of Dependencies, it has two needs which were mentioned by all stakeholders, these were the connection between the ECUs and the connection between the SWCs. This may imply that the ECUs and the SWCs are the artifacts that stakeholders interacts with frequently. The remaining two needs in this category were mentioned by the test engineer and she would like to see the relationship that exists between different revisions of the SWCs and also to see the relationship between the LCs and the SRDs. This is something that she said to be a challenge in her area of work in a sense that it is important to know if she was testing the correct revision of the SWC. It also applies if she is testing the requirements of the LCs which is found on the SRDs, then it is also important to know if she was testing the correct requirements.

5.3.2 Clusters

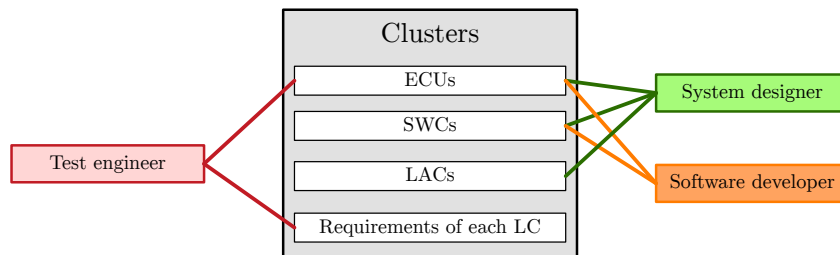


Figure 5.16: Needs in Clusters group.

As of the category of Clusters in figure 5.16, we have ECUs which was mentioned by all the stakeholders. As it was mentioned earlier, the ECU is said to have a composition of LCs, so this makes a single cluster. The same concept of cluster applies to SWCs, however the SWC is said to have a composition of ECU and this, in turn, makes a single cluster. LACs have a composition of LCs and this was only proposed by the software developer. The last cluster was the requirements, since each LC is said to have many requirements then it explains why we have this need under the category of Clusters. This was only proposed by the test engineer since she mostly works with testing requirements.

5.3.3 Features

The category of Features covers the additional things that can be added to the visualization to make it more attractive and flexible. As you can see on the visualization presented in Section 5.1.2, there are lots of input signals and output signals on a LC, and that was a visualization of one small sub-system. So the additional features categorized in this section would rather produce a better outcome. The system designer and the software developer proposed to have a feature of filtering, this would allow the visualization output to show less input and output signals. The implementer has a flexibility to apply different techniques so that he/she gets a nice filtered visualization. The system designer had needs such as to have a sorting feature on the visualized output, to visualize bus types and to add color for each type, to visualize the IDs of the requirements and how they are related. He also proposed to have a more dynamic visualization where a viewer of the diagram could select, deselect, or right click on artifact to view extra details if available on the visualized artifact. The system designer also insisted on producing the output that was user friendly.

The filtering was said to be beneficial but again the software developer said that it was also good to visualize everything at a time, this means the filtering should be to specific artifacts. The test engineer added features such as to be able to see the failed ECU, this is based to her tasks of testing, so then, the ability to visualize such

situation could be of benefit. She also added on the ability to different the internal signals and the output signals.

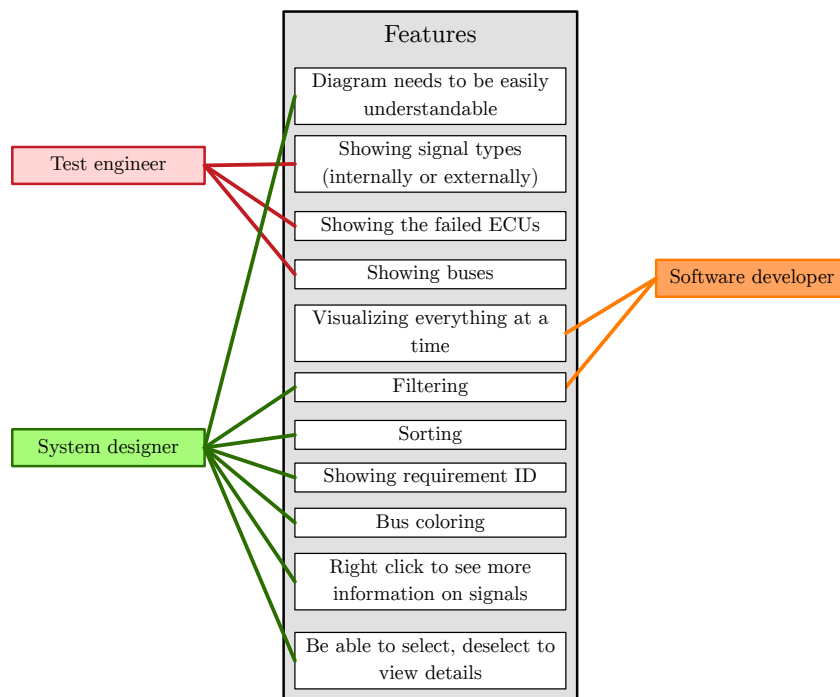


Figure 5.17: Needs in Features group.

