Recovering and Modeling Sensor and Actuator Architecture in Automotive Embedded Systems

HOJAT KHOSROWJERDI
SORIN DAN TATAR

KTH Industrial Engineering
and Management

Master of Science Thesis
Stockholm, Sweden 2014

KTH Industrial Engineering
and Management

Recovering and Modeling Sensor and Actuator Architecture in Automotive Embedded Systems

HOJAT KHOSROWJERDI, SORIN DAN TATAR

Master of Science Thesis MMK 2014:77 MES 007
KTH Industrial Engineering and Management
Machine Design
SE-100 44 STOCKHOLM

KTH Industriell teknik och management

Examensarbete MMK 2014:77 MES 007

Återhämtning och Modellering av Sensor- och Ställdons Arkitektur i Automotive Inbyggda System

Hojat Khosrowjerdi, Sorin Dan Tatar

Godkänt	Examinator	Handledare
2014-09-02	Martin Törngren	Xinhai Zhang
	Uppdragsgivare	Kontaktperson
	Scania CV AB	Mattias Nyberg

Sammanfattning

Under det senaste årtiondet har mängden kod som används i fordon ökat exponentiellt. På grund av detta skiftar bilindustrin mot att vara software-intensive. Som i de flesta mjukvaruintensiva branscher, drivs systemets utveckling i snabb takt av marknadens krav. Återanvändningen av värdefull legacy-code är en effektiv metod för att minska tiden till marknaden. Vid Scania är mjukvaruutveckling främst baserat på en omfattande legacy plattform. I detta sammanhang är det nödvändigt för systemförståelse, återanvändning, underhåll, systemverifiering och säkerhetsanalys att upprätthålla en omfattande beskrivning av mjukvaruarkitekturen. Men för att skapa en sådan beskrivning behövs ytterligare resurser, och det är svårt att upprätthålla följdriktighet med föränderliga implementationer.

Ett sättet att lösa detta problem är Reverse Engineering. Mjukvaruarkitekturen kan hämtas automatiskt från inbäddad källkod och presenteras på ett sätt som är specifikt för domänen.

Denna avhandling är en del av ESPRESSO-projektet. En del av ESPRESSO-projektet går ut på att återvinna lastbilens mjukvaruarkitektur från källkoden. Syftet med detta arbete är att utöka täckningen av arkitektursåterhämtningen genom att lägga till kopplingar mellan hård- och mjukvara. För att uppnå detta har en hårdvarumodell, inspirerad av EAST-ADL hårdvaru-meta-modell, utvecklats och använts i den befintliga infrastrukturen. Hårdvarumodellen användes för att samla in och bearbeta information för att lagra den i Neo4J grafdatabas. Förslag på användargränssnitt tillhandahölls för interaktion med databasen, men implementationen var inte en del av examensarbetet.

Utmaningarna under arbetets gång uppstod främst på grund av det faktum att varje Scania-avdelning använder sin egen partiella systemmodell av lastbilsarkitekturen. Flera vyer och begrepp från olika avdelningar skulle slås samman i en enda modell. För att uppnå validering till en viss grad, användes databasen i samband med användargränssnittet. Gränssnittet var medelvärdet med vilket några scenarier kontrollerades både mot intern teknisk dokumentation och ingenjörer som arbetar med dessa system.

KTH Industrial Engineering and Management

Master of Science Thesis MMK 2014:77 MES 007

Recovering and Modeling Sensor and Actuator Architecture in Automotive Embedded Systems

Hojat Khosrowjerdi, Sorin Dan Tatar

Approved	Examiner	Supervisor
2014-09-02	Martin Törngren	Xinhai Zhang
	Commissioner	Contact person
	Scania CV AB	Mattias Nyberg

Abstract

From the past decade onward, a trend has been seen in which the amount of code used in a vehicle is increasing exponentially. Because of this growing factor, the automotive industry is gradually shifting towards software-intensive. As in most software-intensive industries, the system's evolution is driven at a fast pace by the market's requirements. The re-usability of valuable legacy code is an effective method of reducing the time to market. In Scania, software development is predominantly based on an extensive legacy platform. In this context, maintaining a comprehensive software architecture description is necessary for system understanding, re-usability, maintenance, system verification and safety analysis. However, to develop such a description involves additional resources, and it is difficult to maintain consistency with evolving implementations.

One way to solve this problem is Reverse Engineering. The software architecture can be retrieved automatically from embedded source code and presented in a manner specific to the domain.

This thesis is part of the ESPRESSO project. One part of ESPRESSO is to recover the truck's software architecture from source code. The objective of this work is to extend the coverage of the architecture recovery by adding connections between hardware and software. To achieve this, a hardware model, inspired by the EAST-ADL hardware meta-model, has been developed and employed in the existing infrastructure. The hardware model was used to gather and process information in order to store it to the Neo4J graph database. User interface suggestions were provided for querying, but the implementation was not part of the thesis.

The challenges facing this work arose mainly due to the fact that each Scania department uses its own partial system model of the truck's architecture. Multiple views and concepts from different departments had to be merged in a single model. To achieve validation to a certain degree, the populated database was used in connection with the user interface. The interface was the mean by which a few scenarios were checked both against internal technical documentation and the engineers that are working with those systems.

Acknowledgments

Hojat Khosrowjerdi and Sorin-Dan Tatar

Stockholm, August 2014

We would like to thank our KTH supervisor Xinhai Zhang for being always available to discuss issues, for guiding us and for providing valuable feedback and support. We would also like to thank Mattias Nyberg for being our supervisor at Scania and for the support he gave us throughout the thesis work.

We appreciate the work with Magnus Persson, Jonas Westman and Daniel Hilton, which brought us closer to the results of our work.

Finally, we want to address special thanks to our examiner Martin Törngren for valuable suggestions towards improving our work.

Nomenclature

The abbreviations used in the thesis report are listed here.

General purpose

ADL	Architecture Description Language
API	Application programming interface
AR	Architecture Recovery
ASIL	Automotive Safety Integrity Level
AUTOSAR	Automotive Open System Architecture
CAN	Controller Area Network
CPU	Central Processing Unit
E/E	Electrical and Electronic
ECU	Electronic Control Unit
HSI	Hardware software interface
HW	Hardware
I/O	Input/Output
ISO	International Organization for Standardization
SAR	Software Architecture reconstruction
SW	Software
XML	Extensible Markup Language

Scania specific

AB	Architecture Browser
APPL	Application layer
BCI	Bodybuilder Interface
CMS	Chassis Management System
COMP	Common Platform
COO	Coordinator
EMS	Engine Management System
EOL	End of Line Parameters
FPC	Functional Product Characteristics
LLAP	Low Level Application layer
MIDD	Middle layer
NCG	New Cab Generation
NGS	New Generation Scania trucks
OAS	Object And Structure tool
R&D	Research and Development
RTDB	Real-Time Database
SESAMM	Scanias electrical system
SDP3	Scania Diagnos & Programmer 3
SOPS	Scania Onboard Product Specification
SPECTRA	Scania's proprietary product configurator

UF	User Function
VIN	Vehicle Identification Number
VSEN	Virtual Sensor layer

Contents

1. Introduction

This chapter describes the problem, motivations and goals defined for the thesis work. Further it identifies potential stakeholders and investigates previous related works. Finally it defines the delimitation and the methodology of the research study.

With the advancement of electrical and electronic control units in automobiles, the automotive industry has started a new era in its history. Many manual control systems in cars have been replaced with modern automatic ones. Also, new features have been introduced in response to new technological innovations. Although handling a vehicle has become simpler thanks to automation technology, new control systems added to the vehicle make it even more complex. Modern cars are equipped with numerous electronic control units (ECUs) to perform desired functionality (e.g. braking, steering, cruise control, fuel and ignition control, etc.). This means that vehicles have become not only dependant on embedded systems, but also vulnerable to existing defects in software and hardware components. Therefor, with millions of code lines in a vehicle's embedded systems, the probability of failures is not anymore negligible. This is where the bugs in the ECU software can become a threat to all road users.

In addition, the trend toward modularization and reusability in automotive industry leads to the introduction of shared parts and components. Hence, engineers reuse legacy code and extend it to customize products according to the customer requirements. Based on the theory from the area of embedded software [1], adding even a simple task to an already working ECU software, does not guarantee the lack of deadline misses. Therefore, safety matters can degrade when the software evolves.

To avoid accidents resulting from any of the aforementioned issues, the International Organization for Standardization (ISO) distributed a safety standard called ISO 26262 [2]. The aim of this standard is the functional safety of E/E systems and the targeted group is road vehicles. Even though ISO 26262 is not defined for heavy vehicles yet, large enterprises strive to comply their technical descriptions and workflow with it. Scania, as a representative, has taken the chance to investigate the possibilities of mapping its in-house development methods to the requirements of the ISO 26262. The ESPRESSO project is Scania's initiative to investigate the gap towards ISO 26262. Research projects including PhDs and Master theses have been defined within this purpose, resulting in a series of tool chains. Each of these tool chains is designed to fulfill one of ESPRESSO's goals, e.g. Architecture Browser can display visually to the end user the stored architectural information and it can also filter to different layers of abstraction. However, all of them rely on the overall architecture since the ISO 26262 requirements have to be linked to architectural elements.

This thesis is part of the ESPRESSO initiative because it builds and extends previous work performed in this context. The aim is to extend the coverage of

the work done, by introducing information on connections between hardware and software. Because this was not considered in previous work, the thesis can be seen as an isolated project. The goals are however meaningful if the context of ESPRESSO is taken into account.

Since an automotive embedded system is comprised not only out of the ECU software, but also out of its hardware parts, the issue was put forward on how to add the hardware aspect to the overall architecture [2, 3]. Once the hardware structure is made available, it will provide a better comprehension of how the system is constructed from this viewpoint. In other words, the outcome of the work will supply developers with an improved systems understanding, better analysis and debugging. For this purpose, our work handles the lack of hardware information within the main project. Because ISO 26262 also covers hardware development, the outcome of this thesis will improve the understanding of the challenges Scania will have towards adopting the ISO. On the other hand, adding hardware structures will make it possible for engineers to trace possible issues from software to E/E components. Another interesting use-case will be the possibility to trace between HW/SW and requirements.

Complexity in Automotive Embedded Systems

Complexity of automotive embedded systems is increasing dramatically with the constant introduction of new advanced features and functionalities every year [4]. On one hand, technological innovations add large numbers of new hardware components with increased performance and capabilities to the list of utilizable parts for vehicles. On the other hand, the ECU software that handles many operations such as interacting with hardware components, analyzing the huge amount of real-time information, scheduling tasks to meet deadlines, communicating with other ECUs, and executing multi-dependant applications has become even more complex. The increasing trend in the number of code lines being produced for numerous functionalities during the last decade (See Figure 1.1) is also the result of rising complexity in the automotive embedded systems [5].

Another major source of complexity lies behind variability. Variability comes with modularity and standardized interfaces. While customizable products are achieved by means of a few specific entities, the rest of components are composed of shared elements. Having introduced the variability in the system, the software should deal with many different possibilities. As the products change depending on the customers requirements, the same software is employed to control the system. Therefore, immense number of code lines are written not only to handle various types of I/Os, but also to consider which module is active and is implemented in the system [7].

As the Scania situation proves, the complexity is introduced by new software modules as well as by existing legacy code that has to be interfaced to both new systems and to variants of this systems. This can be for example situations in which a coordinating ECU has to work with both old and new braking systems and their corresponding communication. A different situation is when the legacy system must handle newer systems introduced by new hardware, but also system changes dictated

by variability.

Another Scania specific attribute that results in increasing complexity is the in-house hardware and software development for its most important products. Based on several internal documents the total number of ECUs employed in Scania products can be approximated to be around 100. While this ECUs are used in all kinds of Scania's products, most of them are in-house. In addition, the truck in itself can be seen as a distributed system of ECUs that have their own specific tasks but communicate through CAN buses to perform multi-dependant processes all at the same time. Figure 1.2 shows the distribution and placement of standard ECUs in an ordinary Scania truck. Taking into account that a normal truck has over 30 ECUs or components that accomplish such functionality, it is obvious what the benefits would be from a unified database of the architecture.

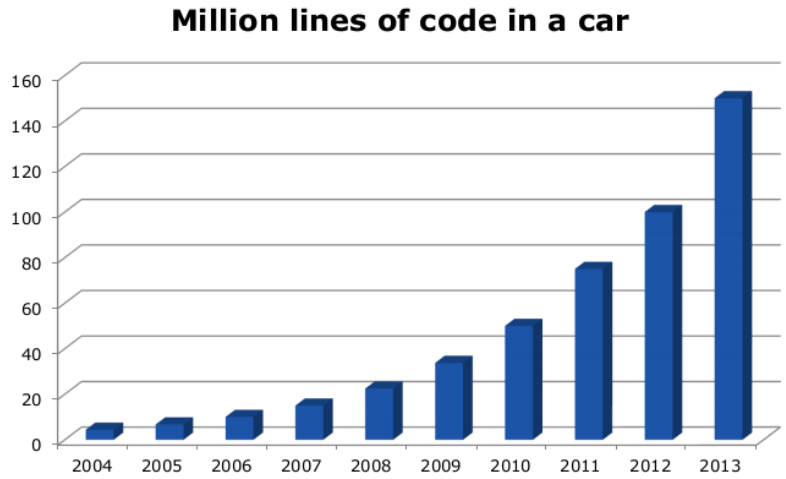

Figure 1.1: Trend of code complexity in embedded software during last decade [6]

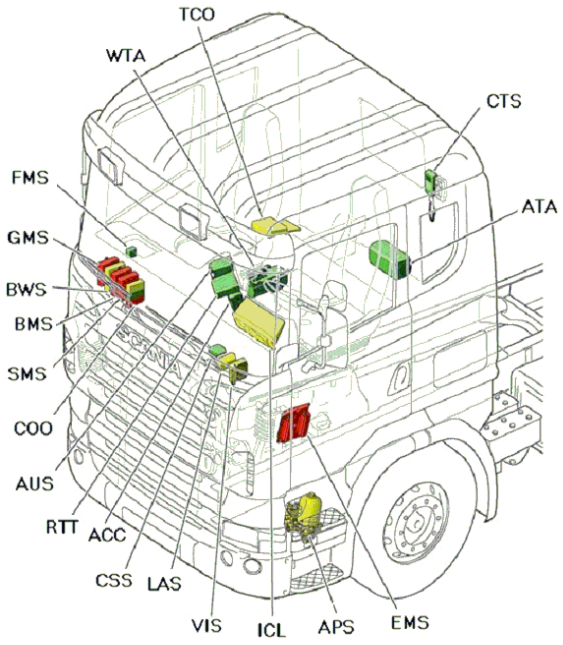

Figure 1.2: Distributed ECUs in a normal Scania truck [8].

1.1 Organization of the thesis

This section describes briefly the contents of the thesis report and specifies division of work. The main chapters are:

1. Introduction
   Introduces to the context of the problem by providing the problem's description, the motivation and goals of the project and describing the interested stakeholders. It briefly presents the work that has been accomplished.

2. Background
   Provides the theoretical background for methods and technologies used with the purpose of achieving the project's goals. It also describes the context in which the thesis project handled.

3. Modeling and Recovery
   This chapter connects first of all the theory by explaining how to use it in order to achieve the goals. The second part deals with describing the methodology and analysing the created model. In the end explains the tool chain implementation.

4. Results and Analysis
   Deals with presenting the results by displaying and explaining the retrieved and persisted information. It also presents screen-shots from the front-end of the ESPRESSO tool chain, Architecture Browser, with the results of this thesis.

5. Discussion
   Debates on the results and the challenges encountered. Describes what approaches could have been done better.

6. Conclusion and future work
   Provides clear answers to the problems of this work and offers recommandations for future work.

7. References

Next, the division of work is described in regard to the written part of the thesis, the report. It is to mention that both students were involved in finalising the report and both contributed to every section, but some areas were handled mainly in an individual manner:

Hojat Khosrowjerdi:

Chapter 1: 1.1-1.4,1.7

Chapter 2: 2.2, 2.3

Chapter 3: 3.2, 3.3

Sorin-Dan Tatar:

Chapter 1: 1.1-1.3,1.5, 1.6

Chapter 2: 2.1

Chapter 3: 3.1, 3.4

Both authors have dedicated equal amounts of work for writing chapters 4 through 6.

From the implementation perspective, the tasks were individually managed, but the work in it's own could not be split clearly between the two students. For an efficient workflow the tasks had a main responsible while the second student had the responsibility to check the implementation and results. An overview of the work is described in the following.

Both students developed the plan for the project and the basic structure for the implementation. The theoretical research topics were distributed between the students, but the creation and integration of the model, together with data sources analysis was a shared task.

Hojat Khosrowjerdi started with the analyses of the the COO8 source code and determined the software layers that are of interest. His analysis of the code was used for the pattern matching to retrieve the relevant information. In a similar approach he analysed COO7 with the same result. For EMS S8 he implemented separately in Java the patterns and information retrieval mechanism which was integrated in the tool chain.

Both students were implicated in researching different sources of information and deciding which were usable and in what way. The PT and component library were utilized with the work of both authors.

Sorin-Dan Tatar wrote in Java the basic framework for the tool chain. He used software architecture recovery methodologies in Java for retrieving information from COO8 and COO7 based on the provided analysis. He integrated the provided EMS S8 patterns into the main toolchain. The BCI1 and CMS1 analysis and recovery implementation was done together. Sorin-Dan Tatar wrote the implementation of the cable list parser and integrated it with the main work.

The students put an equal amount of work in analysing the results and discussing them for the purpose of drawing the conclusions.

1.2 Problem statement and description

A common problem in the automotive industry refers to the assurance of functional safety, especially in E/E systems that control vehicle behaviour. In order to standardize the implementation of functional safety in E/E systems, ISO 26262[2] was defined and poses in itself a set of problems that the industry has to solve. As a preparatory measure at Scania, the ESPRESSO project investigates the gap between ISO 26262 and current development practices at Scania. Similar to the case investigate in paper [3], this thesis is part of, builds upon and extends previous work in the ESPRESSO project. This thesis deals with some of the technical challenges that are deduced from ISO 26262's requirements, i.e. describing an overall architecture of electrical hardware components connected to the software. A constraint in the search for solutions is that Scania's way of work should not be altered since it is regarded as one of its main strengths as an industrial power.

A premise towards the investigation of the ISO 26262 is the overall system architecture. On one hand the overall architecture must be documented and mapped to the actual implementation. On the other hand, the architecture should be analysable from both software and hardware viewpoints, which is not always the case. These two concerns can be summed up into two main problem statements. The first issue in this thesis addresses architectural drift focused on the low level software layers. While the software architecture can be found through the analysis of source code, hardware description is spread throughout different data sources including source code. The second issue relates to the lack of architectural views that contain HW descriptions with links to software. The following are explanations of the problem statements, together with the research questions.

Question 1 How can the system architecture, focused on hardware description, be retrieved in order to compensate for architectural drift in Scania's environment?

Architectural drift appears often due to the divergence of the implementation from the system-to-be. Similarly, it occurs during the life-cycle of automotive embedded systems as for example when legacy code evolves without being documented. This issue has to be taken into consideration, especially at Scania where the majority of ECUs are being developed in-house.

Although architectural drift is a general problem, the solution has to come in a form that is customized to the needs of the end-users. This is also the case

with the software architecture recovery (SAR) which is one of the sought-after approaches for having a consistent connection between the documented architecture and the implementation. Even though SAR techniques have already been employed in ESPRESSO project, they have not focused on hardware description. This means that the low level ECU software responsible for handling ECU pins and peripherals is not recovered yet. In addition, the SAR approaches do not include global variables in architectural models. However, the customized nature of the first problem statement requires this work to investigate, how to retrieve the architecture part describing the connection from the global variables to the code entities that describe the pins of the ECUs.

The solution for this problem has to be composed out of the results from several investigations which are carried out within the case of Scania. These investigations are performed based on research questions that are further inferred from the first problem statement. For instance: what architecture recovery techniques are best suited for the requirements of this work, which information must be retrieved for the required functionality, and which data sources are available at Scania that contain the desired information. Furthermore, the research is directed towards the perspective that mainly the developers and architects had about the source code from the lower levels and which they expected to find in an overall system description. This in turn means that the solution starts by identifying which parts of the low level layer in the source code are of interest and in the end, how to relate different pieces of information to each other.

Question 2 How can an overall model be composed so that it describes both software and hardware elements?

Information regarding different aspects of the system are normally managed in different databases. However in most of the cases these databases are not interconnected. Therefore, system understanding becomes difficult, synchronization of all data becomes challenging and finding issues and bugs becomes complicated. Likewise, at Scania, different departments have different architectural views regarding only parts of the entire information. In other words, they utilise their own data sources and processes to develop part of the HW or SW they are responsible for. It is obvious how this can lead to slow processing for issues that involve multiple departments. What we reason here is that although the concept of an overall system architecture exists, the components for it are separated within each department's perspective. The outspread use and manual handling of information also adds inconsistency issues in the system architecture. This can be prevented to some degree with an overall architecture and meta-model created by composing the different views and models from each department.

The second problem statement refers to the lack of a unified, consistent and complete database that contains both HW and SW elements in one common architecture with the appropriate connections. This involves the answer to the question of how to describe the architecture with hardware and IO features into account. One of the primary points of solving this problem is to achieve an overall system understanding consistent with the implementation, which is also a premise to fulfil the safety standard. In addition, it would benefit the generation of impact analysis and fault trees.

However, there are many challenges behind the problem. For example, where and how is the required information available, or how to retrieve the desired information from different department databases. On the other hand, considering the existing meta-model and the retrieved SW information in the ESPRESSO database, also a primary topic is how to extend the meta-model to include and integrate hardware information with the recovered architecture.

Moreover, complementary to all the aforementioned questions, identifying the stakeholders of such functionality is always a challenge. Hence, suggestions should be made how to visualize the information so that it suits them best. Finally, the variability aspect has to be considered, but not implemented. In accordance with other ongoing theses, some variability points will be kept in mind such as the items of our work that lead to variability.

1.3 Motivation

ISO 26262 emphasizes that in order to avoid accidents and human injury, safety critical requirements should be met throughout design and implementation. One part of achieving this is to establish an understanding of the structural characteristics of the overall system.

There are different tools and languages that can describe an architecture in a high-level abstraction and which can help developers understand the overall structure rather than individual implementations. These tools are most of the time based on a top-down concept in which the software is first described on a high level abstract level then implemented. The problem in automotive industry is that it is cluttered with legacy code to which refactoring iterations have added more and more code to the extent that the originally envisioned architecture drifted or has been even lost compared to the latest implementation. In these cases SAR techniques are one of the preferred solutions for retrieving the high level structure and architecture of the projects.

The main driving force behind ESPRESSO can also be linked to the necessity of having a tool, for developers, which is able to follow the connection between implementation and architecture. The benefits of having a good architectural description implies a better understanding of the system and also a better control if change is needed. Another positively affected area is system analysis, in which case, dependencies, reliability, safety and other aspects can be analyzed. Architectural description also permits requirements decomposition, an indispensable part in a project's workflow. Another strong point will be the traceability between HW/SW and requirements.

The integration of information from different databases within Scania under a more consistent source can be seen as a prerequisite for conforming to the safety standards. Focusing from the main ESPRESSO project towards the thesis at hand, the benefits extend from a software only aspect to include also hardware elements. An overall architecture view is not necessarily limited to the source code description. Hence, visualizing the connected hardware can, under the right conditions, greatly facilitate code development by providing extra information.

1.4 Goals

From a broader perspective this work reflects the goals and aims of ESPRESSO. These can be summed-up to the following topics:

- aiding engineers by making the development more efficient in regards to time and cost
- producing safer vehicles
- becoming prepared to fulfill ISO 26262

In a more isolated context, this master thesis will focus on evaluating the possibility of retrieving, modeling and integrating a new HW architecture recovery module within the existing ESPRESSO project at Scania. The new module will deal with both the software and hardware of the sensors and actuators which are connected to the ECUs. The purpose is to improve the development process by visually presenting the HW/SW architecture. The aim is to use a single unified database as a source for the front-end presentation.

The first goal, which is developed from the problem statement, can in this sense be described as to define an abstract model for architecture recovery that able to also be populated with source code HW information and the accompanied connections through the software in the application modules. On a more detailed level and in the context of this work it can be described in terms of objectives from the first question of the problem statement:

- Extending the existing software architecture recovery with recovery of the pin description and their connection through the software modules
- Implementing the entire chain(data retrieval, processing and storing) in the current project for accomplishing this.
- Retrieving the right information that will support this functionality

The second question of the problem statement develops into a second goal and the underlying objectives. **The second goal** can be summarised as creating a meta-model that encompasses both software as well as hardware concepts together with the connection between them. This can be grouped in the following objectives:

- Defining the information needed for hardware architecture description
- Locating the desired information and evaluating its retrieval method
- Defining how the hardware elements should be presented visually
- Extend the meta-model for accepting hardware
- Integrating and mapping the hardware information to a graph database
- Provide suggestions for variability integration

1.5 Stakeholder analysis

In order to have a comprehensive view over the problem statement and motivation of this thesis, the parties that have an interest in the project are analysed in this section. These parties are referred to as stakeholders.