6

Discussion

This chapter presents answers to the research questions stated in Section 1.3 and on the second part of this chapter, we have explained on how to use the source code that has resulted to the automated visualization.

6.1 Research questions

RQ 1. What are the needs of different stakeholders towards the automated visualization of the electrical architectures?

The needs which we have presented in the Section 5.3 depend on the requirements of tasks that each individual has been working on and also their responsibilities. The responsibilities depend on the position of the individual at the company. One important thing to note is that, the needs of stakeholders were based on the analysis of each interview session. The elicitation process was not affected by our automated visualization prototype. The prototype was shown to the stakeholders at the very end of each interview session, which in turn allowed the stakeholders to think outside of the box when discussing about their needs towards the visualization of Database. In order to answer this research question, we will discuss the synergies that exist between the categorized needs(dependencies, clusters and features) and the stakeholders.

Dependencies : High synergy exists on the dependency about connection between ECUs, this is because it has been mentioned by the first stakeholder (figure 4.3) on the first cycle of the applied AR methodology and also by all stakeholders (figure 5.15) on the second cycle. The next dependency which seems to have moderate synergy is the connection between SWCs and this was mentioned by two stakeholders on the second cycle and also mentioned by the Software Engineer on the first cycle (figure 4.3). The remaining dependencies can be said to have weak synergy since they were only mentioned by the Test Engineer on the second cycle. These dependencies are the relationship that exists between different revisions of the system with respect to their corresponding SWCs and also the relationship that exists between the SRD and the LC, which simply means to which SRD does the LC belongs. In this report, the two dependencies with weak synergy are mostly applicable to Test Engineer and this can be due to the fact that the revisions and the SRDs of the car keeps on changing, so it is important for the Test Engineer to know if she is testing the correct(current or old) artifacts.

Clusters : The first two clusters presented in figure 5.16 which are ECUs and SWCs have high synergies with respect to the number of stakeholders participated in this report, this can be explained in similar way as the dependency of the connection between ECUs and dependency of the connection between SWCs. The cluster of LACs can be said to have weak synergy as it was mentioned by only the System Designer and the cluster of requirements of each LC can also be said to have a weak synergy as it was mentioned by only the Test Engineer.

Features : This category corresponds to the needs that are neither in dependencies category nor clusters category. It includes the properties that were proposed to be added on the visualized diagram. These properties are:

1. Showing signal types (Externally and internally)
2. Showing the failed ECUs
3. Showing buses
4. Showing bus coloring
5. Showing everything at a time(all the ECUs of the system)
6. Showing requirement ID
7. The diagram needs to be understandable.

Suitability is the only non-functional requirement which can be seen from the reported features in fig 5.17. As it was defined from the ISO/IEC paper [25]), suitability is “the capability of the software product to provide an appropriate set of functions for specified tasks and user objectives”. The features which support this quality are filtering, sorting, right click to more information on the specified artifacts, be able to select and deselect to view details of an artifact. This quality is expected to be applied on the tool that will provide the visualization.

In general, the category of feature seems to have weak synergy between the stakeholders in almost all of the needs reported. The need to have filtering on the visualized output was mentioned by the System designer and the Software Developer and so this need can be said to have moderate synergy. All in all, each stakeholder interviewed can be seen to have their own preferences when it comes to features that they want to be considered on the visualized diagram as well as the tool that does the visualization.

RQ 2. How does an automated visualization of electrical architectures fulfil the needs of stakeholders?

On the interviews with stakeholders, we showed them our visualization of electrical architectures (Figure 5.2). Our diagram had visualized the LCs, signals, both the source signal and the receiver signal of the sub-system *Visibility Control*. The next paragraphs respond to RQ2 with respect to the perception of each stakeholder towards the automated visualization.

The test engineer was interviewed first, one of the difficulties that she mentioned when using the Database was that, it is difficult to locate the desired information in the Database. After we showed her the diagram, she responded that this type of visualization give a good complete picture of the system. She also added that it was easy to understand how the system works and how everything is connected. In addition to that, she said sometimes they discuss with other stakeholders who do not use the Database and with this visualization, it will help through their discussion.

The software developer was the second person to be interviewed, when asked about the challenges of using the Database, he said that the Database did not have a good way to visualize how nodes (ECUs) are connected and also he said that it was difficult to find the desired information unless someone knows by experience where to look for a specific information. When we showed him our visualization, he responded that it was very nice. Since our visualization did not have information of the nodes which is useful for his work, he said it was a good start. Note that, the need to visualize the nodes was reported to have high synergy on the first research question. In general, the Software Developer responded that with this kind of visualization, it was easy to understand the connections in a system.

The last person to be interviewed was the system designer. One of the mentioned problem of using the Database was similar to the previous stakeholder which was the difficulty in locating an artifact on a Database. Another problem that he mentioned was that, sometimes an artifact could be wrongly allocated on the Database and this could be due to human-errors. An example was when someone has allocated the LC to the wrong LAC or maybe forgetting to allocate the LC to the specific LAC. He mentioned that, this kind of mistakes can't be observed in the Database unless if you have a complete overview of the allocation of each LC to their responsible LAC. So, when he was asked about the visualization, he responded that it was beneficial in this kind of situations.

6.2 Use of the source code

This Section presents the tools required for creating an automated visualization, including the how-to steps.

6.2.1 Tools required

6.2.1.1 Open source projects

- **JSON Optimizer**

A Java project developed in this thesis work. The project can be downloaded from <https://github.com/florencemayo/visualization1.git>

- **VisualizationModels.Acceleo**

An Acceleo project developed in this thesis work. The project can be downloaded from <https://github.com/nattaponx/VisualizationModels.Acceleo.git>

- **JSON Discoverer**

A tool developed by Javier L. Cánovas Izquierdo and Jordi Cabot for discovering the implicit schema (meta-model) and the data model (model instance) of JSON documents. The tool consists of 7 projects:

- jsonDiscoverer
- jsonDiscoverer.coverage
- jsonDiscoverer.docs
- jsonDiscoverer.web
- jsonDiscoverer.tests
- jsonDiscoverer.examples
- jsonDiscoverer.zoo

The project can be downloaded from <https://github.com/SOM-Research/jsonDiscoverer.git>. Note that only jsonDiscoverer project (core implementation) and jsonDiscoverer.examples are used to create meta-model and model instance, but it is recommended to download all the projects.

6.2.1.2 Eclipse software & plugins

- **Eclipse Modeling Tools**

An Eclipse modelling package containing tools and runtimes for building model-based applications. The package can be downloaded from <http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/mars2>

- **EcoreTools**

A plugin for developing graphical modeling for ECORE. The plugin can be downloaded from <http://www.eclipse.org/ecoretools/>

- **Acceleo**

A code generator for M2T transformation. The plugin can be downloaded from <https://eclipse.org/acceleo/>

- **PlantUML**

An Eclipse plugin version of PlantUML tool allowing users to create UML diagrams from textual descriptions. The plugin can be downloaded from <http://plantuml.sourceforge.net/updatesitejuno/>

6.2.2 Installation

Please follow the instructions below to install the tools:

1. Install Eclipse Modeling Tools.
2. Once Eclipse Modeling Tools is installed, open the software and go to **Help > Eclipse Marketplace...** Search for EcoreTools and Acceleo, and install them.
3. Go to **Help > Install New Software...** Copy the the download link <http://plantuml.sourceforge.net/updatesitejuno/> and paste it in **Work with:** text field. Select PlantUML and install.

4. Clone the projects JSON Optimizer, VisualizationModels.Acceleo, and JSON Discoverer projects from GIT repositories and import them to Eclipse Modeling Tools.

6.2.3 Deliverables

6.2.3.1 Optimizing JSON document

To create an optimized JSON document, go to JSON Optimizer project in Eclipse Modeling Tools and follow the instructions below:

1. Go to **src/jsons**. Open the JSON file called **vcspa**, and replace the content in the file with the content in the raw JSON data extracted from the Database.
2. Go to **src/fileOptimizer** and run the Java file **JsonOptimizer**. An output optimized JSON document will be created in the same directory where **vcspa** is located.

6.2.3.2 Creating a meta-model and a model instance from JSON document

To create a meta-model from JSON document, go to JSON Discoverer project in Eclipse Modeling Tools and follow the instructions below:

1. Go to **jsonDiscoverer.examples/exampleData/simpleDiscoverer**. Open the JSON document called **json1A**. Replace the content with the optimized JSON data from 6.2.3.1 and save.
2. Go to **jsonDiscoverer.examples/src/jsondiscoverer.examples**. Run the Java file called **ExampleJsonSimpleDiscoverer**. The ECORE file of meta-model will be created as **exampleSimpleDiscover.ecore** in the same directory where **json1A** locates.

To create a model instance,

1. Go to **jsonDiscoverer.examples/exampleData/injector**. Open the JSON document called **json**. Replace the content with the optimized JSON data from 6.2.3.1 and save.
2. Go to **jsonDiscoverer.examples/src/jsondiscoverer.examples**. Run the Java file called **ExampleJsonInjector**. The XMI file of model instance will be created as **exampleInjector** in the same directory where **json** locates.

6.2.3.3 Generating a visualization

To create a visualization, go to VisualizationModels.Acceleo project and follow the instructions below:

1. Go to **VisualizationModels.Acceleo/src/VisualizationModels.Acceleo.main**.
2. Right click on the MTL file called **generate** and select **Run as > Run Configuration...**
3. Create a new Acceleo configuration. Specify the Acceleo project, main class, model instance, and location of the output textual description in the text fields

Project:, **Main class:**, **Model**, and **Target:**, respectively. Click Run button. The output file will be generated in the location you specify.

4. Open the textual description file.
5. Go to **Windows > Show View > Other...** and select **PlantUML**. The visualization will be displayed in the View.