According to the ISO/IEC/IEEE 42010 Systems and software engineering - Architecture description [9] standard, a stakeholder is any party that has a relation towards the system at hand. Examples of stakeholders can comprise individuals, teams, organizations or combinations of the previous.

The interest that is relevant for a set of stakeholders is called concern. The concern can be related to any influence exercised on the system like social influence, legal, business related, technological related and so on. It can involve aspects regarding security, performance, structure, etc.

To be noted is the fact that the standard separates the concepts of architecture and architecture description. While the architecture of a system is described as the collection of properties of a system in relation to the environment it is in, the architecture description is a "work product" used to express an architecture. The architecture description depends on the definition of the stakeholder's concern which may refer to different purposes like analysis, testing, documenting, development and design, etc.

Using the standard[9] as a guideline, Figure 1.3 presents the situation of the stakeholders in this project as interpreted by the authors. The description of the stakeholders can be made according to the group to which they belong to in relation to the Architecture Browser(AB) tool chain. The AB tool chain is considered to include the results of this work among other modules.

1. Sponsors

    a) Scania

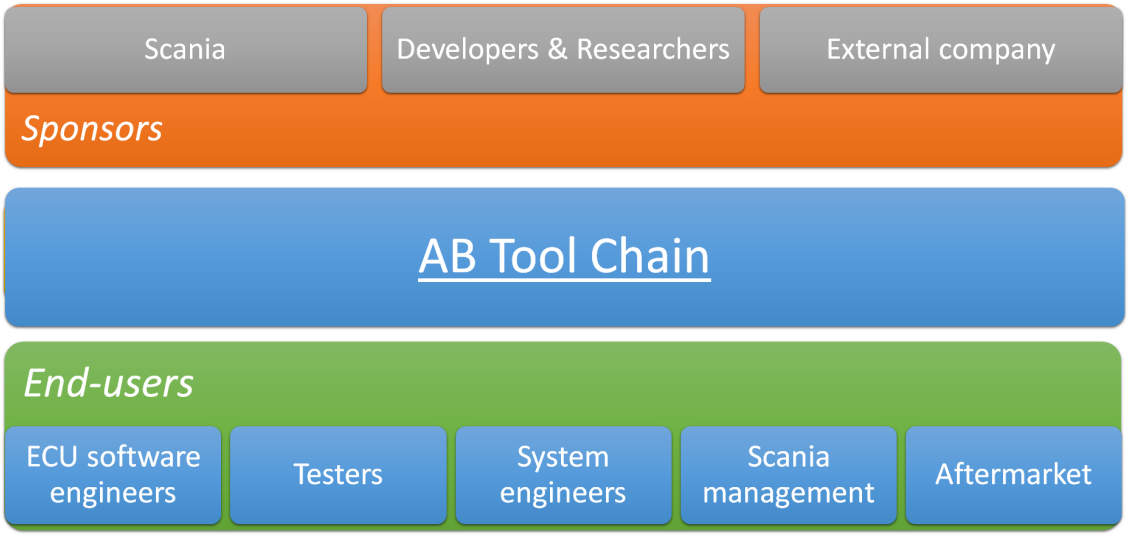

Figure 1.3: Stakeholders overview

Scania is the main party interested in the results of the ESPRESSO initiative and by extension, in the results of this work. A a high level interest for Scania is the gap analysis between ISO 26262 and the current situation.

b) Developers & Researchers
This thesis together with other theses and PhD researches are involved in the development of the Architecture Browser tool chain. A number of master and bachelor theses have been defined and are part of the AB tool chain and ESPRESSO initiative. In this sense, the students are interested the development and the theory behind this work. Besides theses, PhD researches are actively conducted within the ESPRESSO initiative.

c) External company
A third party is involved with sponsorship, but the information on this is not covered by this work.

2. End-users

a) ECU software engineers
Besides representing the end-users, this group was also used as a validation point. Discussions and interviews were initiated with them in order to understand their needs. Based on their needs the goals and expected results were reshaped appropriately. Their iterative involvement influenced the evolution of the meta-model and that of the implemented tool-chain. In other words, the ECU software engineers were involved mainly in influencing how the hardware to software architecture will be modeled in the meta-model. This group is also consulted to understand which software architecture elements are of interest and until what degree of abstraction the model should go. For this particular work, there was the need to understand to what level the connections between hardware and software should be made.

b) Testers
The testers group were not directly involved by the authors, but considerations were made when implementing towards the possibility of tracing signals and data-flow to some degree.

c) System engineers
This group was also not directly involved, but logically, they might benefit the most from an overall system architecture.

d) Scania management
Thought was also given that the management could use the high level abstraction of the meta-model to assess different parameters.

e) Aftermarket
In the aftermarket case, the results of this work can be used to asses in a different way variability on the electrical/electronical side.

In a bigger view, it was assessed that most of the time, each department uses its own model which contains part of the entire system and also only components about which the users are interested.

1.6 Previous work

This thesis is part of the ESPRESSO initiative and as such it was preceded by other theses which focused on different parts of the main goals and objectives. Besides this, it runs in parallel with other theses with which it will have to later integrate, but not in this work. In order to create a proper understanding of how this thesis project relates within a bigger view with the work done at Scania, the previous works will be briefly explained. This will be used to eliminate redundancy, but also to create constraints and limitations.

In [3], the focus is directed primarily on architecture recovery aimed towards CAN communication in the application layer of the ECU software. What has been achieved is a network diagram comprising the ECUs and the main CAN networks. Also, the authors have implemented a front-end tool in order to verify the recovered CAN communication against an external design database. The future work mentioned in the paper refers to variability, requirements and low level software layer coverage.

In addition, early in [10], the question was put whether or not it is possible to generate models of the hardware contained in the ECU systems automatically, but also how requirements could be modeled in regard to ISO 26262. The implementation was done on a constrained case study for a particular functionality of one ECU. The author mentions that further proof of concept needs to be considered in regards to adding testing and simulation to the hardware model contained in the ECU system.

Furthermore, Behrooz and Juan [11] worked on recovering the architecture related to the CAN messaging and it's connections throughout the the higher level software layers. Their work also takes the initiative of analysing the variability challenges in different ECUs for a possible architecture recovery solution implementation.

Oussama and Ahmed [12] tried to automatically recover the functional architecture of a truck's embedded software from the existing infrastructure at Scania. The emphasis is placed on the alignment of ECUs software architecture to AUTOSAR architectural models. The authors claimed to map *"from SCANIA application components to AUTOSAR application layer"* [12]. The authors also state that as a future work, investigation needs to be performed into *"including sensors and actuators connected to ECU in order to trace data flowing from/to peripherals"*. The authors of [12] present as a future work the retrieval and analysis of the scheduler in automotive embedded system. This in turn can be followed with analysis of data-flow penalties, critical path calculations, resource allocation, time optimization and CAN-bus load.

Lastly, Oscar [8] had as a primary goal in his work to generate a graphical representations of the ECU software architecture. This representation had to include data-flow and control-flow for compliance with ISO 26262 by means of architecture

recovery. As a second goal of the thesis, he proposed to develop a graph generation tool that will aid software development. His purpose was to help developers create software closer to the recommendations of the ISO 26262. He concluded with some themes for future work such as considering HW/SW interactions, time scheduling and dynamic aspect of the code.

1.7 Thesis delimitation and methodology

To identify the right method for the research, the problem statement, research questions and goals should be considered. In this thesis, the aim is to retrieve the system architecture of a Scania truck with the focus on hardware elements. It is also necessary to design an overall model for describing the software and hardware parts of the system. To achieve these goals, it is required to know what information is needed, where to find it and how to extract it from different Scania data-sources. Therefore, several investigations are carried out to collect this information which is of a descriptive or qualitative nature. In addition, the research is based on Scania's proprietary data sources which are limited and are only available within the company. Also, not every data-source is of interest in this thesis. Hence, we are not dealing with large quantities of data. According to this, the research work strategy is a case study within the Scania ESPRESSO project. Interviews and observation are the main methods used for the task of data collection. The interviews are performed with the end-users. The end-users, as mentioned before, are the engineers from different departments that have an interest in the project. Usually they belong to the departments from which the information is recovered and from which partial views and models are taken as examples. This task is important since a main part of the project involves studying Scania's workflow processes, ECUs source code and connection between different data sources.

After the necessary data sources have been investigated and relevant information has been retrieved either by means of architecture recovery or by other methods where source code is not involved, the aggregation process begins for the available data. The resulting elements are connected and stored in a common database that contains also information from previous processes. The final phase is to present visually the stored data with regard to user inputs. The user interface, created in another sub-project of ESPRESSO, is not developed in this thesis, but it is used as a feedback and validation element, instead of using the database interface.

The expected deliverable for the client can be viewed from the company's and the university's point of view. The following exemplifies at a very abstract level the basic properties of the deliverable. The company's concern is to receive a database in which hardware components are connected with the software layers in the form of a system architecture, a tool chain that can perform this process and a report that describes the project flow.

Due to the given time and manpower constraints the thesis also needs to define specific limitations the deliverable will be subjected to. Compromises were included in the project in order to reach the closure phase of the master thesis. The first limitation that was set concerns the type of architecture recovery that is performed on the source code. The complexity of the embedded code limited the investigation

to only static analysis, although hardware in the loop simulation is performed at Scania and could possibly be used for dynamic code analysis and architecture recovery. On a technical level, pointers were not fully investigate because of the high complexity level of the code and because it would reach the dynamic aspect.

Because of the limited time, the implementation had to be performed for only a small number of ECUs, i.e. : EMS S8, COO 8, COO 7, CMS1 and BCI1. Since the nature of a case study restricts the possibility of result validation and the general aspect of the investigation, feedback and test results were received mainly from the stakeholder groups.

2. Background

This chapter describes the existing knowledge that is gathered, studied and used in this project. The focus is directed towards presenting the Scania environment in relation to the specific processes it uses in its workflow and the proprietary concepts. The elements of importance will be emphasised together with their connections with the research questions. Also, a brief description towards ISO 26262 related aspects will be made. The last part of the background is dedicated for literature study on the topics of the research questions. The main topics of the literature study are architecture recovery, ADLs, model composition, graph databases and variability in brief.

2.1 Scania overview

Scania is a worldwide power in the automotive industry. It is specialized in heavy trucks, buses and engines used from general industrial applications to the marine domain. Scania has approximately 38600 employees worldwide from which around 9000 are working at the factory which is located in Södertälje, Sweden. For the year 2012 the sales amounted to 79603 millions SEK with an operating income of 8300 millions SEK. This amount along with considering the earnings for the first quarter of 2014 year (2257 millions SEK), shows the importance and extensive contribution of the prestigious company in the transportation industry [13].

As shown in Figure 2.1, Scania has a modular system for its productions which implies that a majority of vehicle and engine parts can be combined in various configurations. Thus, new developed components are able to be mounted besides other old parts in the vehicle or engine structure. This not only contributes to production efficiency, improved maintenance and service quality, but also improves customers satisfaction by providing customizable products [13].

Figure 2.1: Modularity in Scania Products [14].

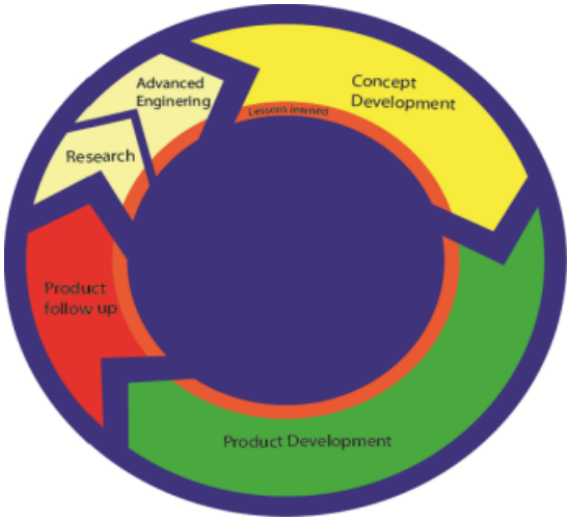

Figure 2.2: Scania Development Cycle [15].

2.1.1 Research and Development Center

Scania has a view that its products should be manufactured based on scientific methods, research and development. Hence, it can contribute both to improving vehicular technology as well as enhancing quality of its products. The research and development center (R&D) which is located in Södertälje, hosts more than 3000 employees with various background and specializations. While having tight cooperation with marketing organization to identify customer needs and requirements, research and development center collaborate with production departments in order to develop products and the technology behind them simultaneously. Considering present and future needs, R&D center attempts to minimize lead times and eliminate waste. Based on R&D factory concept [14] and according to Figure 2.2, the work is structured in three distinguished phases: Pre-development, Development and Product follow-up that are color coded with yellow, green and red colors respectively. Pre-development focuses mostly on research and finding new concepts and materials to fulfill future needs and requirements, and identify potential challenges that might be introduced to the transportation business. As its name suggests, Development phase comprises all processes from design and simulation to implementation and documentation that result in a new product. Finally, with Product follow-up, it provides the possibility to gather reviews, refine or redesign to improve functionality, safety and sustainability of products [14].

2.1.1.1 Research and Development organization

Figure 2.3 shows overall picture of the organization within R&D center. Apart from management and human resource offices, it is divided into three main departments, N, R and Y, in which their focus is on different systems of engine and vehicle. The N department develops Power-train which comprises transmission, engine and systems that provide power and control mechanisms for driving the vehicle. The R and Y departments are responsible for the development of truck, cab and bus chassis, and vehicle definition and product follow-up respectively [16]. Departments are divided

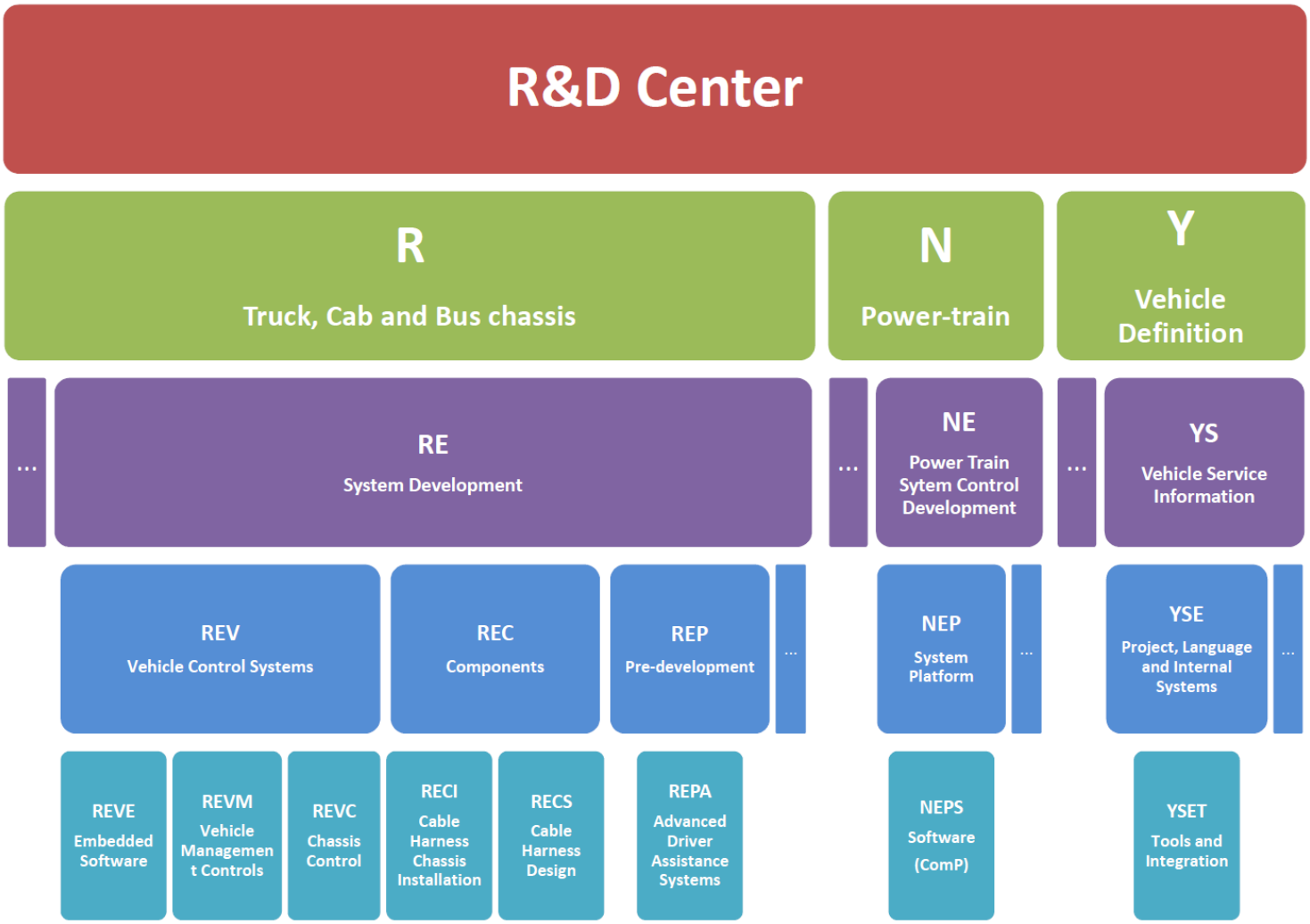

Figure 2.3: R&D Center Structure at Scania

into sections and groups according to the similar hierarchical models. For example, RE and NE are both System Development departments in their branch and distribute the work among RES/NES, REP/NEP and REV/NEV sections. However, these sections that have the same naming pattern in the two departments, have different responsibilities. As such, RES is system architecture and tools whereas NES is Engine Control Software. This thesis work is carried out within REPA group, a subset of R department. Here E stands for System Development, P for Pre-development and A for Advanced Driver Assistance System. In addition, some other sections of RE, NE and YS Units like REVE, NEPS and YSER that their work are related to this thesis work, have been investigated further. Those sections are identified with a brief description of their main work focus in the downside of the Figure 2.3.

2.1.2 Electrical systems

The E/E system for Scania vehicles are mostly developed and implemented in-house with limited exceptions which involve external suppliers. It composed of SW and HW components each of which is provided by a set of departments. Different departments have different perspectives and models of parts of the E/E system. For example, engine ECUs are developed in power train departments which concern only the parts that interact with the engine controller. Information regarding different aspects of the system is normally managed in various databases. However, not always these databases are synchronized.

The Electronic Control Unit (ECU) is an embedded system that handles a number of certain functionalities in a vehicle. The communication between ECUs is achieved in Scania trucks through Controller Area Networks (CAN). The CAN is one of the most important systems of communication, which connects, in the majority of cases, a coordinator ECU with the rest of the ECUs within a truck. Different lines of communication, buses, are described with different colors depending on their importance as it can be seen in Figure 2.4.

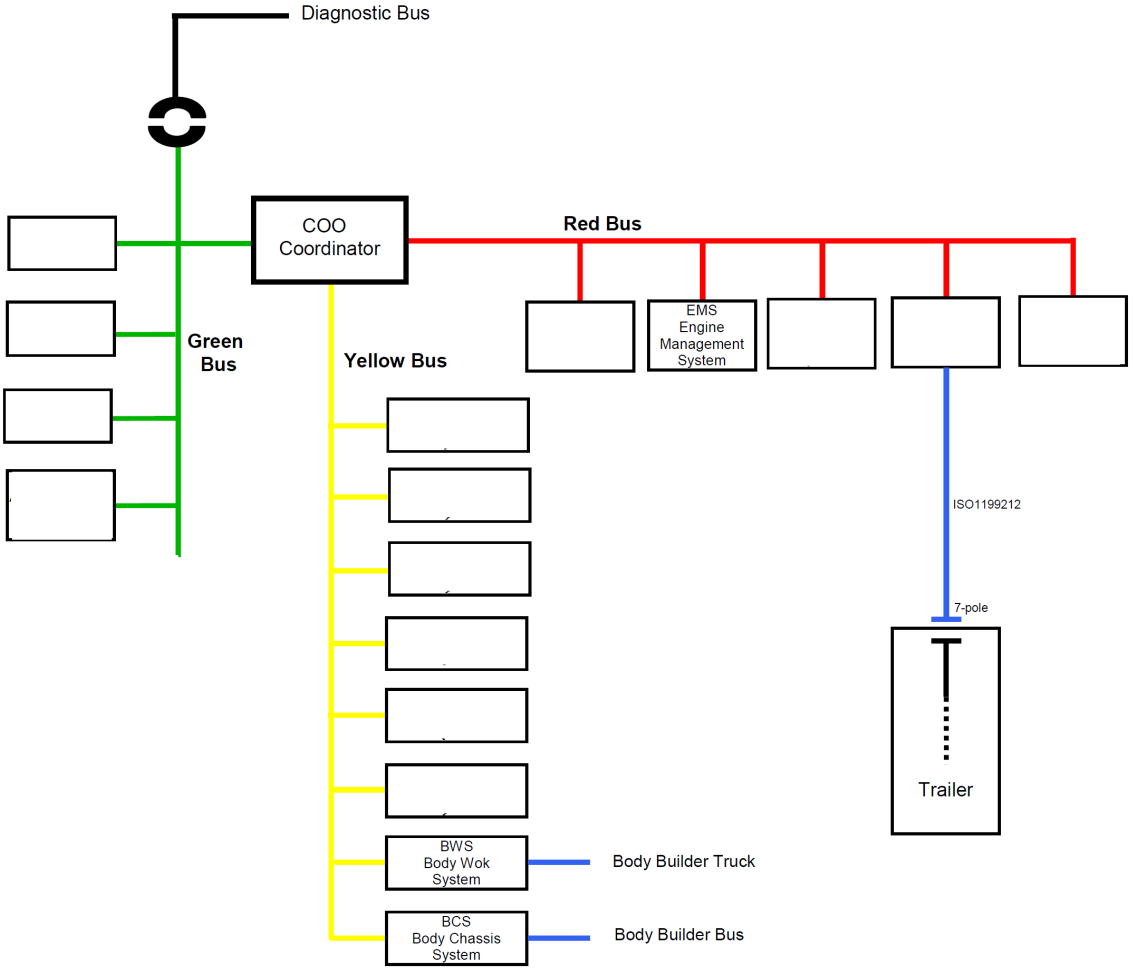

Figure 2.4: Scania's electrical system layout with CAN communication [17]

While CAN is the standard used to achieve communication between ECUs, a

truck's electrical system is comprised of more elements. A description of its composing elements would be:

- Hardware components:
  - ECU components
  - Connectors
  - Other components:
    - Sensor
    - Actuator
    - Switch
    - Power related
    - Other
- Cables

A collection of cables that connect or influence directly an ECU are physically grouped and called a harness. This can be seen in Figure 2.5.

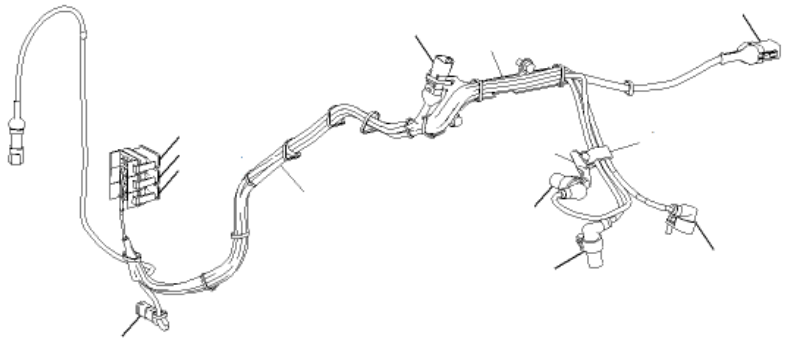

Figure 2.5: Physical representation of a cable harness. EMS rear cable harness

2.1.2.1 ECU Software

The Electronic Control Unit(ECU) is a common way of referring to an embedded system that controls one or more electrical systems or subsystems within a vehicle. The ECUs are specialised to a certain functionality of functionalities and named accordingly like: EMS (Engine Management System), BCM(Brake Control Module) and so on.

One ECU is not limited to one CPU or one control functionality, but can satisfy multiple functional requirements [18]. The core of the ECU is represented by one or multiple microcontrollers and clocks. Beside this, it is composed out of IOs, memory of different kinds, communication links, connectors, housing and other possible elements.