Note:

1. The steps to create meta-model are performed only one time, unless the schema of JSON document extracted from the Database is changed. The steps to create a model instance and to generate a visualization must be performed every time that you want to create a visualization of a new JSON document.
2. The tools created in this thesis are ready to be used as open source software, still they need some improvement and to be merged as a single tool.

7

Validity Threats and Research Ethics

This chapter is divided in two sections. The first section discusses different validity threats that we have encountered and how we mitigated them. The second section discusses about the research ethics.

7.1 Validity Threats

In this section, the three types of threats to validity are presented. The threats are being discussed with reference to what have been detailed explained by C. Wohlin et al [6].

7.1.1 Conclusion validity

This type of validity applies to what could hinder us not to draw a correct conclusion from the treatment and the outcomes of the experiment.

Reliability of measures: Some of the questions that we asked during the interview were not answered effectively, meaning not in the way we wished to be answered. This might have been caused by either a wrong way of formulating a question or maybe the respondent did not understand the question. In order to mitigate this threat, we spend a bit of time on repeating the question by adding examples so that the interviewee could understand the question and it turned to be effect but time consuming.

7.1.2 Internal validity

Here we talk about threats that can affect the independent variables without the researcher's knowledge.

Instrumentation: In our visualization, we had applied the MDSE technique to get a diagram out of a text (textual description). And the text was transformed using a tool named JSON discoverer. The tool does the transformation of JSON file to XML file. We cannot be sure that the tool may be 100 percent efficient but again, its human who has designed a tool and nobody is perfect and so, this can also be a threat to the final diagram that we get for the visualization. However, we

manually verified on the transformed XML file and the final diagram to make sure all the components and all the data elements found in the original file were mapped correctly.

Selection: On the second cycle of the AR methodology, we interviewed only 3 stakeholders. We are limited in terms of the number of stakeholders. The stakeholders selected had different roles interacting with the Database. Two stakeholders were of the same role (system designer), one at the system level and another one at sub-system level. The third stakeholder was of different role, the Test Engineer and she was working on a system level. Different stakeholders have different needs, what we obtained from the stakeholders have a bit of similarities but that does not mean we might have gotten less or more needs. However, we consider the threat at acceptable level since:

1. We have applied the AR methodology which might contain a number of iterations. For the first iterations, with limitation in resource access, we considered the people who were interested in the visualization the more. We can involve more people in next iterations of the research.
2. The three stakeholders, test engineer, software developer, and system designer, who participated in the semi-structure interviews in the cycle 2 were chosen by the first stakeholder who give us the scope in the cycle 1. We considered the three stakeholders the representatives of each role for the data collection in cycle 2 as the Database was one of the main tools they used almost every day, their interests in automated visualization, and the their roles covered the three main phases in software development process, which are design, development, and testing phases.

7.1.3 Construct validity

This concerns generalizing the results based on the theory or concept behind it.

Inadequate pre-operational explication of constructs: We had experienced a bit of misunderstanding while interviewing one of the stakeholders. The question on the interview template was not clear to the interviewee so the interviewee explained a lot of things and many of the responses were out of topic. The purpose of the question was to know how does the interviewee find the tool (the Database) but then the interviewee decided to compare the tool to a similar requirement handling tool that we had not used or seen before. In order to mitigate this issue, we had to let the interviewee finish what he was explaining and we had to explain again on the purpose of the question, the side-effect was that we had used more time on this interview.

Mono-operational bias: The visualization was done on a single subsystem with rather few number of components and signals. The results is expected to be different if the system was rather bigger than the one visualized. Also, the visualization would have been different if it involved different artifacts such as requirements, LACs, SWCs or ECUs and this would vary for different subsystems. We consider

the threat at acceptable level. Again, we have applied the AR methodology which allows to go through different iteration do this threat may be mitigated on the next iterations.

7.1.4 External validity

Here we discuss about the conditions that limit us to generalize the results.

Interaction of selection and treatment: The needs that we obtained do not cover the entire population of stakeholders at VCG. The company is big and hence it has different people that are responsible for different tasks. We consider the threat at acceptable level since the task was to visualize the data stored in the Database, then it would have been better if the stakeholders that we interviewed were of different levels, meaning primary stakeholders, secondary stakeholders and key stakeholders. Some stakeholders do not interact with the Database but that does not mean, they do not have an impact on the needs that we analyzed. This means that, the stakeholder who does not interacts with the Database have an indirect impact on the results of the needs that we have obtained at the end. This threat can be mitigated on the next iterations.

7.2 Ethical consideration

Two important things that we took into account when conducting this study were, harm that might happen to participants involved in this thesis work, and to report everything with honesty. This Section is aimed to present the actions we took in order to minimize the risk of harm according to the principles explained by Louis Cohen, Lawrence Manion, and Keith Morrison [5] and to the way we report the results.