The general software structure of an ECU within Scania is composed out of the following layers:

- Low level software layer (also called ComP, i.e. common platform)
  This layer behaves as a RTOS and handles the communication with IO pins towards and from the external environment
- Middle layer (also called MIDD or LLAP depending on the ECU)
  This layer handles and distributes to global variables the information from the common platform.
- RTDB (Real Time Data Base)
  This layer contains global variables that are synchronized between ECUs and between application modules within th same ECU. Applications and ECUs can communicate and share values only through RTDB variables.
- High level software layer (Appl)
  This layer represents the implemented functionality.

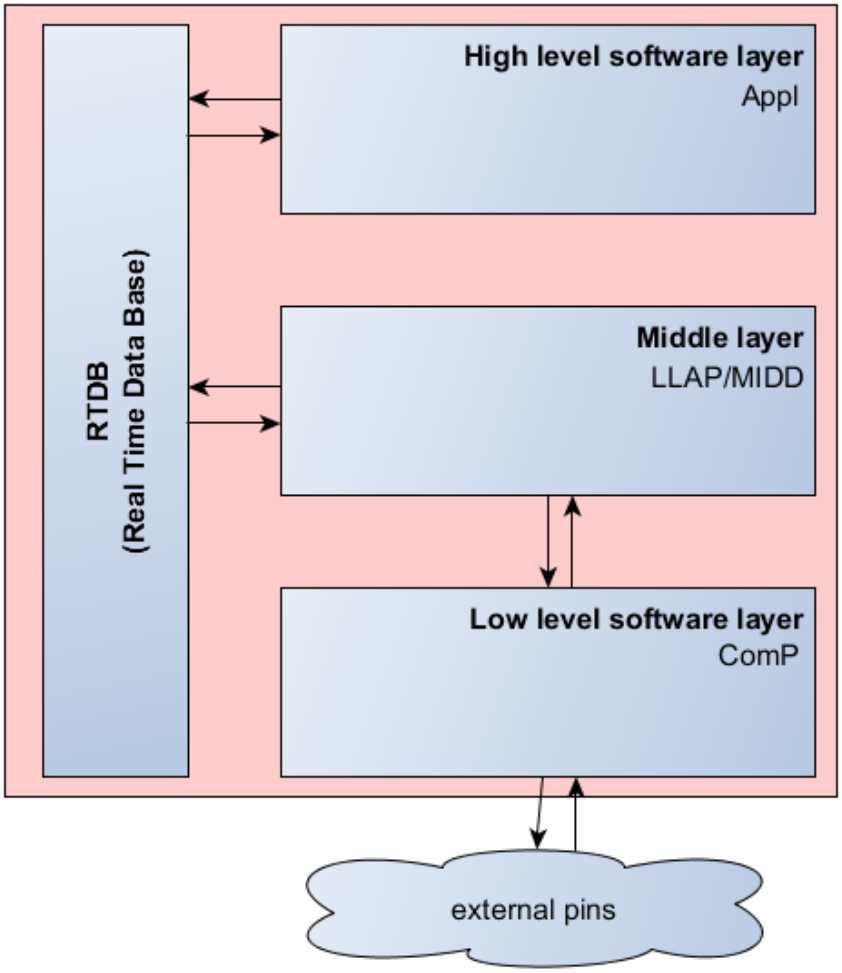

Figure 2.6: Common ECU structure.

The Scania technical documentation often uses the concept of the system of an ECU in different descriptions. An ECU's system is regarded as the physical

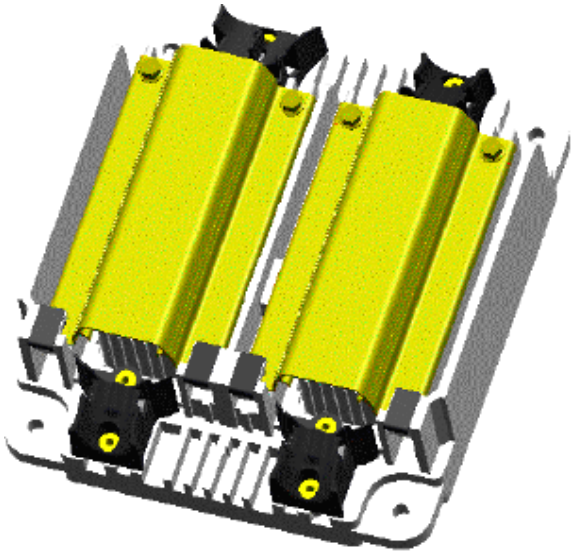

Figure 2.7: EMS S8 [19].

ECU together with the harnesses and hardware components (sensors, actuator, etc.) which it influences.

The next paragraphs describe briefly some of the most important ECUs.

EMS S8

The Engine management system (EMS) S8 is an ECU, directly placed besides the engine (See Figure 1.2), that controls the engine by means of the sensors and actuators that are mounted on the engine. The ECU is developed under collaboration of several units within R&D center at Scania including NES (Engine Control Software), NEP (System Platform) and REV (Vehicle Control Systems) and some other sections. EMS S8 communicates with the rest of the truck through CAN-buses, battery and ignition key. It is the control unit for the Scania Common Rail engines which is replaced its previous version, but still is backwards compatible. Figure 2.7 shows the front side of this ECU in a graphical view.

EMS S8 can be seen as both sensor and actuator from other ECU systems point of view in the vehicle. For example, as a sensor it can provide engine information such as engine speed, engine coolant temperature, fuel temperature, engine oil pressure, engine oil level, intake manifold temperature, alternator state and consistency check using the chassis number, to the control units that requested these information. Moreover, the functions in which the EMS will act as actuator are alternator PWM requests, demanded cooling fan, demanded control over the AC-compressor, demanded engine brake control, etc. [19, 20].

COO7

The Coordinator (COO) is an Scania ECU family which acts as a gateway. This means that coordinator ECUs collect information regarding actions made by the driver and distributes the information to other ECU's via the CAN network. As can be seen from Figure 2.8, the information is collected from switches and sensors, but also from other ECU's. REVE (Embedded Software) is the responsible unit for the development of COO7 software code and functionalities.

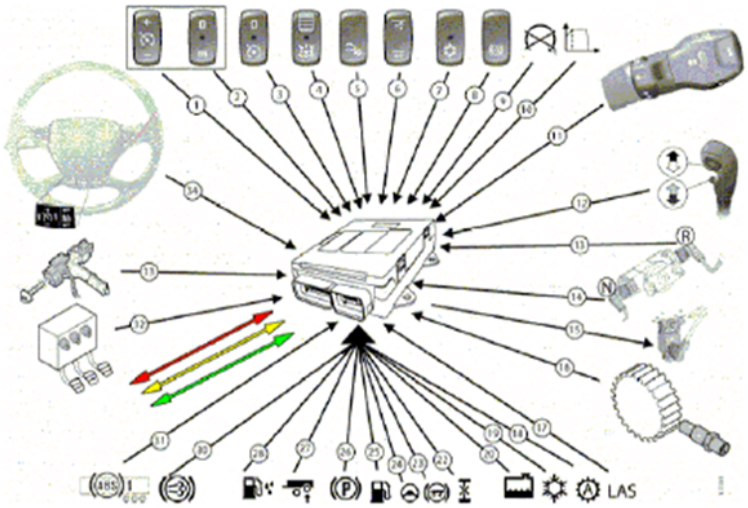

Figure 2.8: COO7 interactions with environment and its functionalities [21].

The COO hardware is the same for both truck and bus vehicles. However, the software may vary depending on the specified conditions and requirements. It can also be used in generic industrial and marine applications. In addition, COO can be exposed to a harsh electrical environment, large temperature variations, high humidity and vibrations. To work in this rough environment and meet the requirement for safe function and reliability it is be built according to the highest of automotive standards[21, 22]. The Coordinator is a central element in a truck's embedded distributed system as it can be seen in the following image, Figure 2.8.

COO8

COO8, Figure 2.9, is the new generation of COO, which will replace the current COO7 in trucks and buses. It is needed to ensure the growing demands in resources (processing power and memory) for future vehicles. It is likely to have an increased need of I/O as well.

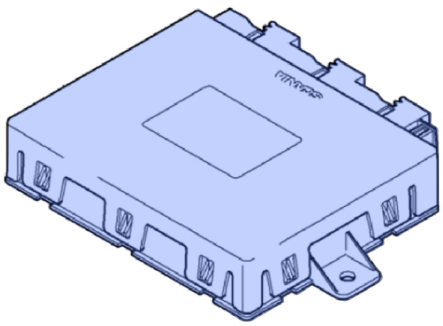

Figure 2.9: Front view of COO8 [23].

BCI1

BCI is the next generation bodybuilder interface and the newest member of the BWE family. The BCI is scheduled to replace the predecessors by offering a

common platform for trucks and for buses. It is needed in order to secure demands on memory resources, processing power and I/O functionality.[24].

CMS1

CMS1 is the first generation of the Chassis Management System. The CMS will provide more ECU recourses, to enable introduction of increased functionality, in the electrical system. The CMS will reduce the length of cable harnesses, as well as the amount of cables through the cab. It will also minimize the complexity of the electrical system, by utilizing a modular structure, with a standardized interface.[25].

Besides the ECUs other components that are of interest are the sensors and actuators.

2.1.3 Data sources

This section will briefly present some of the data sources which are found within Scania.

SOPS files

SOPS files are xml format files that contain the "Scania Onboard Specification". This files act as a blueprint for how a truck is configured. They contain consistent and structured information that can be grouped as follows:

- FPC block
  Contains codes with different variant names. FPC codes decide the variability of a truck, i.e. which functionalities are active and which hardware components are used. The collective information from the FPC block of the SOPS file controls what electrical components/hardware are used and in consequence switches on and off modules in the source code.

- XPC block
  This represents the actual value that the FPC takes, the actual value that is programmed into the ECU.

- ECU block
  Specifies the ECUs used. Presents the parameters which identify the ECUs, that can be changed in the workshop without FPC codes.

- CableList block
  Contains the names of xml files which describe the wireing of the electrical components in a particular truck configuration.

- UF block
  Contains a list of user functions that are implemented in the ECUs.

- Version block
  Contains information about when the SOPS files are created or modified.

Perforce

Perforce is a proprietary revision control systems used by Scania to persist and index both source code and documentation, including requirements.

The file structure for the ECU's source code is different from one ECU to another and also between different generation.

Previous information from the Neo4j graph db

Neo4j is a graph database used to store architectural information generated by the ESPRESSO project. The information can be queried using Neo4j's specific query language called Cypher or from the code using APIs.[26]

PT and SDP3

SDP3 PT is an engineering tool for specifying debugging methods and visually displayed in SDP3. PT is part of the SDP3 production environment. SDP3 stands for Scania Diagnos & Programmer 3 and communicates with Scania vehicles and Scania industrial and marine engines. The tool was created to connect to the the electrical system by CAN communication. The software is used to troubleshoot, calibrate and update ECU's software, etc.

Cable lists

Cable lists are files in xml format which describe connections between electrical hardware components in the style of electrical schematics. This files can be retrieved and updated using the PT tool.

One cable list can describe connections which would be associated to a harness. One cable list file can have a "Name", multiple "Connection" elements which are composed out of "Color", "CableLength" and two "ConnectioEnd". The "ConnectionEnd" elements contain as the most important elements: "PinId", "PinLabel" and the component with a component code as value, that defines the type of hardware. The component code is the formal name given to components as placeholders. The code comprises normally out of one letter and digits. The component code is a placeholder for the functionality that the specific component has to fulfil and not the actual physical part.

- Name
  The name of the harness and implicitly the cable.
- Connection
  - Color
    The color code for the cable.
  - CableLength
    The cable length.
  - ConnectionEnd
    - Hardware component: ECUComponent, Connector, Component
      The hardware components are groupted in three main branches in the cable list. ECUs and ECU related systems and elements are ECUComponent, the connectors are also separately specified and any other component that doesn't fit in any of this, is simply "Component". The value of this is represented by the component code.
    - PinId
      The name of the pin.
    - Label
      The label of the pin from the electrical schematics.

  - ConnectionEnd...

- Connection...

ECU family from the PT

This represents a collection of files within a predefined folder structure, retrieved via the PT tool. The files are stored in xml format and provide a description that connects a specific configuration of the hardware and software of an ECU with its component code. Every new version of an ECU, regarding software and hardware, is described here in the previously mentioned way.

Component library

The component library file is an Excel document which contains all component codes with human generated description. Only one user is the owner of the document and has the right to edit it. In this way there can not be conflicts. There is a different sheet for each type of Scania product, e.g. truck, bus, etc. Some of the available fields are:

- a reference letter
- a series of digits and possibly letters(for variants)
- type
- description
- harness
- system

The reference letter specifies the broad functionality that the component fulfills. Based on it and the type, the following groups are available:

A	Ignition coil/Spark plug
B	Switch
C	Connector/ Splice/ Joint connector
D	Diode/ Sensor/ Module/ Resistor/ Transistor
E	Control unit/ Sensor/ Limiter/ Radio/ Amplifier/ Converter/ Screen/ DSRC/ IO Expander/ Video/ Display
F	Fuse/ Fuse holder
G	Ground
H	Lighter/ Heater/ Squib unit/ Auxiliary equip
K	Diagnostics
L	Lamp
M	Motor/ Electric machine
N	Signal horn/ Loud speaker
O	Instrument/ Tachograph/ Counter/ Timer
P	Battery/ Central el. unit/ Alternator/ Power distributor
R	Relay/ Holder
S	Switch
T	Monitor/ Sensor
U	Aerial/ Refridgerator/ Spring coil/ Condenser/ Microphone/ Additional AC/ Radio input/ Splitter/ USB/ ID reader
V	Solenoid valve/ Pump/ Control module/ Dosing unit
W	Control lamp/ Warning lamp

Table 2.1: Component code meaning from component library

SESAMM tool

SESSAM is the name of Scania's common electrical system software tool and it stands for Scania Electrical System Architecture Made for Modularization and Maintenance. It was developed within the RESA department with the purpose of adding quality and automation on a higher level to the workflow of system architects.

Spectra

Spectra is a Scania framework and application used to manage product structure which is time and condition depending. The data stored has a hierarchical layout and contains both logical and hardware elements. It also contains the structure for the variability in Scania's products. A downside of the tool and database is that the data is most often inserted manually which can lead to incompletness and inconsistencies.

OAS

OAS is a next generation tools for object- and structure-administration within Scania. The main motivation for this tool is defined through methods and procedures for maintaining, as well as replacement of current IT-support for administrating, the product description. The database it uses is being migrated from other sources and merged.

Multi

Multi is an aftermarket software tool and database for maintenance guidance and replacement parts. The inconvenience with this tool is also that it is manually populated, but it is consistent although not complete.

2.2 ISO 26262 in SCANIA

One of the main concerns that has lead to the foundation of the ESPRESSO project within Scania is functional safety of E/E systems or more specifically, the conformance of vehicles design and properties to ISO 26262 standard [2]. In other words, the current development process of automotive products in Scania does not follow the required approach to functional safety, which is emphasised by the terms and conditions of ISO 26262. On the other hand, Scania trusts its current workflow and claims full safety of its products. Therefore, several research projects including this thesis work have been defined within ESPRESSO to investigate possibilities to map Scania safety design and evaluation to the corresponding requirements in ISO 26262 standard.

2.2.1 Functional safety

ISO 26262 is a branch of the international functional safety standard, IEC61508 [27], which focuses on safety issues of critical E/E systems. ISO 26262 addresses the safety of embedded systems in automotive industry and it covers all development phases from specification and design to production and maintenance. Risk evaluation is the main concern of the standard. In this a systematic approach has been defined to estimate failure probabilities in the hardware and software that handle risky functionalities [2]. ISO 26262, which is structured according to the V-model [28], comprises ten development phases for vehicle production. The 'V' signs in Figure 2.10, represent overview interactions among different manufacture phases and it requires thorough understanding of the vehicle on a system level. In fact, many automotive manufacturers endeavour to possess qualifications in compliance with ISO 26262 not only to enhance their products safety evaluations, but also to satisfy their customers concerns and remain in the vehicle's competitive market.

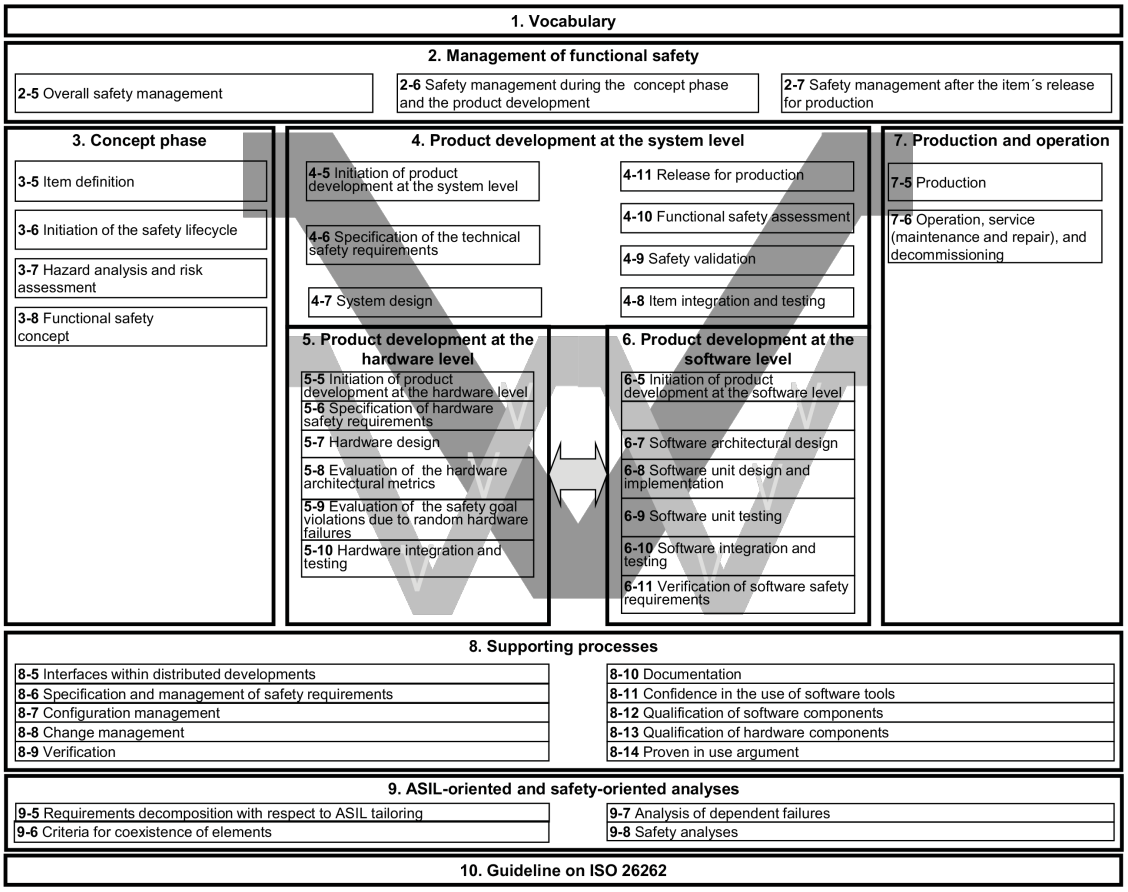

Figure 2.10: Main view of ISO 26262 [2]

The blueprint of ISO 26262 safety standard is built upon the architectural analysis of embedded systems. Here, the aim is to assign safety measure levels (ASILs) [2] to components in the architectural models of E/E systems. Therefore, safety constraints and requirements can be tracked down to implementation layers. As a conclusion, a vehicle's embedded system should evolve in a top-down approach from models and architectures down to implementation in hardware and software level. This is for the purpose of examining whether requirements of functional safety are satisfied. The problem arises when a vehicle is not produced according to te necessary concepts of the ISO 26262 standard. Because of its great demand on time and cost, companies do not agree with instant changes in their workflow toward model-based approach. One way to resolve this challenge is to reconstruct the architecture of E/E systems from existing implementations in a bottom-up approach [3].

2.2.2 Hardware-software interface specification

Apart from system level development, two main chapters of ISO 26262 focus on hardware and software production levels (5 and 6). These chapters are not only focusing on their separate concepts, but also emphasising the interfaces between hardware and software elements. More specifically, hardware-software interface need to be investigated in both HW/SW views to identify potential risks and hazards. Thus, the

specification of interactions between hardware and software components should be defined according to safety concerns in the ISO 26262 standard. HSI identifies hardware components that are handled by software applications or hardware components that provide information for the software. Actuators and sensors are examples of such devices. Therefore, it is obvious that hardware architecture and the interfaces between software and hardware components should be well documented or it should be extracted from the implementation, to be able to perform safety evaluations on the embedded system of a vehicle.

2.3 Literature study

This thesis involves research and technical knowledge from multiple areas, thus a standard literature review cannot be performed in the original sense. It is firstly very hard to find research papers that fit the subject of this thesis and secondly, the problem at hand is highly connected with the context of Scania's industrial environment.

Because of the previously stated, the subsections of these chapters handle each a part of the research areas. For each of this research areas, an analysis is done on the latest published articles and the different properties of the different techniques are argued. The information from the discussed papers will be used as a base for analysis and later implementation.

As a first step, an approach is made towards the theory behind architecture recovery and it is argued what benefits different recovery techniques bring. Following this, the focus is shifted towards architecture description languages since this work attempts to describe an overall architecture focused on automotive. The next topic refers to model composition and multi-view modeling for being able to assess the models that the different departments have and fuse them together. In the same time, the particularities from each department need to be preserved in views that are as similar as possible to what the end users know.

The next section refers to the graph database, which has been found the most suitable for storing architectural information. The arguments behind this are not the focus of this thesis.

The last section handles variability, which although is not within the scope of this, thesis still needs to be accounted for and regarded as a future work.

The analysed papers were discovered partially by searching for specific key words in KTH's on-line library and other repositories on the Internet. Articles that had a peer review and/or appeared in publications were considered reliable sources. A great part of documents based on which the thesis is relying are Scania internal technical documents and cannot be disclosed.

2.3.1 Architecture recovery

In software engineering, architecture is defined as the structure of model elements from which the system is built out of. It shows interactions and also composition of components in a higher levels of abstraction [29]. In other words, a software

system's architecture reflects the blueprint of the structure view and the major design decisions which have been made during the evolution of the system [30].

During the development of software, legacy code evolves, which in turn entails changes in the system architecture. Documenting this evolution requires high effort and emphasis on tracking architectural changes which is not always the case in practice. Therefore; the designed architecture becomes inaccurate and architectural degradations are inevitable [3, 31]. Architectural drift, erosion, mismatch and chasm are side effects of introducing design decisions into descriptive architecture that are not reflected in the designed architecture. The resulted inconsistencies make system understanding, testing and verification difficult and challenging [3, 30, 31].

One approach to solve this problem is to apply architecture recovery (also called in the literature architecture reconstruction, mining and extraction [31]) methods to extract and describe the implemented architecture, instead of spending so much time and effort on synchronizing perspective and descriptive architectures [3].

Architecture reconstruction is the process of recovering architectural information and building a representation of the implemented system from existing artifacts. This is a bottom-up approach which usually involves reverse engineering and is necessary because of the three main reasons [3, 29, 32]:

1. There is no documented architecture of the system.
2. The documented architecture is not valid and is not updated properly.
3. Inconsistency between the designed architecture and the one that has has been implemented.

2.3.1.1 Architecture recovery in automotive industry

Embedded systems in automotive industry, are becoming larger and more complex. They consist of several ECUs with a vast number of code lines for integrating different sophisticated functionalities in the vehicles. Nevertheless, safety critical requirements always should be met during different phases from development to verification. To detect safety issues in the implemented software, one could simply examine the architecture. More specifically, many safety related problems that are originated from design issues can be uncovered from architectural perspectives [33]. Hence, having clear and compatible descriptive architecture to the design, is crucial both for system understanding and early error detection in the software [3].

In addition, many automotive industries use a modular design for their products. This means that they reuse software and hardware from one or several ECUs in different configurations independently. Therefore, the final architecture of the system will be dependent on the use case and will not be resolved before the compilation of the source code. In regards to this, architecture recovery can be employed to retrieve the architecture of the final configuration or of the individual modules. In other words, developers can use this architectural information to foresee integration possibilities and potential challenges [29].

2.3.1.2 Architecture recovery phases

Architecture recovery (also called reconstruction) procedures are in general composed of four main steps (See Figure 2.11) that are listed below [29]:

1. Raw data extraction. In this phase, preliminary architectural information is collected from different artifacts such as source code, design documents and build scripts.

2. Database construction. It is necessary to structure the extracted information and store them in a well established layout. Therefore, in this phase the meta-model of the database should be designed and all collected raw data should be stored in the database in compliance to the model.

3. View fusion. Each set of extracted information contains a partial view of the architecture of the system. Hence, in order to build complete and precise picture of the architecture, possible relationships or dependencies between stored data need to be recognized. Fusion phase combines and relates elements in the database to create a more abstract layer or to add more information to one or several elements.

4. Architecture analysis. During previous processes, some assumptions and theories are shaped and evolved which result in the formation of the system's architecture. These hypotheses ought to be tested and verified if they are correctly made. Thus, the aim of the analysis phase is to examine the constructed architecture to detect any errors and/or to suggest if further refinements or supplementary information is needed.

All four aforementioned activities are performed iteratively in the architecture recovery process. Accordingly, the resulted architecture will be refined steadily and will have the highest accuracy and conformance to the implementation [29].

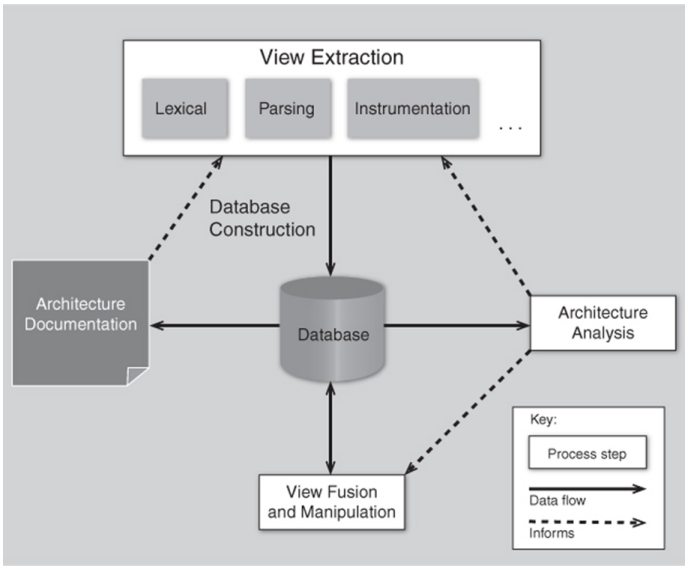

Figure 2.11: Architecture recovery procedures [29]

2.3.1.3 SAR techniques

In literature, various techniques have been proposed for reconstructing architecture of the distributed software including data flow based, pattern based, model based

and domain based approaches [32, 34]. Classification of these approaches, on the other hand, requires considering many factors such as intended goals, necessary inputs and expected output. Therefore, one can group SAR approaches, for example, based on the techniques they use to handle the data [31].

Figure 2.12 shows the organization of overall architecture recovery approaches and their classification based on the different considerations [31]. As can be seen from the figure, SAR techniques are divided into three main categories whether they are manual or automatic. In fact, the defined boundary in the taxonomy is not explicit, meaning that a combination of those techniques usually is used to reach the desired recovered architecture. In a quasi-manual technique, the user itself identifies what elements are architectural entities and uses a tool to construct the views. Here the tool reduces the engineering effort by providing visualization facilities to map and build abstraction layers. However, in semi-automatic approach, it is the tool that assists the reverse engineer by detecting and discovering abstraction elements that are specified by user. The iterative aspects of recovery processes are also handled by software tools while the user can guide it to track the right architectural elements. In other words, the difference between manual and automatic architecture recovery lies under how the user and the tool interact with each other.

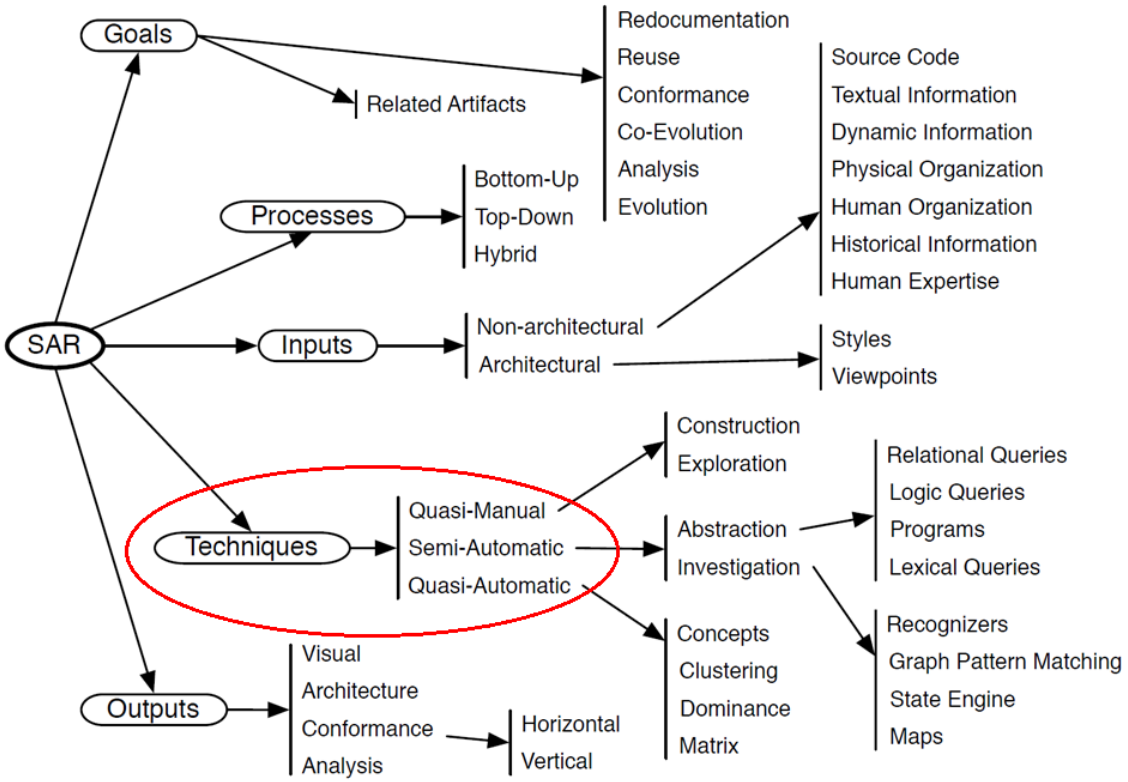

Figure 2.12: Different types of software architecture recovery [31].

2.3.1.4 SAR techniques for ECU software

Source code together with runtime and execution traces are among the most trustworthy and dependable references for extracting architectural information from the

implemented software. While statistical information like call dependencies, cross-references and hierarchies can be obtained from the source code, the behavioural or dynamic views are retrievable from execution traces of the system. Static analysis aims at outlining the structure of the software system containing components and their connections whereas dynamic analysis, in object-oriented programs, tries to draw the picture of system behaviour and elements interactions [31, 32, 34–36]. Because of complexity and large scale of embedded systems in automotive industry and more specifically heavy vehicles, it is problematic and hard work to execute the whole system in order to gather runtime data. In addition, the source code of ECU software which is written in C programing language contains major part of its architectural information. From the source code, structure of software components and their hierarchy as well as behavioral modes and their dependencies are retrievable [37]. Therefore, it is more preferable to apply static recovery methods when C code is in use. When static architecture is the aim of recovery operations, pattern matching and semantic analysis are prefered among other recovery techniques. Especially for C code that is based on naming convention and pointer manipulation [11].

2.3.2 Architecture description languages

One common issue during the design and development of software applications is to give the same view and understanding of the requirements to engineers as initially defined by stakeholders. It is also very important that everyone has the same understanding about the structure of a system and the approach of software implementation[38]. Initially it is the responsibility of the software architecture to resolve these concerns. However, architectural elements should also be standardized. Hence, to distinctly define and classify architectural models, one common language is needed.

Architecture description languages (ADLs) have introduced concepts, components and models to formally describe systems architecture in an understandable way for all user groups. They also facilitate architectural improvements and standardise design phases [31].

2.3.2.1 Automotive ADLs

Embedded systems are composed of both hardware and software. The software part that runs on embedded processors, has become increasingly large and complex [3, 39]. Especially, in automotive industry, which consists of several distributed embedded systems, the software has to take care of various functionalities from fulfilling hard and safety real time requirements to handling resource limitations and communication challenges over shared buses. In addition, developing automotive embedded systems requires considering different factors such as complexity and cost efficiency as well as reducing time-to-market and reusing previously developed hardware and software [39]. Because of these factors, developing automotive embedded systems that usually evolve on legacy systems, requires full understanding of the system in higher abstraction levels [39]. This development seems not possible without the true help of an architecture description language that is specialized for this domain and

covers not only vehicle features and various functionalities but provides modelling and analysis of software and hardware elements [40].

There are several advantages in using ADLs when designing embedded systems. Most obviously, it is possible to define a high abstraction models of components and refine them over development and implementation. Another advantage is that ADL models can be read by computer programs. Thus, making it possible to write analytic and verification programs and detect errors at early stages of design. Finally by representing the interfaces between elements and components it is easier to implement the system according the design [40, 41].

There are many ADLs that are designed academically or commercially to address architectural and modeling challenges in embedded systems. Architecture Analysis & Design Language (AADL), Systems Modeling Language(SysML), SystemC, Modeling and Analysis of Real Time and Embedded systems (MARTE), are examples of those languages [42]. Since automotive embedded systems have intricate real time and distributed properties, they demand modeling structures and ADLs specialized for vehicle systems. EAST-ADL and MODELICA are among those ADLs that support automotive embedded systems modeling and development [40–42].

2.3.2.2 Modelica

Modelica is a modeling language for complex systems that composed of different heterogeneous processes. As can be seen in Figure 2.13, an advanced vehicle with mechanical, electrical and control software components can be modeled by this object-oriented language. In Modelica, the behavior of subsystems are represented by mathematical equations inside classes. These classes are then translated to objects that can be analyzed by the simulation engine [42, 43]. Even though Modelica is a powerful language in modeling complex physical systems, it does not provide architectural models to construct abstraction views of software related entities. Moreover, although its modeling features help engineers to simulate and analyse the behaviour of the system in the real environment, it does not cover interfaces from physical components to software application that control them in order to investigate safety goals.

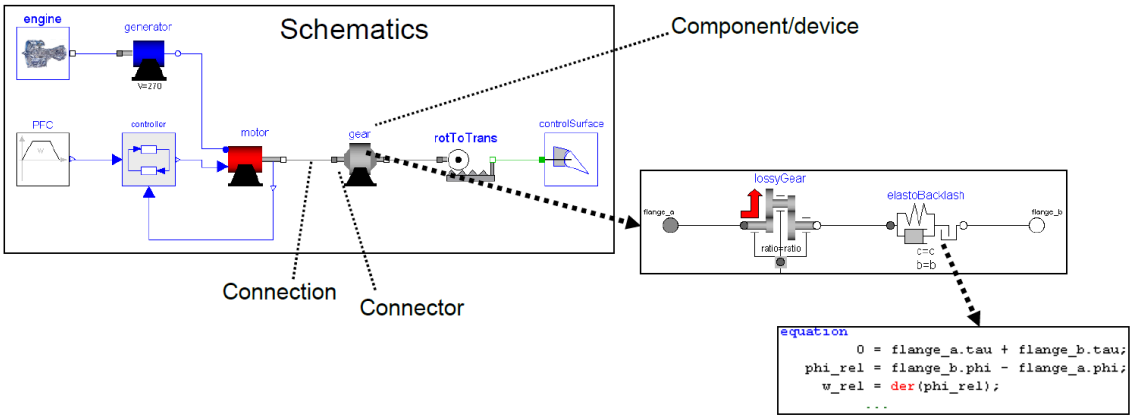

Figure 2.13: Modelica language application [43]

2.3.2.3 EAST-ADL2

EAST-ADL2 is a comprehensive architecture description language used in automotive industry for describing embedded systems. While Autosar standard only concerns about implementation level, EAST-ADL2 has supplementary views at higher abstraction levels that improve system development processes. Architectural perspectives that are governed by EAST-ADL2 cover vehicle features, functions, requirements, variability, software components, hardware components and communication [41, 42, 44–46].

Architectural structures in EAST-ADL2 are abstracted in four main levels, each of which concentrated on a separated aspect of embedded systems development. As can be seen in Figure 2.15, overall behaviour of the vehicle is defined as different features in Vehicle Level whereas other layers tend to realize these features with functions, hardware and software entities and components. Design Level is the place for hardware architecture design where decisions need to be made for proper selection of ECUs, sensors and actuators and their interfaces, and communication busses. However, model analysis of sensors and actuators and also other hardware related considerations before design might be done at Analysis Level. EAST-ADL does not have a separate approach for its implementation level of the embedded software. Instead, it uses Autosar concepts and elements to convert high level and functional descriptions into applicable software components at a low level model [41, 42, 44–46].

Autosar is a well-known software architecture that is dedicated to automotive industry and very often used together with EAST-ADL. It is mainly intended to standardise architectural elements for software development in E/E systems. Autosar covers all vehicular software aspects including basic functionalities, interfaces and communications, modularity and integration, maintaining, updating and upgrading supports during the life-time of a vehicle [42, 47]. Component base design is the key feature of Autosar standard. This feature enables building desired applications by composing software components and connecting them together with proper interfaces . Figure 2.14 and 2.15 shows the architecture view of Autosar. It consists of three main layers and their objective is to provide ECU and hardware independent software application development. The Basic software layer (BSW) establishes a platform of fundamental functionalities for accessing hardware resources and real time control and management on ECUs. These services can be used by upper layer, Runtime Environment (RTE), to handle tasks and information flow between hardware components and software applications. Application layer which is located on top of RTE, contains stand-alone software components that are responsible for vehicle behaviour in different circumstances. All data and control interactions and synchronizations between those components and also lower BSW layer are handled by RTE. Therefore, applications can be designed independently and regardless of what ECU hardware is used on the vehicle [42, 47, 48].

Although Autosar language is made for designing and modeling E/E systems architecture in automobiles, it lacks fundamental concepts to cover hardware architecture intensively which is equally important as software architecture.

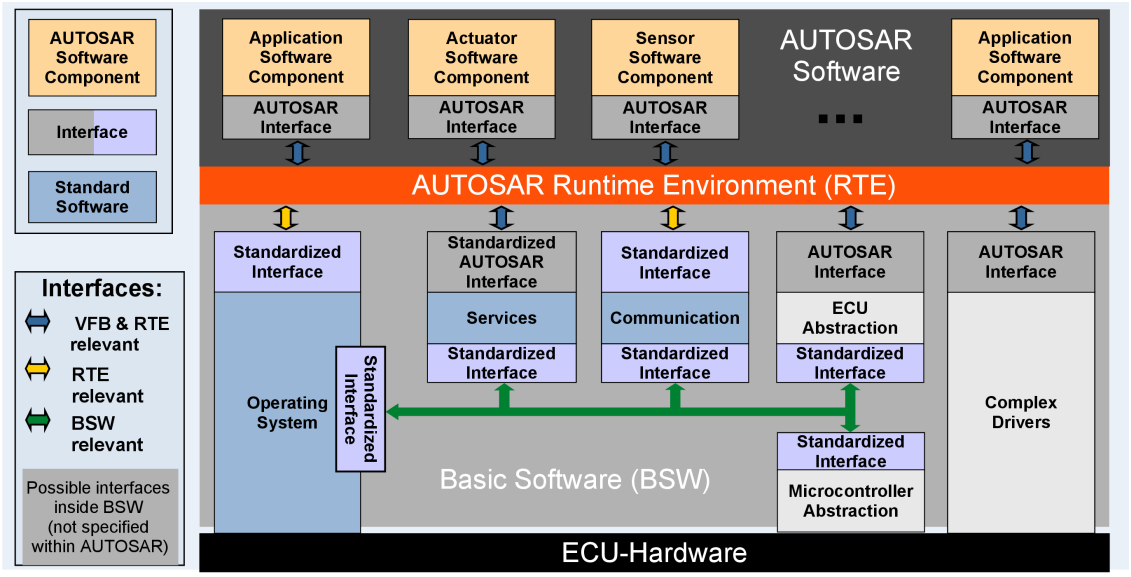

Figure 2.14: Autosar's layered architecture [47]

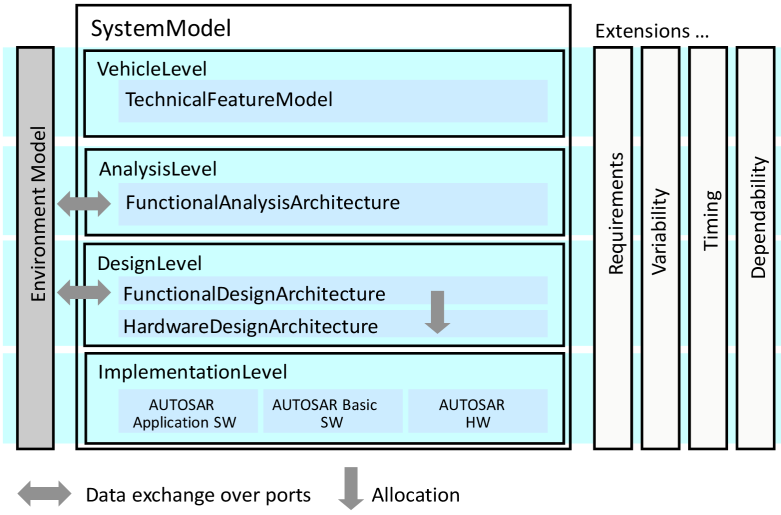

Figure 2.15: EAST-ADL System model and concepts [44].

One of the major concerns in automotive industry is the safety evaluation of vehicles in different levels and situations. It should be cleared out from primary design stages, how dependencies between system elements are organized in architectural models. Thus, it would be possible to recognize safety critical components and analyse corresponding failure consequences. Functional safety is the assessment of system behaviour against its intended functionality. It contains aspects of system reliability, dependability and controllability [41, 42, 44–46]. As a model-driven approach for automobile embedded system design, EAST-ADL provides fundamental methods to perform functional safety process from early architectural design up to implementation phases. The safety support is in conformance to ISO 26262 which is a well-known functional safety standard established for electronics and electrical systems in road vehicles [2, 41, 42, 44–46]. Risks and critical aspects are defined

as modeling entities in the language, in order to demonstrate safety views. Those perspectives that include components ASILs (Automotive Safety Integrity Levels), hazardous events, and safety goals and requirements, are needed to generate fault tree for further analysis [41, 42, 44–46].

Functional safety concerns are reflected in all abstraction levels of EAST-ADL. Figure 2.16 shows parts of ISO 26262 standard that are addressed in each level. While in Vehicle and Analysis levels, safety considerations are turned into relevant requirements and architectural structures, safety standards are integrated into lower levels by providing safety case study and error propagation analyses [41, 42, 44–46].

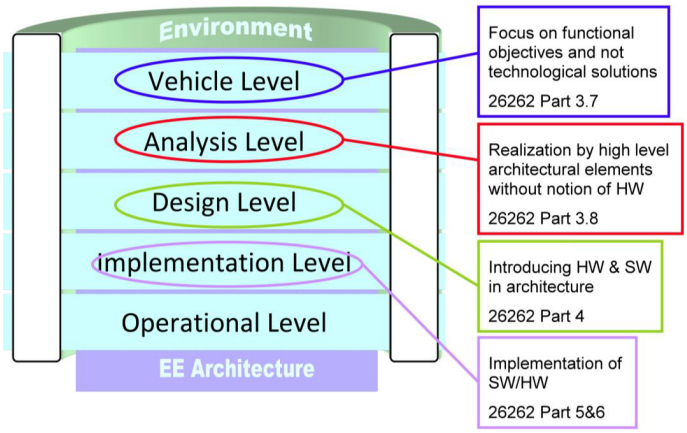

Figure 2.16: Mapping of ISO 26262 standard to EAST-ADL concept levels [41].

2.3.2.4 HW architecture in EAST-ADL

Automotive embedded systems are built of both hardware electrical parts such as ECUs, sensors/actuators, and software application components. Therefore, E/E system's architecture of a vehicle is not comprehensive, unless its hardware architecture has been considered in the design models. Likewise, EAST-ADL which is known as a thorough modeling language in automotive industry, supports the hardware structural views in its domain [46, 49]. More specifically, the meta-model of EAST-ADL provides the necessary infrastructures to design and build hardware architecture conveniently and in conformance to safety standards like ISO 26262 [2]. Figure 2.17 shows the diagram for hardware modeling that is proposed by EAST-ADL. It contains semantic models that are essential for representing hardware view of embedded systems architecture [46]. As such, a Node represents computational components like ECUs, Sensor represents analogue or digital sensors and Actuator represents all types of actuators like valves or motors that are used in a vehicle. A detailed and pure hardware view, similar to what can be found in schematic drawings, also includes pins, wires and connector elements. These components are imported in the model as HardwarePin, HardwareConnector, BusConnector or HardwarePortConnector [46], 2.17.

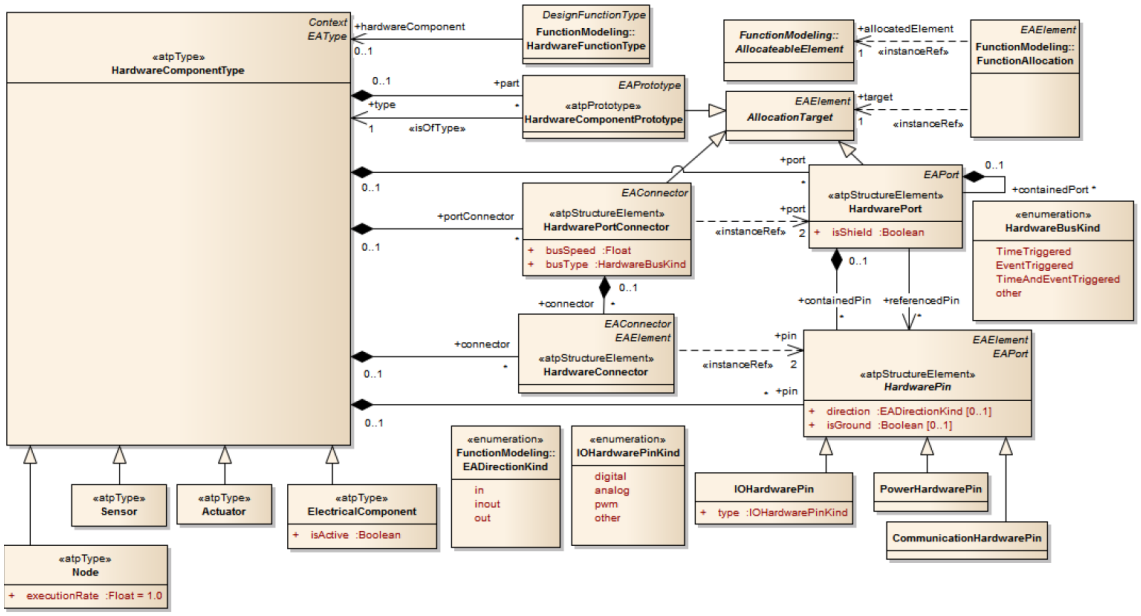

Figure 2.17: EAST-ADL Hardware modele diagram [46]

2.3.3 Model composition

Model composition, in literature, is defined as a group of processes that are necessary to combine two or more models in order to construct an integrated view of the structure [50, 51]. As can be seen from Figure 2.18, the resulted model can have same attributes as its original ancestors or the composition can yield to new combined features in the element [51].

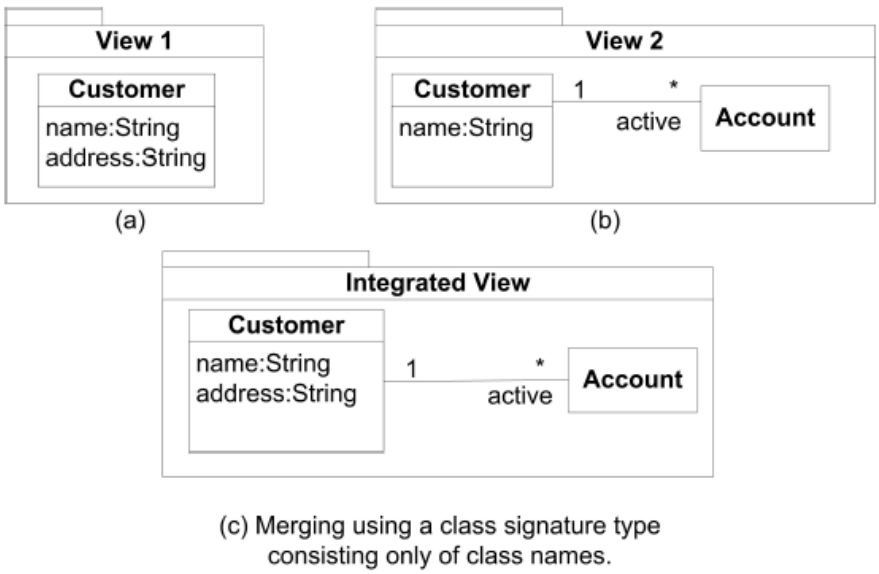

Figure 2.18: Model Composition

Model composition is applicable in several concepts such as meta-model reusability, merging aspect model elements and integration of models that are described in

different languages [50, 52, 53]. It composed of two main phases named as matching and merging steps. As its name suggests, matching phase contains activities to recognize distinct model entities that originated from the same perspective whereas merging phase includes operations to combine those matched entities in order to generate new architectural models [53, 54].