7.2.1 Informed consent

We had four main participants involved in this work, including the first Software Developer who gave us the scope of visualization, and the three stakeholders that were interviewed during the data collection in cycle 2. To minimize the risk of harm, we provided them, (1) clear explanation of the study and (2) what information that is required from them. The clear explanation included the information about us such as our names and our universities, the purpose, the steps taken, and the desired outcome of the study. The participants were also informed what kind of information was needed as part of the study. In addition to that, the participants were not forced to take part in any work in the study.

7.2.2 Anonymity and confidentiality

The data collected from the participants including their personal information were anonymous. The first set of data extracted from the Database contained some in-

formation that could identify who created the data. To make sure that the harm would be minimized, the pieces of information were censored.

During the interview sessions with the stakeholders, we asked for their permissions to record the conversations. The tape records and transcripts were kept anonymous and confidential. In the process of analyzing the data, we removed their personal information that could disclose their identities as much as possible to ensure that there would be no physical harm to them, and to prevent psychological distress and discomfort they might face after giving their information as a part of this thesis work.

7.2.3 Fraud

We also considered the fraud that may happen when collecting, analyzing, and reporting the results from the study. The data in the study are real and not made up as well as the methods we used to get them are explained clearly. Moreover, the data were not manipulated or discarded in order to get the desired outcome. To make sure that we, our universities, and the company will get the most benefit from the study, we report everything both negative and positive results. Most importantly, we have not taken someone else's work without giving the credits to them.

8

Conclusion and Future work

In this chapter, we present three sections. The first section covers the conclusion of our thesis work, covered in Section 8.1. The next part which is Section 8.1.1 covers the discussion about how our thesis work will be sustainable to the way of working with electrical architecture at VCG. In the Section 8.2 which is the last one in this chapter, we discuss the proposed future work.

8.1 Conclusion

The problem mentioned in Section 1.1 was that, a stakeholder found it hard to locate the needed information which in some cases, it mostly worked if a stakeholder knew where to get the information. To tackle this problem, it was then the idea of visualization was introduced which would, in turn allow a stakeholder to get a quick overview of the needed information when interacting with the Database. After we determined that the visualization was needed, we then proceeded with design of the automated visualization prototype as the first step in responding to this problem, this corresponds to the cycle 1 of the applied AR methodology in this thesis work. We conducted several interviews in cycle 2 of the applied AR methodology in order to get more needs towards the visualization of the Database. We followed the qualitative research methodology (Coding) to analyse the data collected from the stakeholders. The stakeholders that we interviewed were of different roles and they all had different experiences on the use of the Database. Their experiences relied on the duration of how long they have been using the Database and also the frequency of how often they have been using it.

This study has shown that automated visualization of data has some advantages and one of them being, it provides a quick overview of the relationship between artifacts stored on the Database. The automated visualization of data was done for the first time at the company and it was fun and interesting to see the emerged needs after the prototype was presented to the stakeholders. The prototype was designed to demonstrate how the visualization can be done and it was well interpreted by different stakeholders. The designed prototype did not fully cover the suggestions that were given by the first stakeholder that we met at the company and that was because of difficulties of getting the data from the company.

The study has also provided different views of how the data can be visualized which were the logical and the physical view. Each of these views have a different way

of presentations, the physical view can be represented using a UML deployment diagram and the logical view remains in UML component diagram as it was done on our prototype. The use of UML notation was beneficial to us in a sense that, we had applied less effort on explaining our visualization output, especially when explaining artifacts such as components (LCs) and the signals (provider and require). However, the stakeholders requested for more information needs to be represented in the visualized diagram which in turn may be a limitation to the use of UML diagrams.

The second cycle of the applied AR methodology was done half way and that was due to the limited amount of time that we had, the proposed implementation of the discovered needs from different stakeholders will be discussed in Section 8.2.

8.1.1 Sustainability

One of the new subjects in software engineering process in the past several decades is how a software development process and the product itself have an impact on the environment in the present and the future. During the thesis work we took this concern into account which aspects of the automated visualization could improve the sustainability by considering several actions described in [18] by Birgit Penzenstadler.

The first action which the author suggested is, the use of a common artifact model for documenting during development process. Our automated visualization was aimed to facilitate the way of working with the electrical architecture of cars by providing diagram overview of the connections among the artifacts within a sub-system. After it receives further development and is ready to be used in VCG, we hope that the visualization can be used as a common source to be documented during the initial development process, and to be a source that provides support for decision making.

The second action we considered is, the consideration of working offline for certain tasks. One of the main activities when the stakeholders working on a specific task was that he/she had to look for needed information on the Database which was quite slow due to problems with the server or the tool itself. With our automated visualization, it would speed up the act of looking for information since the tools used for creating the visualization does not require the Internet.

The third action, which is the optimization of resource usage during software development tasks, is similar to the second one. Not only the automated visualization would speed up the act of looking for information, but also it would decrease the network traffic, reduce resource usage, and energy consumption as the automated visualization can be done on local computers without connection to server.