Matching view elements requires precise understanding and information about the meaning of the architecture description language that models are described in. Therefore, identification process is domain specialized, to correctly detect corresponding similarities in properties of model elements [53]. Named-base technique is the simplest approach for composition matching. Here the operator only looks for identical names among components in the model. However, this approach results in conflicts wherever naming information is not sufficient to decide on concept matching. The solution to the previous method is signature-based technique and it means to track attributes of elements that represent different views of the same concept in a language's meta-model to recognize matchable components [55, 56].

On the other hand, composer operator which is a domain independent process, can be composed of primitives such as "union", "override" and "extend". Combination of these primitives according to composition rules (e.g. merge rules, correspondence rules and override rules) creates merging scenarios suitable for target ADL. As such, Equivalence and Inheritance operators are used in integrating meta-models in UML based domain [51–53, 57].

Current advanced vehicles are composed out of large and extensive embedded systems. Various ECUs, that are responsible for one or several functionalities and interact with different hardware components, are designed and developed in separated departments. However, even though each department has its own view model of sub-parts that is working on, their concerns are related to the same vehicle's embedded system's architecture. Therefore, identical concepts that are described in distinct modeling forms can be merged into a general model to depict the overall picture of the system architecture. In addition, while it is more convenient to study and analyze smaller sub-model entities, composing the models will result in higher abstraction levels and provides the possibility to evaluate the system design more globally [58].

2.3.3.1 Multi-view modeling

In recent years, meta-model composition has been subjected to several research studies [59]. Various methods have been investigated and proposed to cope with challenges and solve mismatches during composition operations. Meta-model Merge, Interfacing, Class Refinement and Template Instantiation are examples of those techniques [59].

The technique of multi-view modeling can be seen as a way of model composition. This technique is described as having two main approaches[9]. On one hand a system can be broken down into distinct meta-models depending on what each view is aiming to represent. The overall system is a composition of the information carried by the different models. The second approach assumes a single meta-model and virtual view in which each view made by filtering out unnecessary information. Some of the future advantages of multi-view modeling are identified and investigated in

[60]. Multi-view modeling is a technique that can reduce the complexity in software systems. In [61] a practical use of this technique is presented in a hybrid approach when the number of views is not set, but all the views are described in the same modeling language. Multi-view modeling has been proven to be of interest as well in the case of ADLs as described by the paper [62].

2.3.4 Graph databases

While a relational database like SQL is capable of storing data in form of tables, a graph database handles information on the basis of graph structures. In graph theory, a graph is composed of nodes, edges (relationships) and properties (see Figure 2.19). Hence, graph database uses nodes and properties to represent data entities and their attributes, and edges to indicate the relationship between data nodes [63].

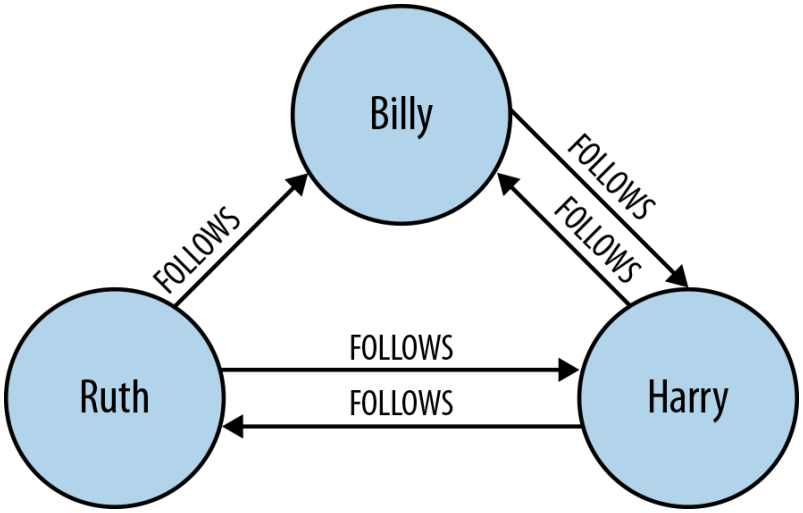

Figure 2.19: graph theory picture [63]

As graphs are extremely convenient to be utilized for data-set and system understanding, its database counterpart can easily represent the structure of information that it holds. The management engine in a graph database uses the concept of $index - free\ adjacency$ which means that connected nodes are directly pointed to each other. This results in improving $traversal\ performance$ at the expense of memory deficiencies[63, 64].

Similar to relational databases, the processing engine can handle all kinds of query operations including create, read, update and delete procedures. Nevertheless, its dependency on the graph theory and algorithms, enhances the efficiency on processing large quantity of data to find specific relationships or properties[26, 63].

2.3.5 Variability

Variability is an ambiguous term with vast meaning coverage which mostly is dependant on the concept that it is referred to. In general, it can be defined as "$The quality, state, or degree of being variable or changeable$" [65]. However, variability can also refer to varieties of certain elements in a system [66]. More specifically

in automotive industry it is the key to modularization and customization of products. With modularization (see Figure 2.20), different parts of a system can be developed independently and thus, new generation entities combine to build various productions. Also, customizable products lead to customers satisfaction since they can select whatever fulfills their concerns and requirements [13].

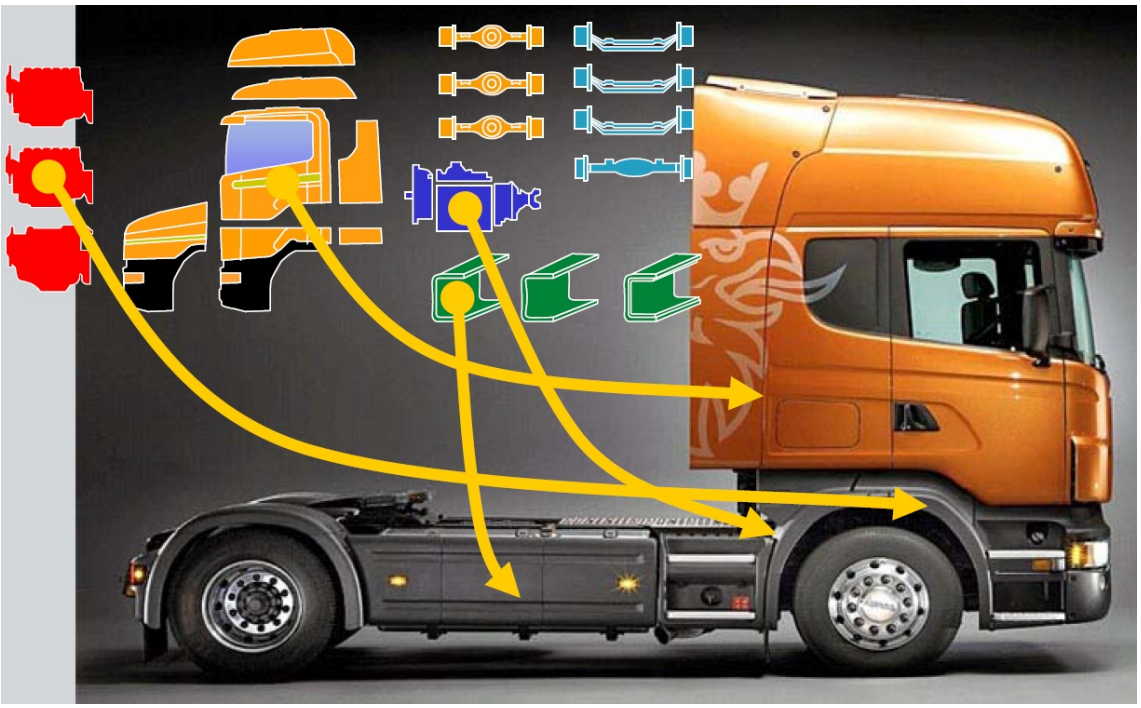

Figure 2.20: Variability and Modularity in Automotive Industry [67]

In a modular system, such as vehicle, variability encompasses all entities, components and parts that are changeable, adjustable or optional. Hence, variability can occur in user functions, architectural models, software and hardware implementations, components and parts. Even a tunable filter that is implemented in software, is counted as a variant element. One important aspect of variability is the variation point, which is the location that variability happens in an entity [66]. The variation points distribute over the whole development phases from system specification and requirements down to production, test and maintenance processes. They evolve during the life cycle of vehicle to provide optional functionalities as well as improving quality and safety [68].

2.3.5.1 Variability in EAST-ADL

Embedded systems in automotive industry consist of ECUs, software applications and hardware components like sensors and actuators that let the system interact with the environment. In order to provide the desired functionalities, developers of such vehicular embedded systems need to resolve many potential contrasting requirements and constraints. Thereby, huge number of variants will be introduced to the system while increasing the complexity dramatically [68, 69]. Therefore, a systematic approach is required to manage the extensive variability from early

design stages. Based on this and other similar motivations, automotive ADLs, such as EAST-ADL, have included variability concept in different abstraction layers of their language [68, 69].

In fact, EAST-ADL provides two levels of variability modeling, one in Vehicle level and the other one in Artifact level. In Vehicle level, EAST-ADL manages variability in a sense to show the existence of variants in the system on a very high abstraction level with no implementation consideration. However, how the existing variability affects the corresponding design is not identified yet. Here, feature models are formed according to common and variable attributes and will be handled on analysis and design levels. On the other hand, artifact level is the location where variation points are defined within feature model elements including Functional Analysis Architecture (FFA), Functional Design Architecture (FDA) and requirements. Now, variability can be linked to functions in the design and analysis layers. As shown in Figure 2.21, feature layer variability configures artifact layer variability and propagates variation points over other implementation layers [41, 49].

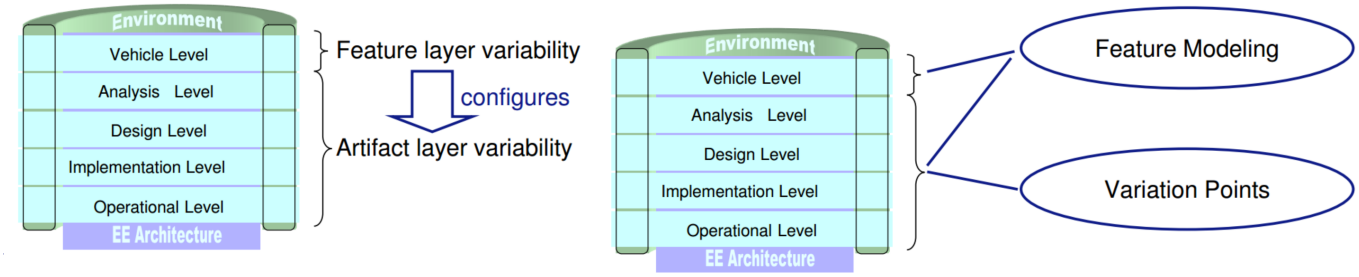

Figure 2.21: Variability concept in EAST-ADL [41]

3. Modeling and Recovery

This chapter describes the connection between theory and implementation in this thesis. It gives an overview of required information, studied data sources, methodology and work-flow. In addition, the implemented meta-model, based on the researched material, is explained afterwards in this chapter. Chapters 3.1 through 3.3 contain within explanation about the design of the solution. Finally, it presents the tool chain implementation and the challenges faced towards achieving the results.

3.1 Architecture recovery, requirements and constraints

Based on the outcome of the background study and literature review in Chapter 2, the necessary information and data sources are identified. These are used for applying relevant architecture recovery approaches and for extracting the right abstraction elements both from software and hardware entities. Further, the investigated reconstruction techniques, which prove to be compatible with Scania's situation, are optimised and implemented to store architectural models in a graph database. Here, the acquired knowledge about the graph databases and model composition is useful not only to efficiently query large amounts of data nodes from database. It is also employed to compose that information with the retrieved architectural elements and update the database.

Moreover, the next step, which is extending the ESPRESSO meta-model to cover hardware views, is done based on the research on automotive ADLs, specifically EAST-ADL. Thus, EAST-ADL architectural elements are compared with their Scania specific counterparts. Follwing this, hardware nodes are added to the meta-model in a standard format inspired from the ADL language but at the same time in a format that is familiar and understandable for Scania engineers. The visualization suggestions for the integration of SW/HW views is one of the final outcomes of multi-view modeling and EAST-ADL investigations in this thesis.

One of the most work- and research-intensive parts of this thesis involved defining what information was needed, finding where that information is located and retrieving it in an efficient way. Defining the information needed was done through planning and research together with other members implicated in the main project. Reaching the interesting information was achieved through meetings with engineers from different departments, which were part of the stakeholder's group of this project. The actual retrieval and processing of the information was realized through the software in a partially automated way.

The specific requirements of this project were both received and deduced from the ESPRESSO project through the requests of the Scania supervisor, Mattias Nyberg. Although within the meetings with different engineers from Scania, other requirements and indications were picked up, they were supposed to be fulfilled only

if they are not in conflict with the existing ones and if the time allows. The meetings in which the engineers were asked about their view on the project and were presented the results, have been called interviews. Mattias specified the definition of done for different levels of completeness of the project:

- For a minimum, the project can be considered done if global variables can be linked to pins on the ECU.
- In a second stage, which is also the desired and the achieved one, the previously mentioned pins of ECUs must connect further on with the sensors and actuators (and other electrical components).
- The 3rd stage would be able to also describe the exact properties of different types of sensors/actuators that can be connected to the ECU pins.

Although the main and only concern was to satisfy the original requirements, suggestions from the interviews and meetings were also collected. Some of this suggestions already fitted in the main requests, others could be used for further improvement:

- Extracting ADC and filter functions is much more complex because part of the process is done in a hardware module and is not reflected in the source code, so this should not be done at this stage.
- Engineers were interested in tracing hardware components up to software modules.
- They also wanted to see the detailed connection network of one subsystem of the truck.

3.1.1 Required information

As is it mentioned in section 2.3.1.2, one of the preliminary steps towards architecture recovery is determining the raw data needed. This step requires initial architectural information which can be found in various system artifacts. Source code, for example, is among the most trustworthy sources of information together with design documents and build scripts. Other than that, complementary hardware information is needed to be used in database construction and view fusion phases to link or map data elements to the right place in the database.

Furthermore, according to EAST-ADL hardware model investigated in section 2.3.2.4, low level HW information is required to construct pure hardware architecture of the vehicle's embedded system. Data sources that contain such information can for instance be found in design drawing and electrical schematics of the vehicle or similar documentations.

Conforming to the research done, the information needed to create an overall architecture of a Scania truck by adding the hardware layer and to store it in a graph database can be grouped as follows:

- Source code
  The source code is needed to extract the connection between the hardware and the software, to map the hardware containing software to the electrical components and to describe the functionality for inputs and outputs.

- External information
  This refers to any information that can be used to describe hardware components and their connections.

3.1.2 Available data sources

The main available data sources at Scania that contain necessary information for this thesis, can be grouped in three main categories:

- ECU source code
- PT
- Component library

Low level software layers responsible for handling of ECU pins and SW/HW interfaces are the favorable architectural elements that can be found within source code. The source code can be accessed through Scania's repository and visioning system, Perforce. Most of the code can be accessed freely with the appropriate credentials. However, sensitive parts of code are excluded from master thesis students. These last described, sensitive code, does not affect the thesis work. The most interesting software layer from the ECU source code has been found to be the ComP platform, i.e. common platform. This is the closest to the low level software which handles hardware and especially I/O pins. The layer contains files which describe the pins according to their functionality and processes some of the inputs in terms of voltage scaling and other properties. Another interesting layer is the middle layer which is called MIDD in the ECUs that are similar or from the COO family and LLAP in the ECUs closer to the EMS. This layer contains much of the functions and software structures responsible to connect the pin concepts to the RTDBs. The last layer which needs to be considered is the RTDB.

From other outspread sources of information at Scania, the PT tool is the best choice for retrieving all E/E components of a truck and their network of connections down to the pin level. In other words, PT holds valuable documents representing electrical schematics in a machine readable format as well as documents that link or map components to their respective software numbers. The latter is crucial for view fusion and also composition of architectural elements in the meta-model and consequently graph database.

In addition, the component library is the key for technical descriptions to the hardware components in the graph database. This file is an excel sheet describing all types of components, their code number, the system they belong to and a short description of their function in the vehicle. The necessity of component library is underscored by the information it provides to engineers and developers and it is easily accessible from the Scania's internal network.

3.1.3 Data extraction process

Because of the experience from previous theses and since the main language used in ESPRESSO is Java, it was decided the the implementation should be continued with this programming language. Figure 3.1 presents the inputs that are fed into the developed toolchain in order to created the overall architecture which can be persisted in the Neo4j graph database.

The four main sources are displayed together with a fifth that is used for mapping the hardware ECUs to the existing software information.

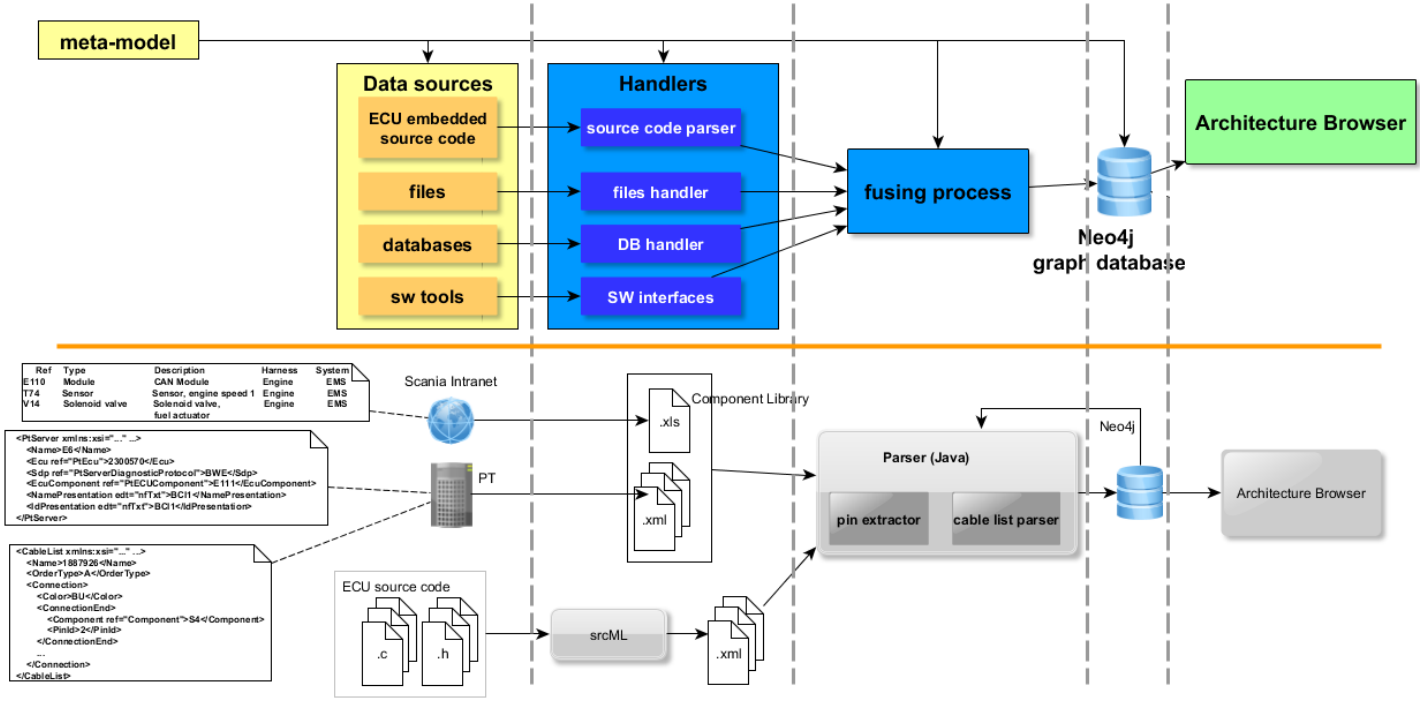

Figure 3.1: Toolchain input sources and flow

For a better overview of the sources and for deciding how the information should be extracted, the sources have been grouped as follows:

- Databases:
  This is the case of the previously processed information stored in Neo4j. The data will be read from the database with the help of a java API and modeled accordingly in order to be further used. This process is automaticly performed by the code.

- Tools:
  This relates to the PT tool mainly, but Perforce was also placed here. The PT application as a tool has its own input in the form of multiple databases, but the output of it is xml structured files. The retrieval of the xml files is done manually, but their processing is completely automatic. Perforce is the

repository where the source code is located. The extraction of the source code was done manually and the software architecture recovery from the source is partially automatic.

- Files:
  The component library excel document is manually downloaded and automatically processed in java using a specific API.

Another approach towards grouping the sources is according to the process used for data extraction. In this situation there are two cases, data extracted and processed by means of software architecture recovery and data extracted and recovered by a custom approach employed in this thesis.

The following describes a more detailed view of the implementation in regards to how the sources are processed throughout the tool-chain. The input and output points for the processing that is accomplished in the tool-chain are described in the Figure 3.2. Although the implementation is not supposed to handle this, the assumption is given that, once integrated, the main project has as a preliminary input a vehicle identification number (VIN). Based on this code, the SOPS file is retrieved and processed. From the SOPS file a series of specific information is recovered, which is used as a next step to determine what ECUs are involved in that particular truck, what software version do those ECUs have and lastly, an array of cable list file names.

At this point the actual work performed in this thesis is used and our tool-chain starts by splitting itself into two distinct processing paths. One path takes as an input the specific ECU versions and analyses the source code for it which will result in the pins description. The other path takes the cable lists from PT as the primary input and processes it. Together with files that describe the connection between the ECU software version and their component code, the hardware components network was built as a result. Besides the cable list, extra description information was added to the hardware components through the component library excel file. Although the implementation stopped here, a further level of detail can be added by describing the individual part numbers for each component code.

As can be seen in Figure 3.1, the toolchain of this thesis is implemented using srcML as a pre-processing tool and JavaSE as the development language for the back-end part. SrcML converts ECU source code to XML file which is then becomes readable by Java applications. PT data sources are also in XML format and are processed afterwards. Since the outcome of the processed data should be linked to the previously stored information in the graph database (Neo4j), connection nodes are identified and extracted from database. Then, new nodes with their respective relationships are created in Neo4j according to retrieved architectural models. Finally, the front-end (Architecture Browser) which is written in C#, is responsible for visualizing the architectural views based on designed abstraction models. However, it is not part of the main work in this thesis.

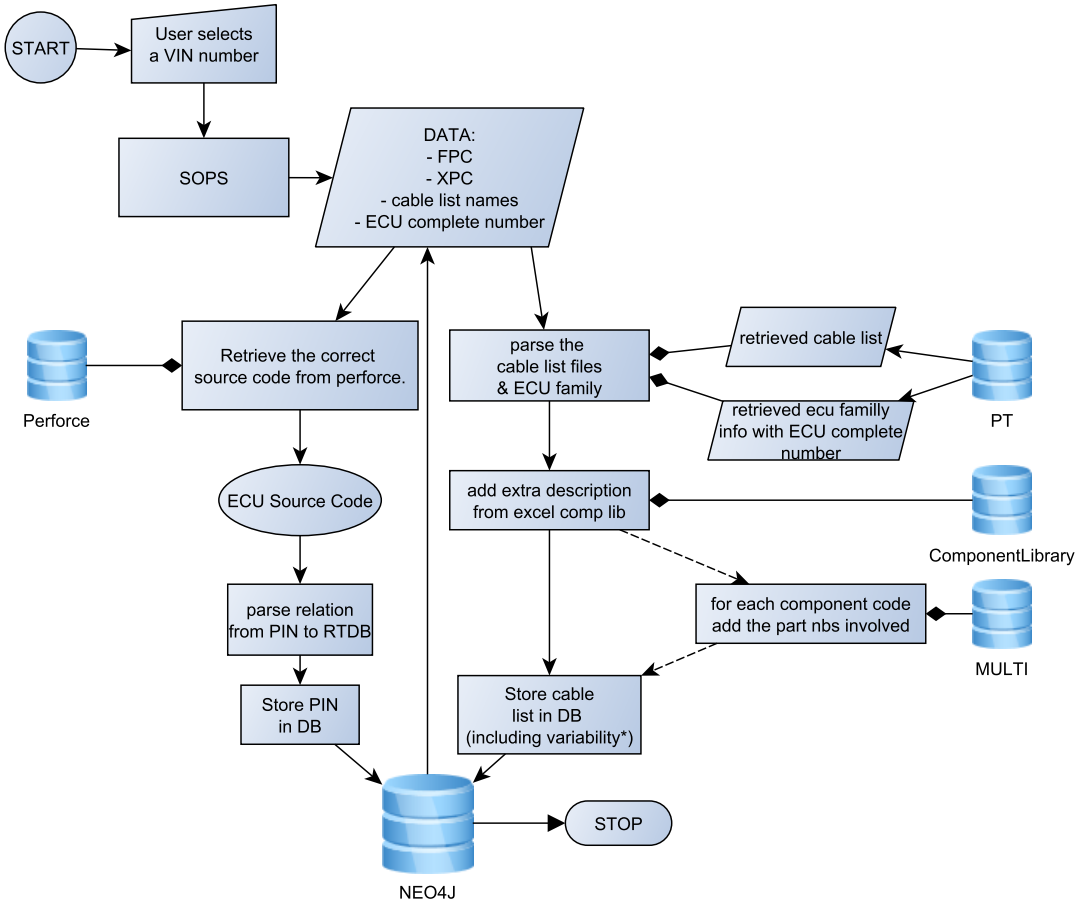

Figure 3.2: Overview of data sources and data flow

3.2 Research methodology and overall workflow

A proper choice of research method should be done not only based on research questions but also in accordance with purpose and goals. This thesis is of a qualitative nature and is dealing with data that should be extracted from different sources and databases available only at Scania. Thus, in this work the research strategy is a case study within the Scania ESPRESSO project focused on HW aspects of sensors and actuators and their corresponding HW/SW connection to specific ECUs in a truck.

The validation task in this thesis is divided between observations and limited interviews. Although the Scania thesis supervisor was the only way of confirming that the work is satisfactory and done, some interviews were performed during normal meetings. Since information was required from a variety of departments, this opportunity was also used to understand the developers and engineers point of view on the subject. The interviews were carried out with the engineers that work in the departments from where the important data sources were identified. The focus was on identifying the elements that made their work efficient. As previously stated, this is because a major part of the work involves studying ECU code and deriving relevant information from it. The engineers were chosen to be the ones who

are responsible for writing the ECU code in order to ask for explanations whenever there is not sufficient descriptive documentation. Results from this thesis were also analysed with some engineers, but their opinion did not influence the results directly, the Scania supervisor's requirements had priority. Every other task, such as learning a necessary tool that can facilitate different processes in each of the above mentioned steps, is considered as the back-end of the research method. Figure 3.3 will be used further on to relate to specific stages of the workflow.

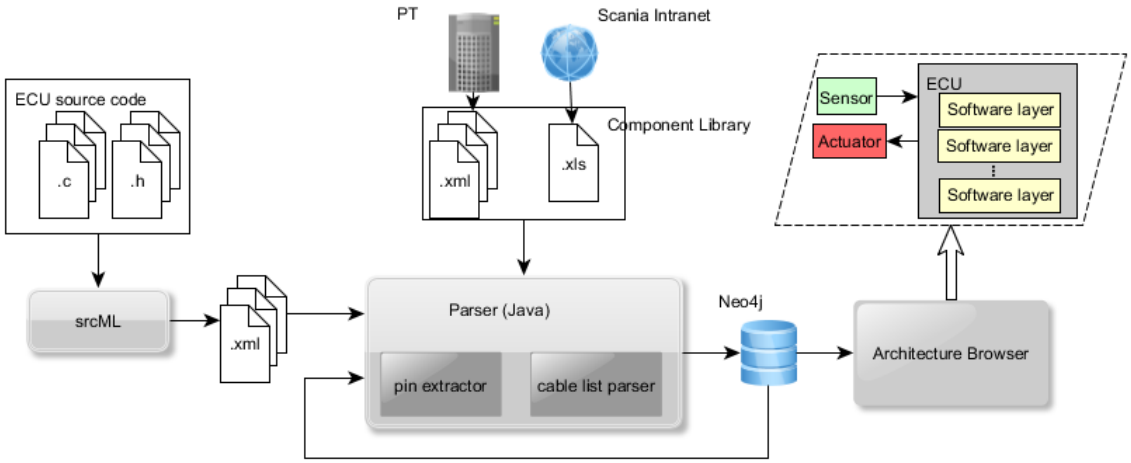

Figure 3.3: Overview of the developed toolchain

3.2.1 Overall project management

To fully describe the workflow environment for the thesis one needs to relate to the main context of the ESPRESSO initiative. The ESPRESSO project is managed according to Agile-Scrum [70] which is already running in research and development units at Scania. Therefore, daily scrum meetings and an iterative approach is selected for reaching the defined goals of the thesis. This process makes early error discovery possible while receiving feedback from the stakeholder group and the supervisor. In addition, one of the first steps towards achieving the goals is the background study which includes, but is not limited to: previous Scania master theses, related papers, research articles, relevant tools and infrastructures that are available at Scania. Hence, the background study, which is necessary to fully understand the research scope and prepare for future challenges, is an ongoing task that should be referred to during different development phases of the thesis. As the final point, the implementation phase is started in parallel with the research study phase. Thus, an iterative approach is taken into account for improving the architectural views based on the newly gathered materials.

3.2.2 Thesis workflow

A major part of this project comprises of searching and finding useful and relevant information that describes how different HW parts are linked to each specific ECU. This step entails spending a considerate amount of research time on investigating

and understanding the .c/.h sources. It also means that time is spent on analysing available documents containing HW connections or consulting Scania engineers in order to find out their approach to retrieve the required information. As a result, a custom model both for HW and database is designed so that essential HW features can be represented in the most efficient abstract format.

On the other hand, the practical part of the thesis work focuses on designing a model for the recovered architecture that can be also populated with information on sensor and actuator hardware and their connections through software in the application modules. This model is used in turn to store the data in a graph database. Since the main project contains several different implementation points assigned to other master projects and PhD researchers, integration must also be kept in consideration.

On a basic level the workflow can be described as retrieval of information and preprocessing in a first stage. This is followed by the main processing and aggregation of the retrieved data. The last stage involves storing the new elements in a graph database based on a developed structure, Figure 3.3.

In order to achieve the aforementioned picture, specific processes are implemented in the tool chain. In other words, the list below describes the sequential tasks that take place during the execution of the tool chain.

First, the source code for specific ECUs is taken from the Scania repository, perforce, and stored locally with the same path layout. Because in the previous step redundant files are also copied, a file crawler is used to navigate through all folders and create an identical folder structure with processed files of interest (this files are c code files, header files and xml files). Next, srcML is employed to parse the .c and .h files of the ECUs to xml descriptions. Then, the xml files are taken as input for parsers created in JavaSE language, where with the help of the xpath library the relevant information is extracted. At this point the description of the pin from the source code is constructed around a structure and stored to the database by linking it to existing elements in the database. Since one of the aims is to create a modular structure of the software, the next step is to retrieve the pin related information from the database together with the ECUs the pins belong to.

Moreover, extra information is taken from PT, cable lists and component library, other Scania data sources, which describe the ECU versions to component code mappings, the electrical network of all hardware components that are used in a truck and user description about the hardware components. This information is combined with the existing one from the graph database according to a structure and stored back. The last part involves presenting the data stored in the database in a visual way.

3.3 Meta-Model

This thesis is built upon previous implementation with the aim to improve it with new functionality. The desired parts that were missing in the previous implementation can be summed up to: retrieval of low level software layers, retrieval of the mapping between the global variables and the ECU pins described in the software and in the end, the external connections from the pins to other hardware compo-

nents.

Therefore, an abstract view of the ECU system architecture of a Scania truck is depicted in order to illustrate the goals of this project. As can be seen in Figure 3.4, software elements retrieved previously by means of software architecture recovery are connected to their physical hardware counterparts by the current implementation. In this context, the aim of the implementation is to add the hardware elements and provide a connection between these and the software modules which they affect or by which they are affected.

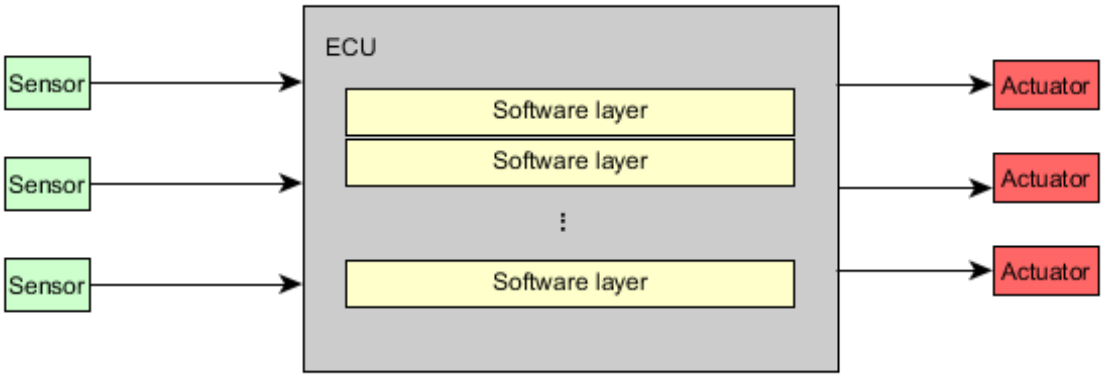

Figure 3.4: High level abstraction of the accomplished HW/SW architecture

In a more detailed view and with reference to Figure 3.5, the implementation focuses on one hand on the ECU source code which describes the configuration of the pins. This includes the RTDB variables which either influence the pins or which store the pin information. On the other hand the thesis project also implements a way to retrieve and store the connection the ECU pins have with the rest of the hardware components in the E/E systems of a truck.

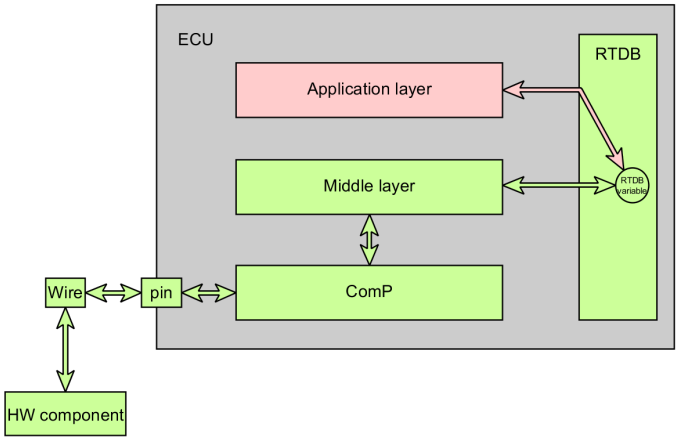

Figure 3.5: Abstract concept of the accomplished work

To have a clear understanding of what information is necessary, how it should be combined with other elements and how it should be stored, an extension to the existing meta-model has been proposed. Figure 3.6 shows a simplified version of the

concept with the focus on the mapping between the existing structure, the software, and the developed model, the hardware. The connection between the two concepts, the pin, has also been introduced with this thesis.

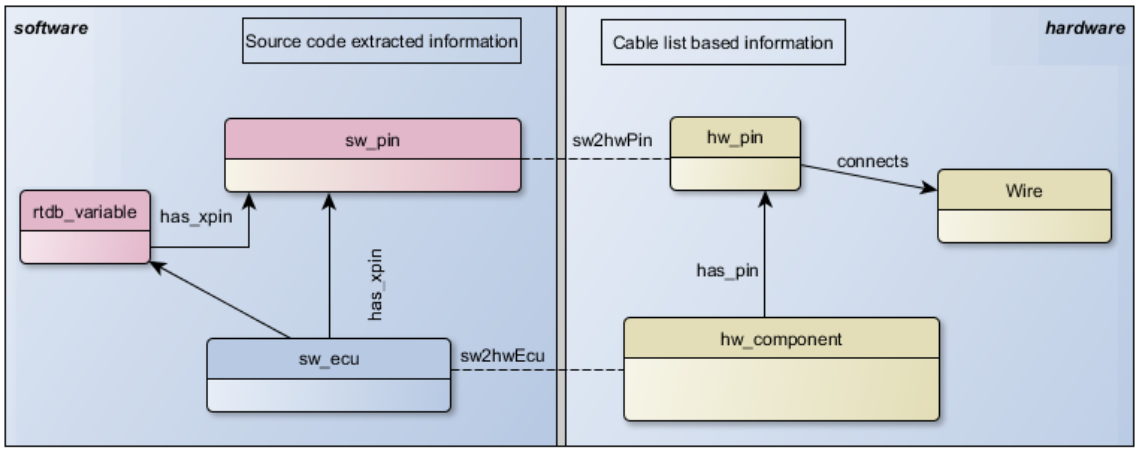

Figure 3.6: Simplified meta-model focused on hardware architectural elements

This model serves also as a base for the visualisation part in Architecture Browser (AB), the starting point of this thesis. AB is a tool developed at Scania for architectural description, to represent hardware architecture of E/E system. Figure 3.8 shows the detailed model that has been proposed and employed. It also represents the real configuration of electrical parts in a Scania vehicle. This diagram is inspired from the hardware model of EAST-ADL, Figure 3.7, and facilitates the use case of hardware nodes matching during database management processes. In Figure 3.8, the hardwareComp represents all hardware components including ECUs, sensors, actuators, connectors and others. Also, pin is the placeholder for all types of pins in the diagram that belong to hardware components and provide a connection point to other elements through wires. The analysis of the source code and other information resulted in creating two types of pins for the meta-model. The pin information recovered from the cable lists is described as the concept of physical or hardware pin, or simply pin. This is the actual pin on the ECU casing or other hardware components used in a vehicle. The pin description that can be assembled from the source code is named "xpin". Xpin is the software representation of the real physical pin of an ECU created in the common platform layer. However, since it is a virtual concept defined in the source code, it does not imply a one to one mapping to physical pins. This could mean that one real ECU pin can have one or several representations in the software.

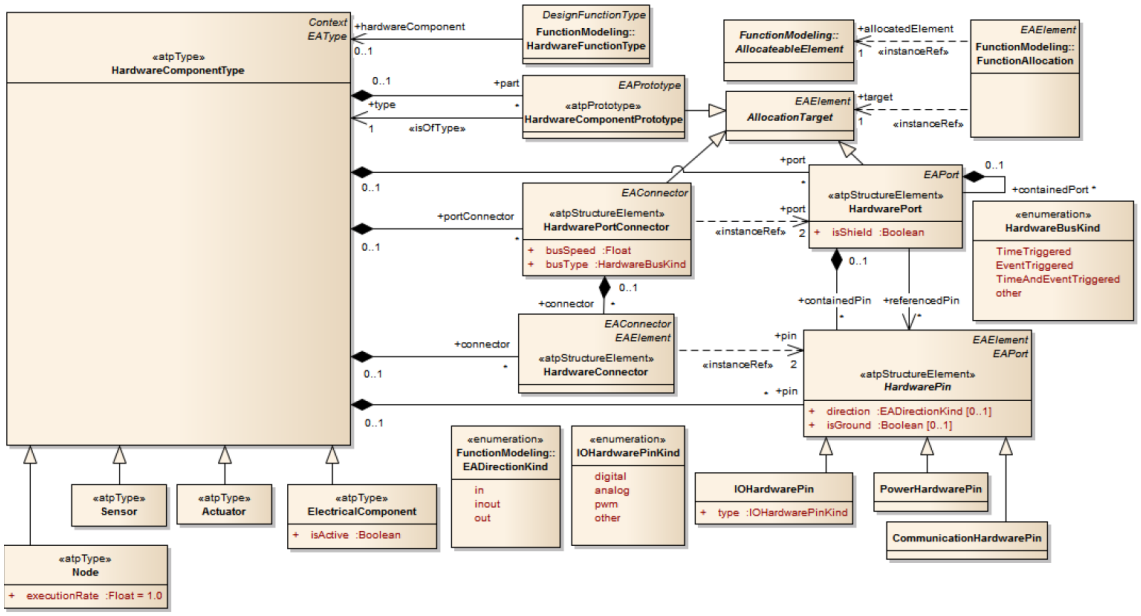

Figure 3.7: EAST-ADL hardware model

In a different aspect, it is obvious that the software and hardware model elements that are recovered and retrieved from diverse data sources ought to be represented and linked to previous elements in the meta-model. Some of these elements represent different views of the same concept. For instance, ECU components have modeling elements both from a hardware perspective and software. Thus, composition processes have been used to automatically find the matching elements and merge them with suitable fusion operators [53, 54].

Resulting from the previously stated, the contribution is in this sense that this project adds the capability to the meta-model to handle hardware components. The effort was put into designing a sufficient hardware model that would not clutter the general meta-model.

The components of the hardware model are described in accordance with the elements found in the cable lists. The scenario is that components connect to other components through pins and wires. This can be interpreted as a connection and a cable list can contain several connections so we can consider all wires from connections within a cable list as being a harness. With this logic, the hardware component, the pin, wire and harness can be defined as elements describing hardware, Figure 3.8.

Together with the hardware elements, directional relations are also partially introduced in the ECU area to better explain the connections. The hardware component has the relation "has pin" towards the element "pin". "Pin" in turn "connects" towards a "wire". Lastly, "harness" "has wire" towards "wire".

The hardware section (area) connects to the rest of the meta-model in two points. The hardware components will be connected to the ECU models recovered from the source code if they correspond to the ECUs. The pin will connect to the software wise retrieved pins only if their name and ECUs match.

As mentioned before, the pin elements retrieved from the source code are also handled in this project. The pins are accessible from the software with two relations

that are mutually exclusive. If a pin can be traced down to one or multiple global variables, than that variable "associates with" the pin. In the opposite situation, if the pin cannot be tracked down to a specific global variable, than the ECU "has" that pin directly. It is to be noticed that multiple global variables can be associated to a single software pin, but in the same way a global variable can associate to multiple software pins. The hardware pin can also have in turn multiple software pins which associate to it. On the other hand, software pins don't necessarily have to be connected to a hardware counterpart. The last scenarios will be described in more detail in chapter 4.

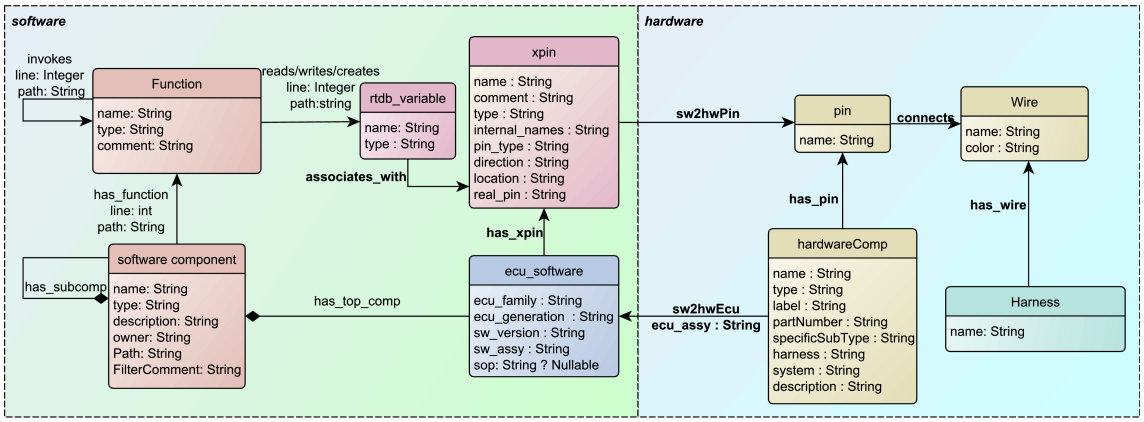

Figure 3.8: Detailed meta-model focused on hardware

3.3.1 ISO 26262 influence

Apart from system level development, two main clauses of ISO 26262 focus on hardware and software production levels which were taken into consideration while designing the hardware model. These parts are relevant to our work since this thesis covers both hardware architecture and also parts of software architecture that are responsible for handling sensors and actuators in the ECU source code. However, the emphasis is on retrieving sensors/actuators architectural models. Hence, referenced to part 5 of the standard, the combination of hardware architecture and hardware-software interface (HSI) specifications is required for investigating further safety hazards and violations [2]. HSI identifies hardware components that are handled by software applications or hardware components that provide information for the software. Actuators and sensors are examples of such devices. Therefore, it is obvious that hardware architecture and the interfaces between software and hardware components should be well documented or it should be extracted from the implementation, to be able to perform safety evaluations on the embedded system of a vehicle.

The focus in this thesis is on a limited number of architectural views both because of complexity of the source code and time limitations. This thesis mainly covers component perspectives of E/E system and software of specified ECUs.

SAR techniques are divided into three main categories whether they are manual or automatic. The source code analysed in this thesis proves to be highly complex not only because of the extensive use of pointers and temporary variables, but also

because of the amount of code. Motivated by this, the best approach to extract abstract models from source files was decided to be lexical analysis and the search for specific patterns. This means that first the source code should be studied thoroughly to find possible starting points and also what patterns have been used in the code. Then a combination of pattern matching and lexical queries shall be used to find architectural elements and their relationships. This process is a Semi-automated approach and could be sub-branches of abstraction and investigation groups. Besides, the objective of this thesis work is to recover and build structural models from available artifacts in source code or other data sources which is sufficiently achievable by static recovery.

3.4 Tool implementation

The following sub-chapter describes in detail the implementation process. The implementation is achieved in accordance with the methodology and it reflects in practice the analysed theory. Some of the tasks could be done in parallel while between others, a sequential dependency had to be respected.

As it was mentioned before, the output of the tool is presented using the existing Neo4j graph database instance. The interaction with the Neo4j database is established using the official Neo4j API libraries.

3.4.1 Input data processing

The conceptual separation of the input was done based on the localization in relation to the source code. The processing of source code and the information external to it are described separately. This two inputs are not dependant on each other in terms of processing and could be approached in parallel. On the other hand the information that is external to the source code can not be connected properly and persisted in the database unless source code specific elements are already stored in the database.

3.4.1.1 Source code information processing

The implementation described in this subsection employs the theory of software architecture recovery for retrieving elements specific to hardware abstractions.

The source code, structured in the form of a folder tree with .c, .h, .xml and other types of files, is copied locally from Perforce. After this, the .c and .h files are parsed with the help of a srcML parser tool called src2srcml. All flags for this tool are activated to get as much information as possible from the files. The output of the srcML tool is stored as xml files with a specific structure in a folder tree that mirrors the original.

Before the source code is handled, the existing database is queried with the purpose of retrieving nodes for the existing ECUs together with their connections towards RTDBs. The querying is performed from the java code using the Neo4j's API with the option for Cypher language query. This procedure is divided into two sequential retrievals. The first one traverses the nodes to find the ECU elements and

returns them. The second procedure involves traversing the nodes from the ECU to RTDBs and returning the unique RTDB variables, where the ECU parameters are specified.

After the existing information about the ECUs and their RTDBs is available, as the meta-model describes it, the main part of the processing starts. Since the original source code has now been transferred to an xml structure, the XPath library is used to handle the xml specific syntax and to aid pattern matching. For each set of ECU with RTDBs, the xml converted source code is processed. The aim is to determine the pin descriptions that can be linked to the previously mentioned RTDBs. The processing of the xml converted source code starts with a file that can be found in all ECUs source code and which contains the software wise description of the pins. The location of the file is in the low level software layer, the ComP platform. This file, xpin_list.h, enumerates the pins from a functionality point of view as it can be inferred from the naming. Some of the information available refers to the pin's functions like if it is analog, input, internal, etc. In the next phase the global variable, RTDB, is tracked down and it is determined if it is either written or read based on the function that handles it. The tracking of the global variables is performed through the source code of the middle layer. After all possible connections from software pins to RTDBs have been tracked down, the information is stored in a pin object that will be persisted in the Neo4j database. Pins that do not have any RTDBs associated are associated via a direct relation to the ECU.

Figure 3.9 presents in a high level abstraction the main processes that influence the source code input and external information.