8.2 Future work

In every study process, there can be some limitations. Previously, we had knowledge in software engineering domain. This was our first time to experience the automotive domain, we have to admit that we have come a long way to reach this point. In the beginning of the thesis, we had a bit of trouble on grasping the terms and what they meant but it is something we came along with as time continued and this has enabled us to get a better understanding in both of these two domains. As the matter of fact, it has been interesting to learn the roles of software in automotive domain and how the use of software in cars has been growing from time to time. In this section, we will discuss on what we propose to be done in the next study towards the automated visualization.

As discussed earlier, we applied the AR methodology which can be done in many iterations. The next step to where we have stopped is the take action step which is the implementation of the analyzed and interpreted data. It will involve the implementation of the categories of needs of stakeholders presented in section 5.3.

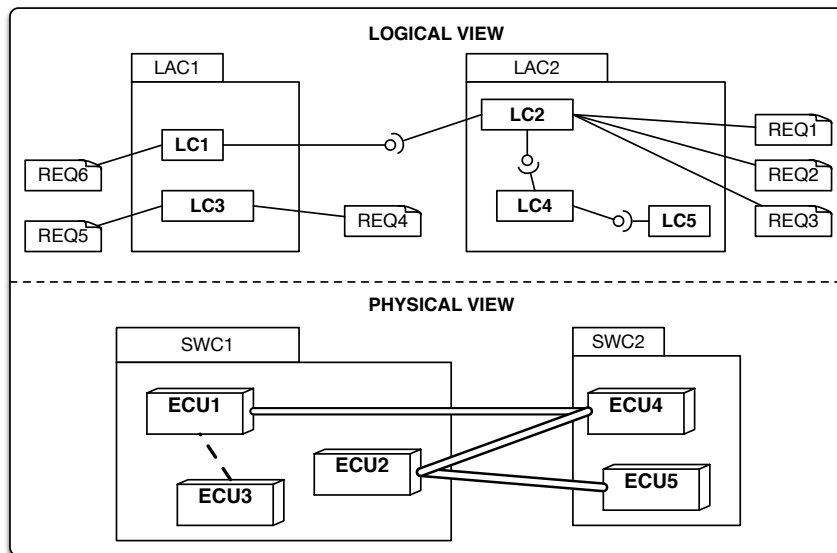


Figure 8.1: The sketch of a logical view and a physical view.

The categorized needs of stakeholders appears in different views/visualization as described by Dajsuren [8] also covered in Section 2.1 and on Figure 2.3. In this section, we propose what to be added on the logical view which we started its implementation on the first cycle and also we propose what to be included on the physical view. The prioritized lists of actions to be taken are:

1. We already started on implementing a logical view of a single subsystem, it would be good idea to use our code on other subsystems and the whole system at large. The positive side effect of applying our code on the large scale is that it may mitigate the construct validity threats (mono-operational bias) discussed in section 7.1.3. It is expected that the output diagram will be even messier than our first visualized diagram 5.1. The logical view can have

additional artifacts which are:

- (a) Include the LACs
- (b) Connection among LACs
- (c) Requirements of each LC
- (d) Showing the ID for each requirement

The upper part of the figure 8.1 shows the example of the expected part of the logical view.

2. The visualization of the ECUs, both at the system level and at the sub-system level is one of the need that we have discovered. This type of visualization covers the hardware view of the electrical architectures. This specific type of view was mentioned by the first software developer who gave us the scope of visualization, the second software developer and the test engineer. The lower part of the figure 8.1 shows the expected sample of the physical view that we proposed. The artifacts to be shown on the view are:
 - (a) Bus and signals
 - (b) Show the failed ECUs
 - (c) Different colors on bus types
 - (d) Connection among ECUs
 - (e) Connection among SWCs
3. It was proposed to combine both physical view and logical view within the same diagram by one of the stakeholder, but again, one has to try out first to see if the resulted diagram is comprehensible as compared to splitting the two views of the electrical architecture.
4. The next study can also apply different approach to visualize the architecture so that stakeholders can get a more dynamic visualization. This type of visualization can provide the ability to do certain things on the visualized diagram such as the filtering, sorting, select and deselect to view details (figure 5.17). Perhaps a 3D visualization can be something more interesting. The implementation of this type of dynamic visualization will also responds to the suitability quality the tool that will do such a visualization.
5. The next cycle which is proposed to be done after the implementation of the categories of needs of stakeholders that we have interpreted in cycle 2 will start again with the data collection step which is usually the first step when beginning the cycle of AR methodology. In this step, it is highly recommended to consider a large scope of stakeholders, both the primary stakeholders and the secondary stakeholders. It is very important to find out what impacts does the automated visualization have to not only the stakeholders who directly interacts with the database but also the ones who does not directly interacts with the Database. This way, the external validity threats explained in section 7.1.4 may be mitigated.