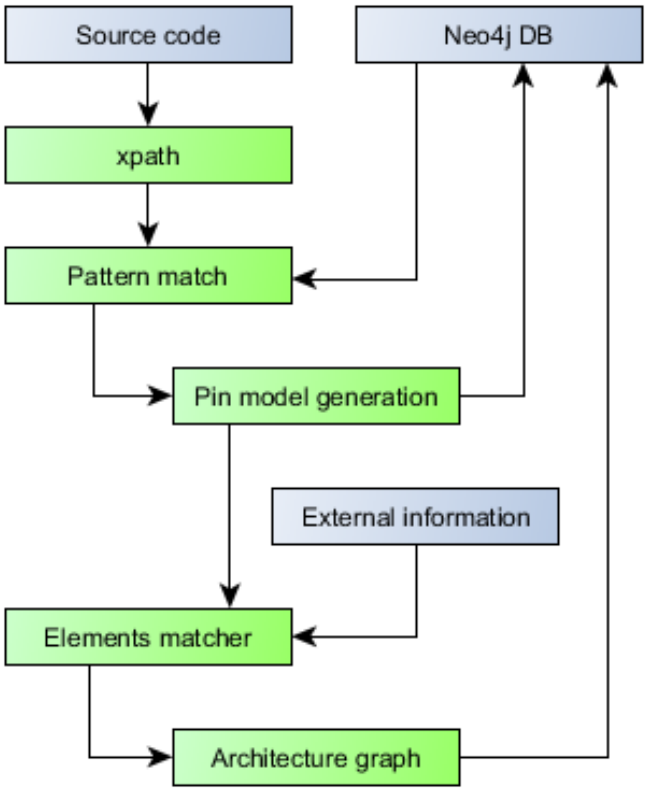

Figure 3.9: Basic flow of data

The algorithm described previously can be traced in higher detail in Figure 3.10. For a better understanding of the work, the processes have been structured on three layers: data source, processing and modelling. The "data sources" is represented by databases and different types of sources which serve as input and output for persistence. In "processing" activities related to data retrieval and storing are performed together with basic filtering between groups of data. The "modelling" layer handles the main fusion between the gathered data and prepares to entities for persistence.

In the case of the source code, as it is also mentioned previously, the pin description from the source code is matched against the exiting global variables from the Neo4j database. The results are structured in the form of pins from source code, as described by the meta-model, and stored back to Neo4j.

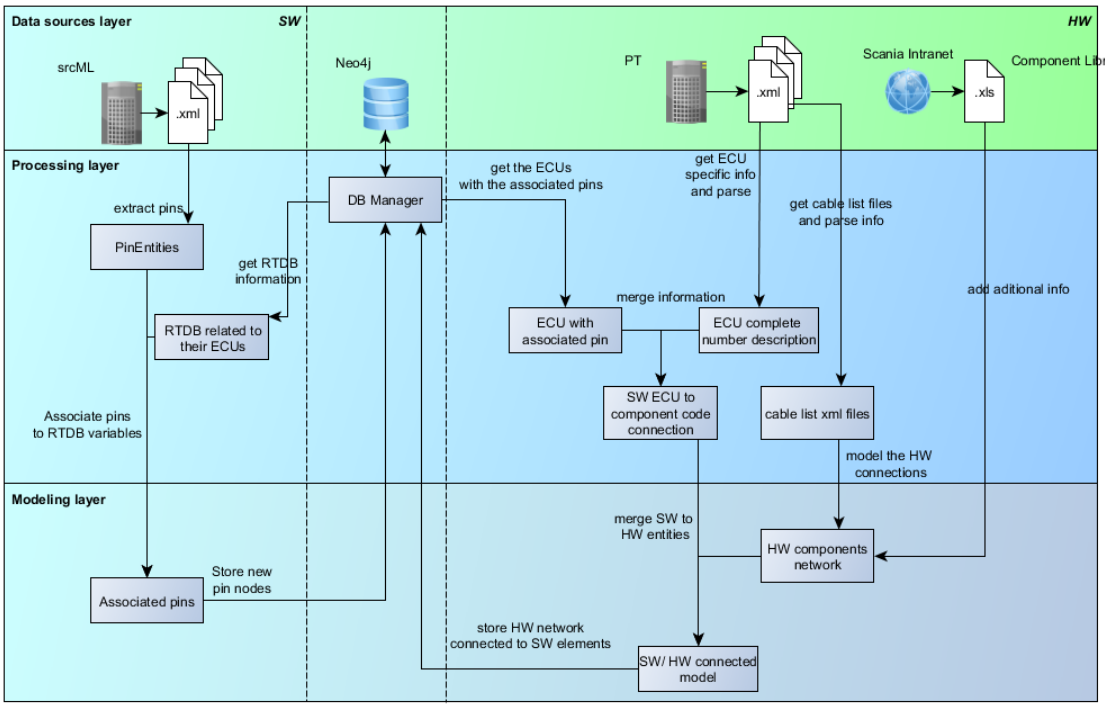

Figure 3.10: Detailed flow of data for implementation

3.4.1.2 External data sources information processing

The source code information processing algorithm described previously can be viewed as standalone. Its main outcome is to connect existing RTDBs from the database to their corresponding pins, which are also retrieved from the source code. The second part of the project, or the second algorithm, deals mainly with the process of mapping the pins and ECUs constructed from the source code with the pins and ECUs extracted from external data sources, Figure 3.9. By extrapolation, this algorithm deals with all hardware components and the connections between them, down to the pin level of detail.

The first part of the algorithm collects from the database the ECUs nodes with their associated pins. In this, the link is added that maps each version of an ECU to a certain component code with the help of the ECU family xml files from PT, giving the first output. The information from this collection will be merged with data coming from files that describe the hardware connection between electrical components down to the pin level. This files are called cable lists and are generated based on the schematics of the electrical wiring in a Scania vehicle. Extra details are added to this network of connections by parsing an excel file responsible for describing and categorizing the components. In the processing layer, the sources are matched and filtered in order to provide complete results. For each new component code, a match is requested from the excel document. The output of the processes is fused into the final model. The resulting model of interconnected components through pins and wires is stored in the Neo4j database completing the picture of the AB with hardware information, Figure 3.10.

This part of the project is important because it involves fusing different pieces

of information from different sources in order to get a more complete view with the hardware elements. The hardware retrieval process has three entry points as inputs, one of which is used to link the software wise data with the hardware. The other two sources are responsible for describing the connections and enhancing the understanding of the geometrical location of components and description. Figure 3.11 presents samples of how the sources would be structured. Both the cable lists and component library are needed to construct a human understandable model of the hardware elements with description of their functionalitie.