Bibliography

- [1] Basili, V. and Weiss, D. (1984). A Methodology for Collecting Valid Software Engineering Data. *IEEE Transactions on Software Engineering*, SE-10(6), pp.728-738.
- [2] Brambilla, M., Cabot, J. and Wimmer, M. (2012). *Model-driven software engineering in practice*. [San Rafael, Calif.]: Morgan & Claypool.
- [3] Cánovas Izquierdo, J., & Cabot, J. (2013). Discovering Implicit Schemas in JSON Data. *13Th International Conference, ICWE 2013, Aalborg, Denmark, July 8-12, 2013. Proceedings*.
- [4] Community Tool Box (2016). *Chapter 7. Encouraging Involvement in Community Work*. Retrieved from <http://ctb.ku.edu/en/table-of-contents/participation/encouraging-involvement/identify-stakeholders/main>
- [5] Cohen, L., Manion, L. and Morrison, K. (n.d.). *Ethic Issues In Research*. Retrieved from cw.routledge.com/textbooks/cohen7e/data/Chapter5.ppt
- [6] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, Anders Wesslén. (2000). *PLANNING. Experimentation in software engineering*. Kluwer Academic Publishers, pp. 66-73.
- [7] Czarnecki, K. and Helsen, S. (2006). Feature-based survey of model transformation approaches. *IBM Syst. J.*, 45(3), pp.621-645.
- [8] Dajsuren, Y. (2013). Automotive Architecture Description and Its Quality. *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pp.727-730.
- [9] Darvas, A. and Konnerth, R. (2016). System Architecture Recovery based on Software Structure Model. *Working IEEE/IFIP Conference on Software Architecture*, pp.109-114.
- [10] Eliasson, U., Heldal, R., Pelliccione, P., and Lantz, J. (2015). *Architecting in the Automotive Domain: Descriptive vs Prescriptive Architecture*. Software Architecture (WICSA), 2015 12th Working IEEE/IFIP Conference on. IEEE.
- [11] Eliasson, U., Martini, A., Kaufmann, R., and Odeh, S. (2015). *Identifying and visualizing Architectural Debt and its efficiency interest in the automotive domain: A case study*. Managing Technical Debt (MTD), 2015 IEEE 7th International Workshop on. IEEE.
- [12] Favre, J.M. & Cervantes, H. (2002). *Visualization of Component-based Software*, 6(2), pp.205-219.

- [13] Glanz, J. (1999). Action Research (pp. 22-23). Retrieved from <http://www.education.vic.gov.au/Documents/school/teachers/profdev/actionresearchjglanz.pdf>
- [14] Grönniger, H., Hartmann, J., Krahn, H., Kriebel, S., Rothhardt, L., and Rumpe, B. (2008). View-Centric Modeling of Automotive Logical Architectures. *Tagungsband des Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme IV*.
- [15] Lauesen, S. (2002). Elicitation. *Software requirements: Styles & Techniques*. Harlow: Addison-Wesley. pp 338-339 and pp 350- 351.
- [16] Niesel, K. (2014). *Software center day: The All-New XC90* [PowerPoint slides]. Retrieved from http://softwarecenter.gu.se/digitalAssets/1506/1506474_sw-center---sk--ne-volvo.pdf
- [17] OnlineQDA (2016). Online QDA - *How and what to code*. [online] Available at: http://onlineqda.hud.ac.uk/Intro_QDA/how_what_to_code.php [Accessed 2 April 2016].
- [18] Penzenstadler, B. (2012). Supporting Sustainability Aspects in Software Engineering. In: *3rd International Conference on Computational Sustainability (CompSust'12)*. [online] Available at: <http://www.computational-sustainability.org/compsust12/papers/24.pdf> [Accessed 30 May 2016].
- [19] Sjoberg, D., Dyba, T., and Jorgensen, M. (2007). The Future of Empirical Methods in Software Engineering Research. *2007 Future Of Software Engineering*, 358-378. <http://dx.doi.org/10.1109/FOSE.2007.30>
- [20] Stachowiak, H. (1973). *Allgemeine Modelltheorie*. Wien, New York: Springer-Verlag.
- [21] Stuurman, G. (2010). *Action Semantics applied to Model Driven Engineering* (Master Thesis). University of Twente.
- [22] Tichy, M. *Model-Driven Engineering (MDE) Lecture 3: Meta-Modelling* [PowerPoint slides]. Retrieved from <https://pingpong.chalmers.se/courseId/4401/>.
- [23] von der Beeck, M. (2006). Development of logical and technical architectures for automotive systems. *Softw Syst Model*, 6(2), pp.205-219.
- [24] Wallin, P. (2008). *Key Challenges in Decision Making for Automotive E/E Architectures* (Licentiate Thesis, Mälardalen University, Västerås, Sweden). Retrieved from <http://www.diva-portal.org/smash/get/diva2:127004/FULLTEXT02.pdf>
- [25] ISO/IEC FDIS 9126-1, *Information technology — Software product quality — Part 1: Quality model*

A

Acceleo templates

A.1 Acceleo template for the automated visualization prototype

```
1 [comment encoding = UTF-8 /]
2 [module generate('http://jsonDiscoverer/discovered/SubSystem' )/]
3
4 [template public generateElement(subsystem : SubSystem)]
5 [comment @main/]
6 [file (subsystem.elementName, false, 'UTF-8')]
7
8 @startuml
9 skinparam nodesep 80
10 skinparam ranksep 80
11 artifact [subsystem.elementName.replaceAll(' ', '')/] {
12 [for (lc:HasLC | subsystem.hasLC)]
13 ['[/][lc.elementName.replace(' ', '')/]['/']/]
14 [/for]
15
16 [for (lc:HasLC | subsystem.hasLC)]
17 [for (port:HasPort | lc.hasPort)]
18 [for (lc2:HasLC | subsystem.hasLC)]
19 [for (port2:HasPort | lc2.hasPort)]
20 [if ((port2.dataElement = port.dataElement) and not (port2.
    elementName = port.elementName) and not (lc.elementName = lc2.
    elementName)) and (lc2.elementName.replace(' ', '').substring(3).
    toInteger() > lc.elementName.replace(' ', '').substring(3).
    toInteger()))]
21 [if not (port.IODirection = port2.IODirection)]
22 [lc.elementName.replace(' ', '')/] "[port.elementName.substring(6)
    /]['-'/][port.dataElement.substring(11)/]" [if (port.IODirection
    = 'REQUIRE')]#--([elseif (port.IODirection = 'PROVIDE')])#--0[/if
    ][if (port2.IODirection = 'REQUIRE')])--#[elseif (port2.
    IODirection = 'PROVIDE')]0--#[/if] "[port2.elementName.substring
    (6)/]['-'/][port2.dataElement.substring(11)/]" [lc2.elementName.
    replace(' ', '')/]
23 [elseif (port.IODirection = port2.IODirection)]
24 [/if]
25 [/if]
26 [/for]
27 [/for]
28 [/for]
```

A. Acceleo templates

```
29 [/for]
30 }
31
32 @enduml
33
34 [/file]
35 [/template]
```

Listing A.1: Acceleo template for generated textual description in PlantUML language

A.2 Acceleo template for the final version of the automated visualization

```
1 [comment encoding = UTF-8 /]
2 [module generate('http://jsonDiscoverer/discovered/SubSystem')]
3
4 [template public generateElement(subsystem : SubSystem)]
5 [comment @main/]
6
7 [file (subsystem.elementName, false, 'UTF-8')]
8
9 @startuml
10
11 left to right direction
12
13 skinparam shadowing false
14
15 skinparam component{
16   backgroundColor lightGrey
17   borderColor black
18 }
19
20 skinparam rectangle {
21   backgroundColor white
22   borderColor white
23   fontSize 11
24 }
25
26 package "[subsystem.elementName.replaceAll(' ', '')]" {
27   [for (lc:HasLC | subsystem.hasLC)]
28   [for (port:HasPort | lc.hasPort)]
29   rectangle [port.dataElement.replace('-', '').replace(' ', '')/]
30   [/for]
31   [/for]
32
33   [for (lc:HasLC | subsystem.hasLC)]
34   [for (port:HasPort | lc.hasPort)]
35   [if (port.IODirection = 'PROVIDE')]
36   [ '[' / ] <size:13> [lc.elementName.replaceAll(' ', '')] [ ' ] / ] -down-0 [
       port.dataElement.replace('-', '').replace(' ', '') / ]
```

```
37 [/if]
38 [if (port.IODirection = 'REQUIRE')]
39 ['[/]<size:13>[lc.elementName.replace(' ', '')[/]]'[/] -up- ( [port.
    dataElement.replace('-', '')[/]]'[/])
40 [/if]
41 [/for]
42 [/for]
43
44 }
45 @enduml
46
47 [/file]
48 [/template]
```

Listing A.2: Acceleo template for generated textual description in PlantUML language

B

Interview questions and transcripts

B.1 Interview questions

Interview Questions (30-60 min each)

| Goal of the question | Question | Related RQ |
|--|---|------------|
| Getting to know interviewee | <ul style="list-style-type: none">- Who is the person? position? responsibility? | ** |
| Emphasizing on understanding the use of the Database | <ul style="list-style-type: none">- How long has the person worked with the Database?- How often does the person use the Database (per day or week)?- What does the person think about the Database in terms of advantages?- What are the problems that the person has faced when working with the Database? | ** |
| Identifying the needs of stakeholders | <ul style="list-style-type: none">- What are the tasks that a person has been working on recently? (1 task is enough).- What does a person have to perform in the Database in order to complete the task?- Why does the person need to see the information of the components? | 1, 2 |
| Opinions on the automated visualization prototype | <ul style="list-style-type: none">- What does the person think about having automated visualization of the electrical architecture?- What advantages and disadvantages does the person envision about this? | 1, 2 |
| Research questions (RQ) [1] What are the needs of different stakeholders towards the visualization of the electrical architectures? [2] How does the automated visualization satisfy the needs of stakeholders? | | |

Figure B.1: List of pre-determined questions.