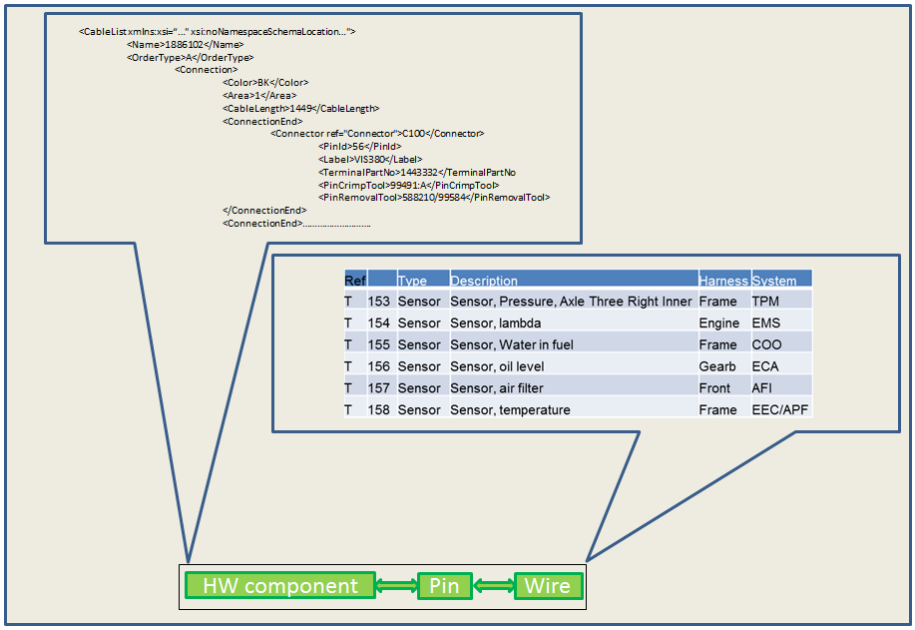

Figure 3.11: Composition of cable list and component library

The cable list, Figure 3.12, describes connections with two ends. Each end having a pin and a component to which it belongs to. The software implemented uses JavaSE with xpath to read and parse each cable list file and extract connections. The connections are stored in memory in the same way they are described in the file, in the form of component-pin-wire-pin-component. Besides the connections, other information is also available in the files, like the color of the cable, the label used in electrical schematics and others. For each of this connections the database is checked if there is already the specified component. The next check is performed on the pins of the component in the same way. The only element which does not need checking is the cable. Multiple cable can exist between the same pins. This scenario is brought forth by the variability factor present in Scania. Even though the connection might be similar or even identical, different trucks might require longer cables or the inputs to pins might change.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--INFO: No connector information found in cable list from Modarc.-->
<CableList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="./schema/CableList.xsd">
    <Name>1895805</Name>
    <OrderType>S</OrderType>
    <Connection>
        <Color>GN</Color>
        <Area>0.75</Area>
        <CableLength>421</CableLength>
        <ConnectionEnd>
            <Component ref="Component">P2</Component>
            <PinId>B-18</PinId>
            <Label>BOL1253</Label>
            <TerminalPartNo>1448955</TerminalPartNo>
            <PinCrimpTool>99495:A</PinCrimpTool>
            <PinRemovalTool>99481/99579</PinRemovalTool>
        </ConnectionEnd>
        <ConnectionEnd>
            <Component ref="Component">R77</Component>
            <PinId>4</PinId>
            <Label>BOL1253</Label>
            <TerminalPartNo>815697</TerminalPartNo>
            <PinCrimpTool>588202:1</PinCrimpTool>
        </ConnectionEnd>
    </Connection>
</CableList>
```

Figure 3.12: Cable list for implementation

Besides the cable list, the component library excel document provides extra descriptive information for a better understanding of a component. A sample is presented from the component library excel in Figure 3.13. This file contains all component code together with human interpretable description and specification of geometrical location. Through the algorithm, each component code's description is enhanced by parsing the excel file and retrieving the line which matches. The component object initialized in the code is extended with this extra descriptions from the excel file.

Ref		Type	Description	Harness	System
T	153	Sensor	Sensor, Pressure, Axle Three Right Inner	Frame	TPM
T	154	Sensor	Sensor, lambda	Engine	EMS
T	155	Sensor	Sensor, Water in fuel	Frame	COO
T	156	Sensor	Sensor, oil level	Gearb	ECA
T	157	Sensor	Sensor, air filter	Front	AFI
T	158	Sensor	Sensor, temperature	Frame	EEC/APF

Figure 3.13: Component library layout for implementation

3.4.2 Composition

As it can be inferred from Figure 3.13, the overall architecture reconstructed in this thesis is based on model composition. The source code, cable lists and component libraries can be seen as products of different departments, but which together can express the overall system to certain level of detail.

The retrieved information from the source code represents the software model of the ECU system. By combining the software models of multiple ECUS the software architecture of a truck can be defined.

The cable list is used as a hardware-electrical model to present the electrical components and the connections between them. This work uses the hardware electrical model to extend the system understanding by connecting it to the previously described software architecture. The mapping between the two is done with the help of the ECU family files.

The component library can be used to ascertain the functionality of an electrical component as well as to estimate the geometrical location of it within the physical truck. This is why this file was used as an input to enhance the overall architecture.

3.4.3 Visualization

In the case of this thesis, the long term aim towards the end-users is to provide them with virtual views made by filtering out unwanted information from a single base meta-model.

The involvement in the visualization part is constrained to designing the concepts for the graphical parts and suggestion on different kinds of views. The main goal in this was to be able to visualize the different electrical hardware components within a truck and the connections between them. This was accomplished by using the cable lists and other external information. Figure 3.14 visualizes the concept of interconnected electrical hardware components. It is not the interface, instead it was used as a suggestion on how visual representation could be done and it also reflects the aim of the thesis.

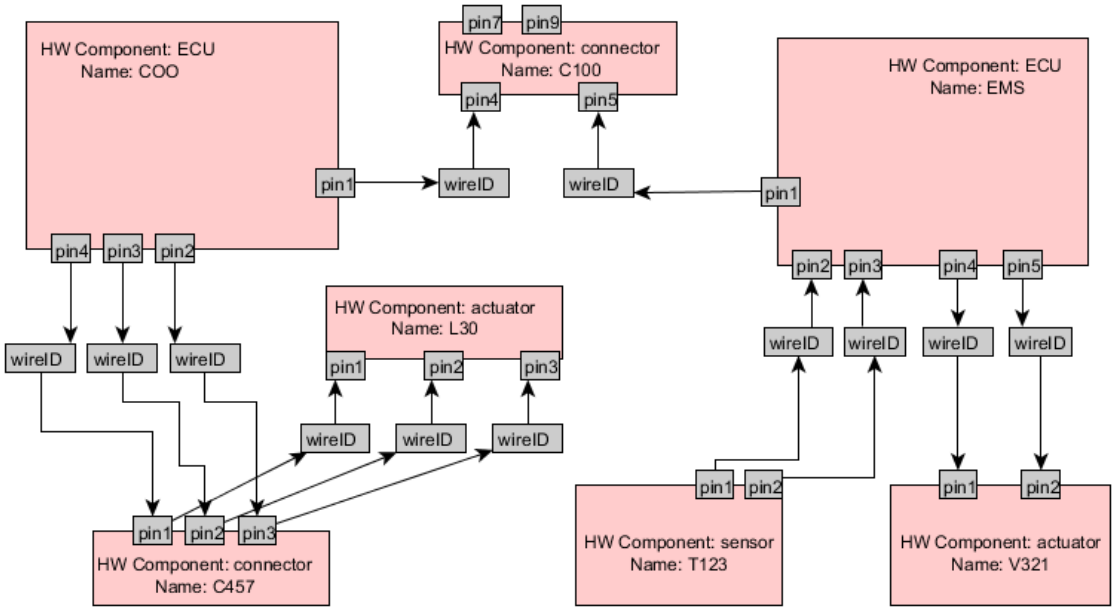

Figure 3.14: Hardware visualisation concept

A second goal was to include the software model and to have a general overview of the entire SW/HW system of a truck. Figure 3.15 presents again the concept

only of a model that contains the interconnected hardware and the links towards the software parts that influence it.

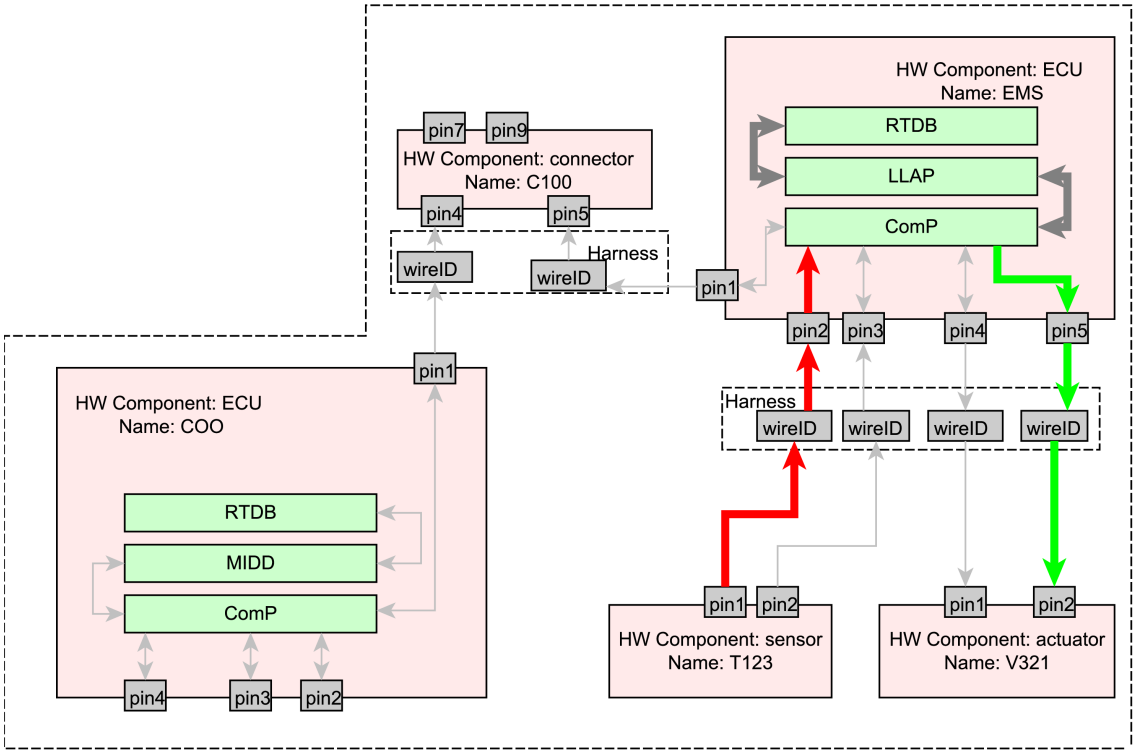

Figure 3.15: Hardware and software visualisation concept

The visualization took inspiration from existing ADL languages and their representation. The major influence came once more from EAST-ADL. Although ADLs are used as a top-down model based approach for software implementation, the concepts and representation part can be successfully converted to suit a bottom-up recovery of hardware and the following visual construction of it.

4. Results and Analysis

Following the design and implementation of the proposed solution, a java software application has been produced that stores and connects hardware elements to their corresponding software counterparts in a graph database. This chapter presents the results of the accomplished work with regard to different aspects.

The achieved results also reflect as answers to the research questions which are defined in Section 1.2. For better interpretation and basic validation purposes, the results are displayed as screen-shots from the graph database and from Architecture Browser, the tool that visualizes the stored data. The visualization in itself is a result since part of the research was also to provide suggestions in this direction.

For simplicity, some basic terms will be defined that are used many times in the rest of this chapter. These terms are defined previously in the thesis and are Scania specific. However, describing them here makes it easier for reader to follow the provided information.

Software ECU, in the graph database, is a node that is directly connected to root in one side and on the other side it is linked to the software application components retrieved from ECU source code. In one way, software ECU represents the software version number of the source code that is recovered. And in another way, it is the linking point of software with its corresponding hardware nodes in the database.

Hardware ECU is the hardware component that is described in cable list and component library as an ECU family. E44, for example is the component code for EMS ECU family.

RTDB variables also refereed to as global variables or real-time variables. In the ECU software, these variables are responsible for communication between functions within either the same or different application layers. In addition, RTDBs store shared information between different modules and are the gateways between application components and low level software platform.

Hardware component is any electrical or electronic part described in the cable list and component library that can be used in a vehicle.

The user interface of the graph database is used to display and check the results of the processing phases. In a first stage, RTDBs that are influenced or affect specific xpins, are linked to them in the database. However, the developed solution did not manage to analyse and connect some xpins with any corresponding RTDBs. But this will be discussed further in the next chapter. The following images present the results as visualized in the graph database user interface. Different situations have been observed, as for instance cases in which one RTDB is associated to multiple xpins and also occurrences where one xpin influences many RTDBs.

From Figure 4.1 it can be seen that one xpin that is retrieved form the source code can be associated with multiple RTDBs. The relationships between these nodes in the database are assigned according to the meta-model. In other words, a global variable node which is linked to an xpin with "*has_xpin*" relation, indirectly

attached to a software ECU node. The figure also displays the mapping between the source code based ECU node and the node that was created form the information in the cable list. The information seen in the figure is not the complete architecture model. Instead, It is chosen to be displayed in the user interface of the data base to emphasise the main attributes and information that could be recovered about the xpins from the source code.

In addition, another important aspect that is shown in the picture is the critical information about xpins. In fact, type and direction of ECU ports that are retrieved from the source code are described in the database as properties of xpins. This information improves the quality of system understanding when it combines with hardware components. For example, an input xpin with pulstrain type in the database indicates the flow of information to the pin as well as expected attribute of the connected signal.

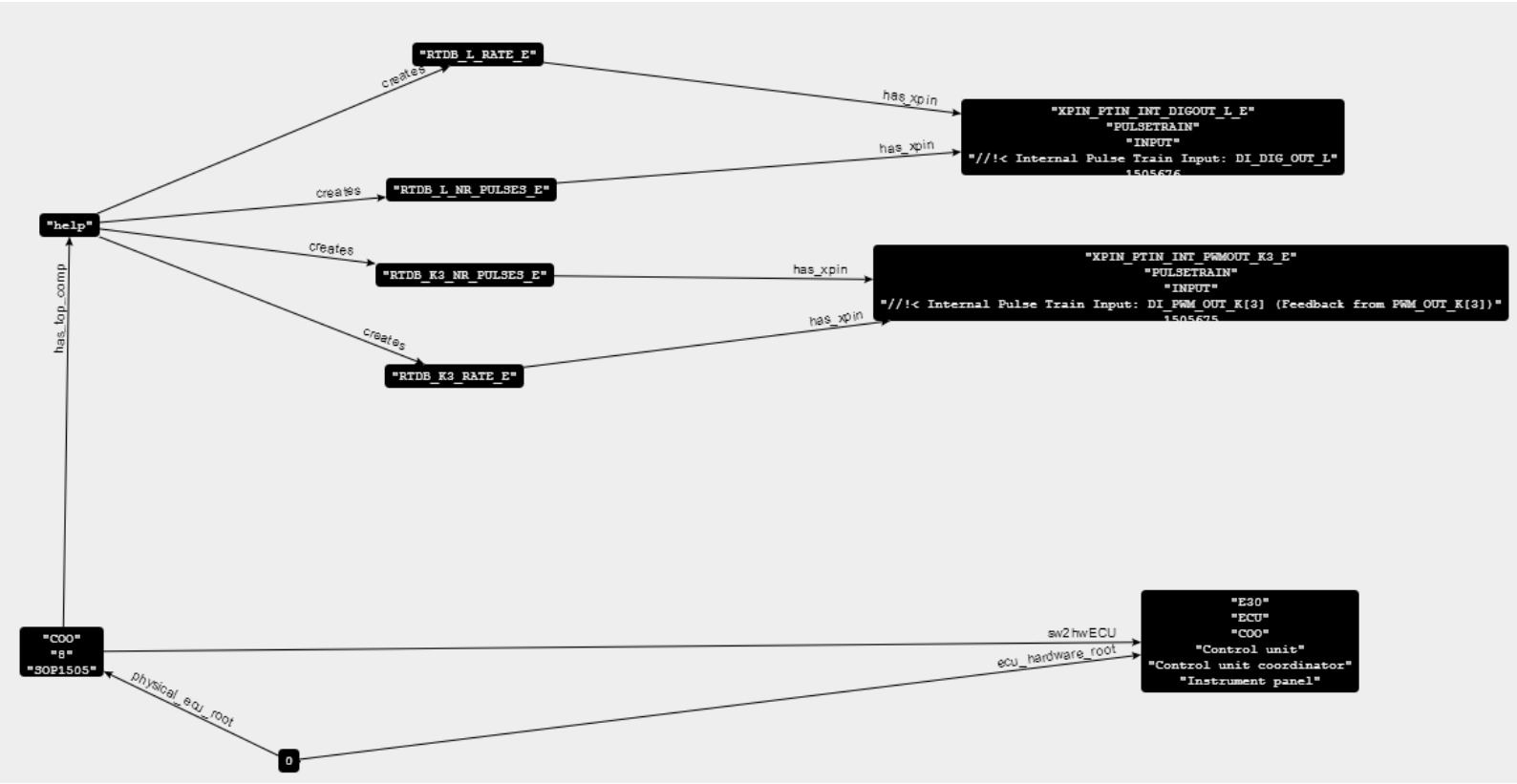

Figure 4.1: Database view of the software pin to RTDB connection

Figure 4.2 focuses on the mapping between physical pins and their equivalent xpins from the same ECU. More specifically, it shows the multi-view modeling of one concept which is ECU pin both in software and hardware aspects. This figure has another interesting situation which is the connection of multiple RTDBs to one xpin. Here it can be noticed that the xpin, is mapped towards the ECU pin name that is retrieved from the cable list. The mapping can be recognised as being based first on the same hardware component that is only the ECU type since only the ECU has source code available. The second mapping element is the pin's naming

convention. It can be seen that in both types of pin description, from source code and from cable list, the name contains the actual pin name.

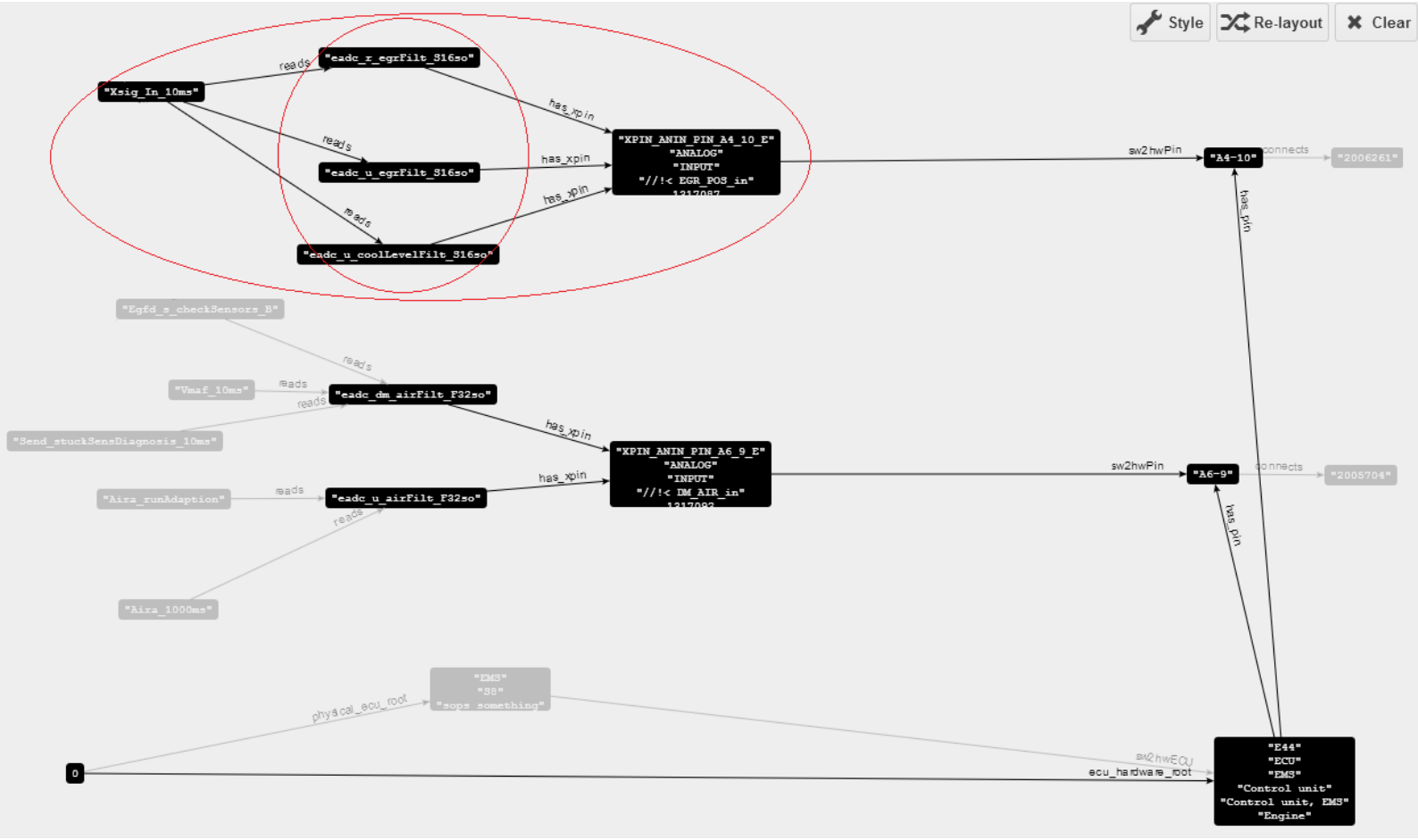

Figure 4.2: Second database view of the software pin to RTDB connection

The Neo4J database web interface is a good method of checking what the database contains after the developed tool-chain is run. It does not emphasize the results though. For this a previous tool was used, Architecture Browser, to which suggestions were made on how to present visually the hardware to software connections and the involved components. AB's main purpose is to visualize different views of Embedded systems. The visualisation in AB occurs in the same way as it is shown in Figure 4.3. The figure presents a simple case in which an RTDB variable can be seen connected to an xpin, both recovered from the source code. The arrow between the RTDB and the xpin shows the flow of data in this case which means the xpin is an output port and its state is controlled by the function that is connected to that RTDB. Further on, the filled circle represents the hardware ECU pin. The connection continues further to another pin inside M1 hardware component. In the down left side of the picture, M1 is described as starter motor. Hence, generally speaking, by looking at the picture below, one can understand and trace the control flow from an application component down to RTDB, xpin, hardware pin and finally the M1 actuator.

The ECU case is special in the context of hardware elements. Only the ECU and its pins have software and hardware models and representations that need to be mapped and matched to each other. The rest of the hardware components are only described by the cable list and component library.

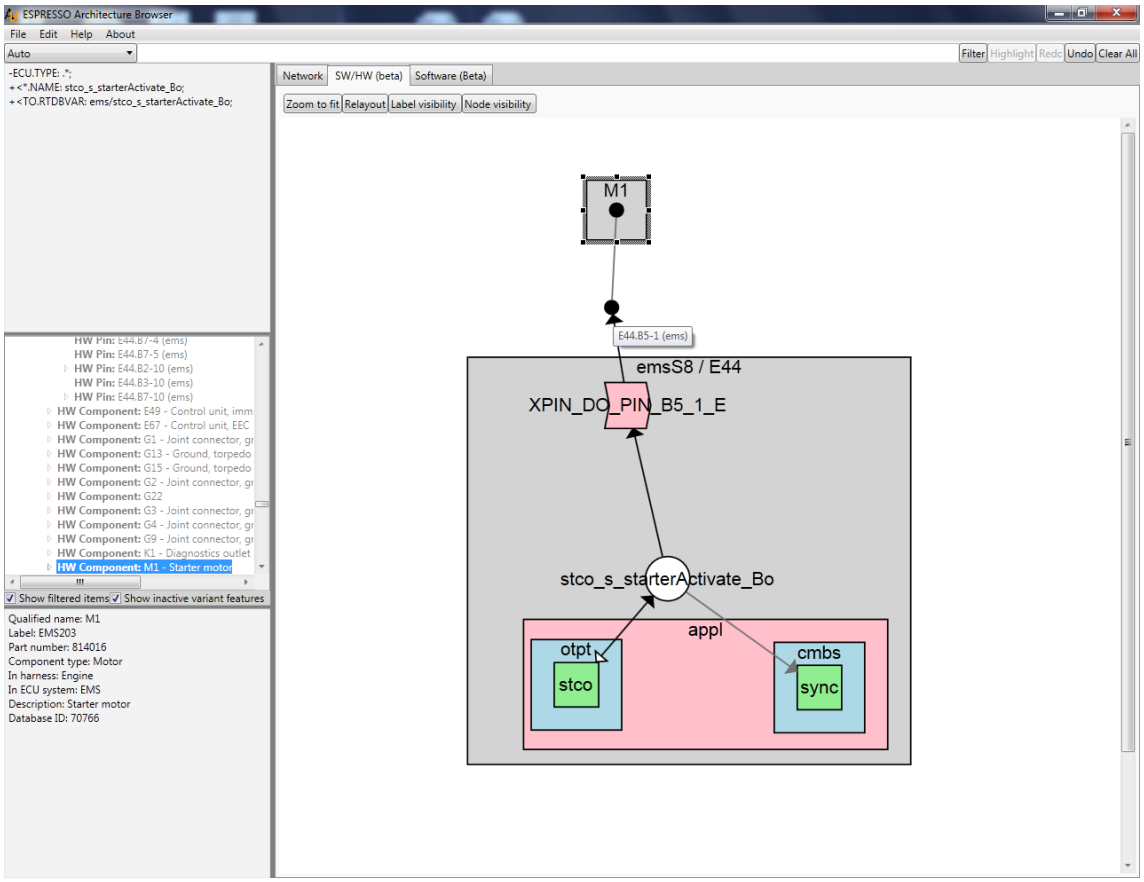

Figure 4.3: AB view of the software pin to RTDB connection

For a more extended view, Figure 4.4 displays the EMS S8 ECU with some of its connections to sensors and actuators and with the RTDBs and xpins. This case is not focused on the connections with external hardware, but displays different xpin and pin scenarios. The scenario puts forward some situations in which the developed recovery solution did not manage to cover all the connections.

As it can be seen from down right and up left sides of Figure 4.4, some pins of V107 and T123 hardware components, do not connect to any xpin in the software. Instead, the connections end at the ECU pin. A reason could be that the pins are from power supply which sometimes are not described properly. Another explanation is that some xpins are handled in software layers that are not investigated in this thesis. Moreover, this application solution does not cover every other possible scenario. The investigation for this together with a thorough validation and testing are suggested in the next chapters as future work.

Another visible item in the screen-shot is an xpin without any connected RTDB. This can happen also for a variety of reasons. One of the more plausible ones is that the xpin only defined in the code, but no functionality is yet implemented.

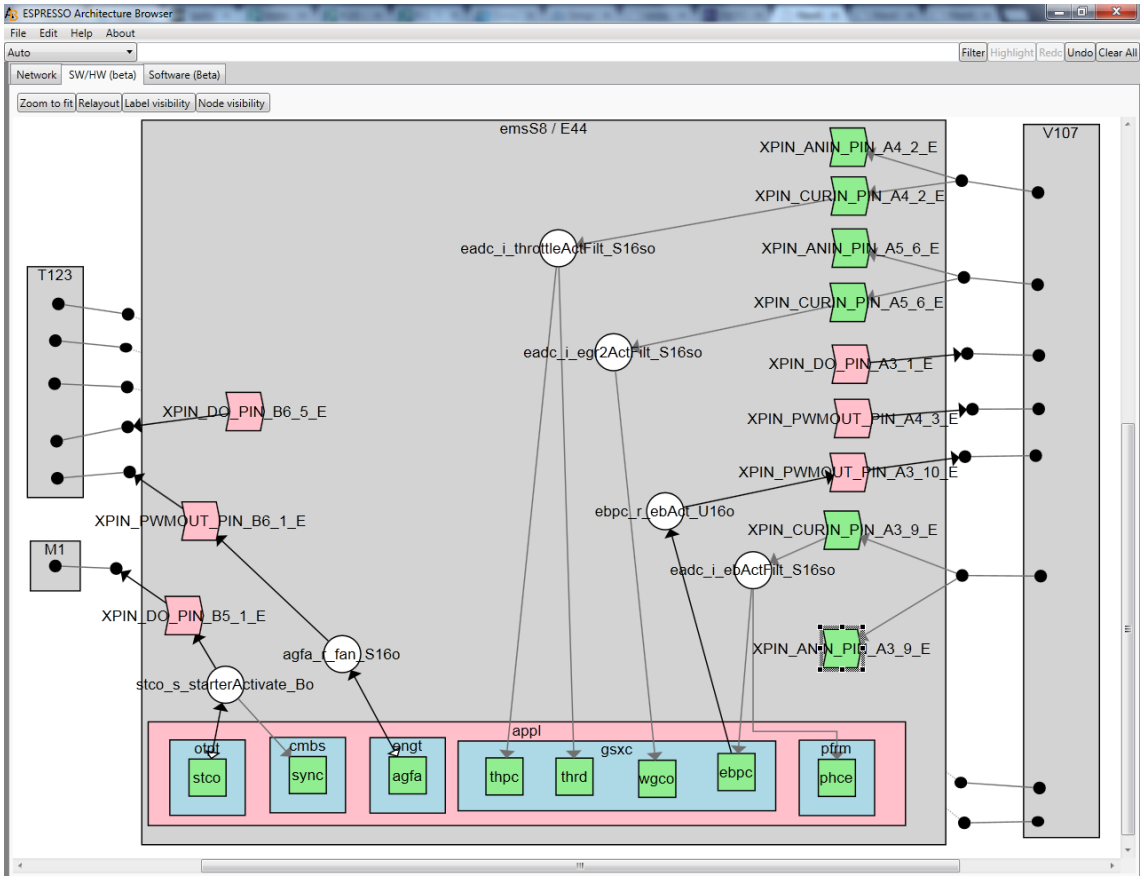

Figure 4.4: Second AB view of the software pin to RTDB connection

The second stage of the results shows how different hardware elements are interconnected. Therefore, it can be said that pure hardware architecture is represented in the following pictures. Figure 4.5 portrays a simple connection between two hardware elements which have one wire in common. The name of the wire, which is also the name of the harness, is visible in this screen-shot from the database UI.

In the case of non ECU hardware components, the pin properties or description cannot be inferred. This is because there is no underlying software that can describe if the pin is input or output and so on.

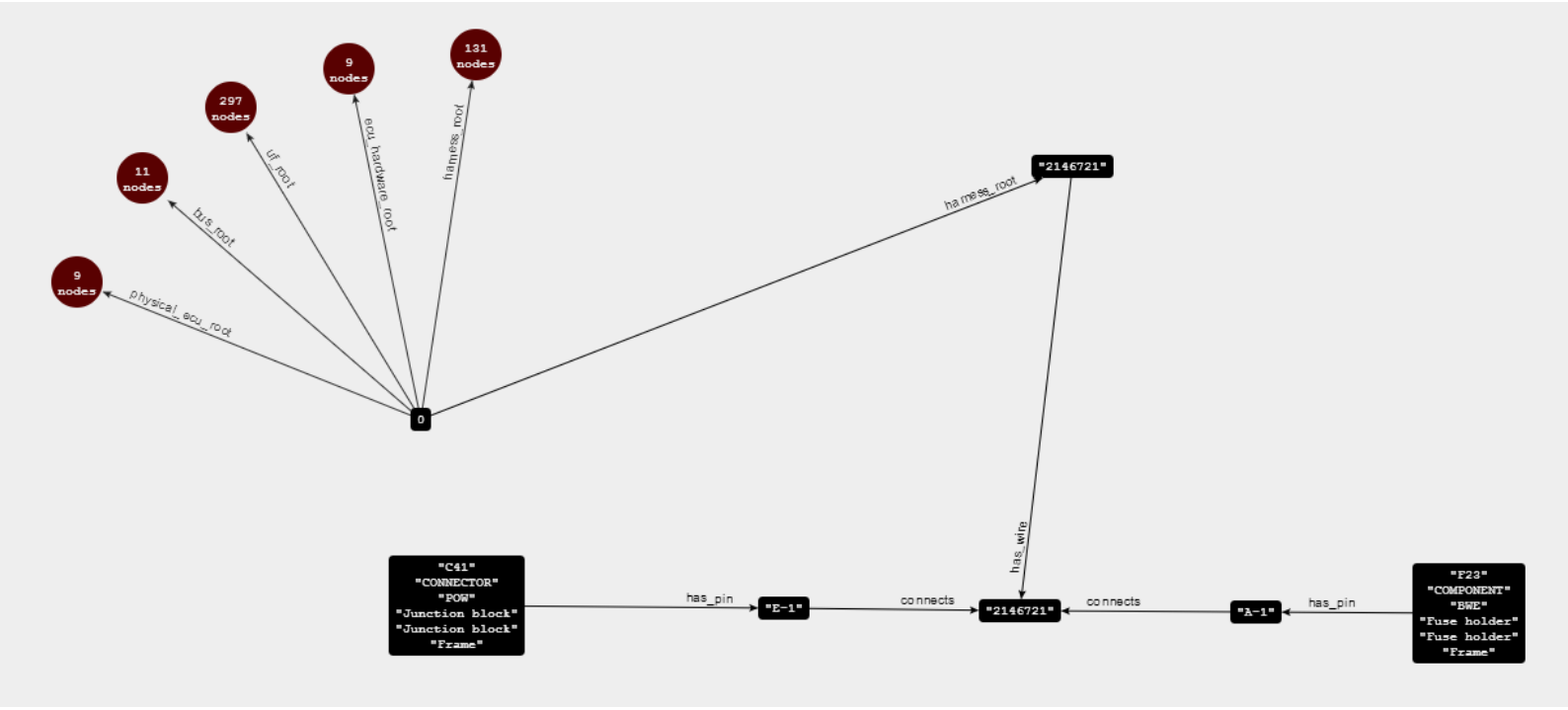

Figure 4.5: Database view of the hardware connections

Figure 4.6 displays a complex part of a truck's electrical system with many connected hardware components. In the down left side is placed the EMS hardware ECU mapped to its software counterpart. Apart from hardware ECU as a central control unit part, this case has connections towards other types of hardware components. Connectors, actuators and sensors with their descriptions from component library are highlighted among them. The functionality of the hardware components can be seen here clearly. For example, C457 is a connector of type splice which means it is used to extend the length of a cable and to multiplex signals. Another instance shows V144 component which is distinctly a fuel actuator and more specifically a solenoid valve. T125, T111 and T121 are all sensors which their main responsibility can be viewed in the Figure.

The existence of the harness is observable through the node that collects the names of all the wires. It can be noticed that the wires have the same name and connect via a specific relation to a harness node with the same name. This implementation is planned to be of use in the visualisation process and multi-view modeling.

In comparison to the previous figure, one can estimate the complexity to which this system can get and the benefits which an overall hardware and software architecture can bring.

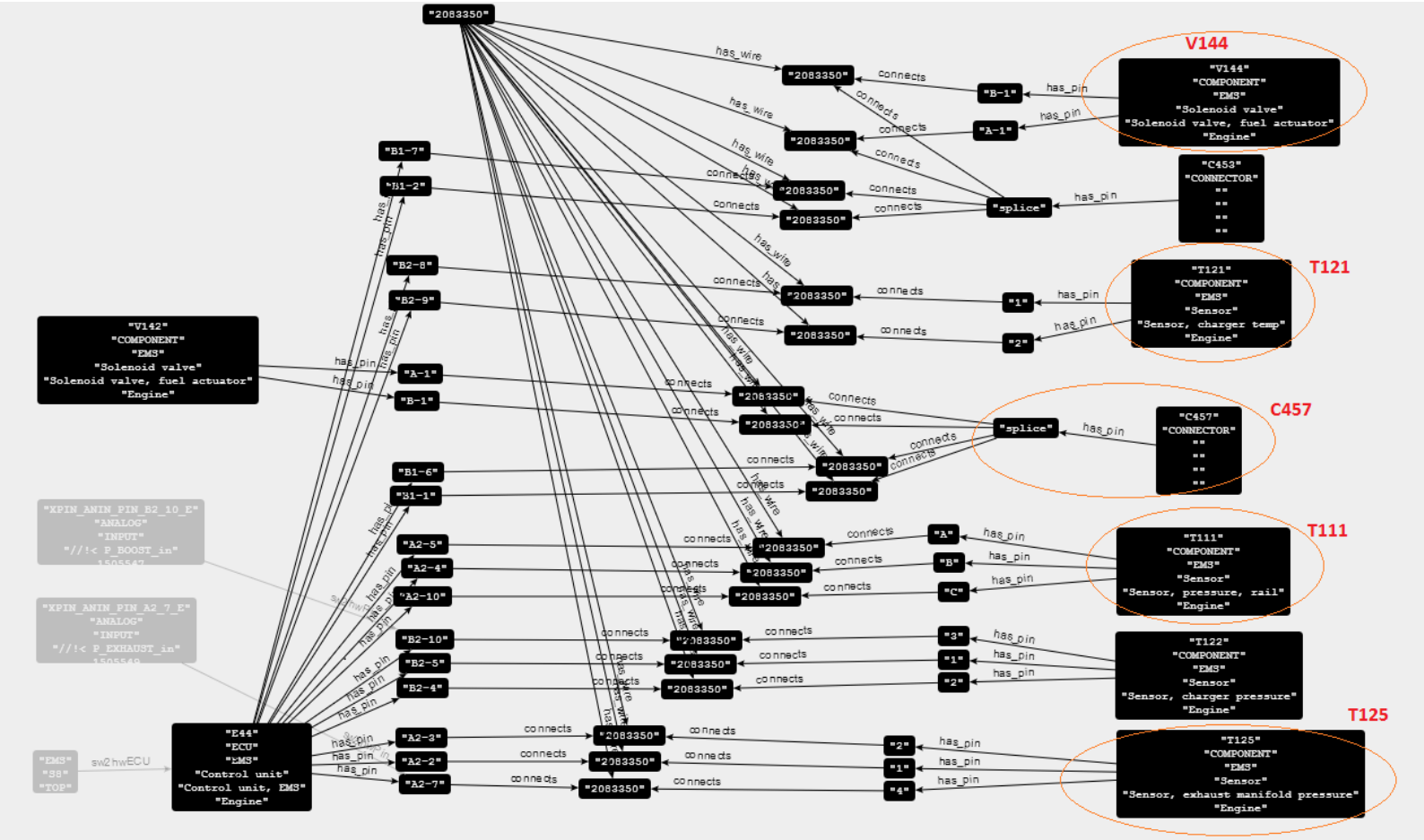

Figure 4.6: Second database view of the hardware connections

This simple example, Figure 4.7, presents a hardware component, B49, which connects through a connector, C451, to COO7. All hardware components are represented by rectangles with a gray background, except the connectors which are white. The connecting line represents the wire in the case of hardware with the pins being the filled circles. The ECU has both the pin from the cable list, the hardware pin, as well as the pin description from the source code. The name of the pin retrieved from source code has the name of the variable which was identified in the list of pins. Its name is usually standardised in the code according to specific naming conventions from where preliminary properties about the pin can be inferred.

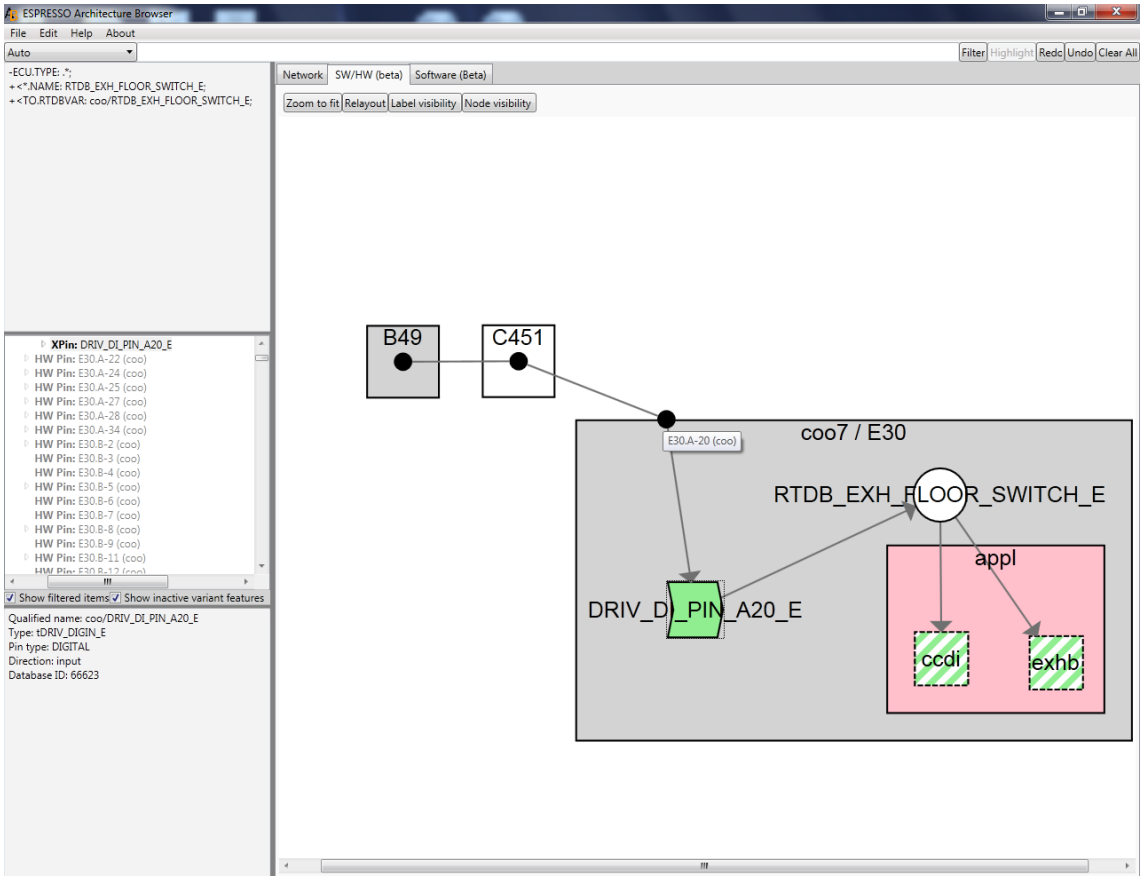

Figure 4.7: View of the hardware connections from the Architecture Browser front-end

The next Figure 4.8 presents a multitude of scenarios concerning the network of connected hardware elements. The example has a as a main hardware component the COO7 with the "appl" software layer. The layer contains application modules with functions which create, read and write to RTDBs. The global variables in turn are influencing in some way the pins retrieved from the source code. This in the end are matched to their hardware counterparts and connect to the outside of the ECU with other hardware components.

The hardware components in this scenario range from sensors and actuators to connectors and power elements. It can be observed how connectors are used to extend the connection of the elements.

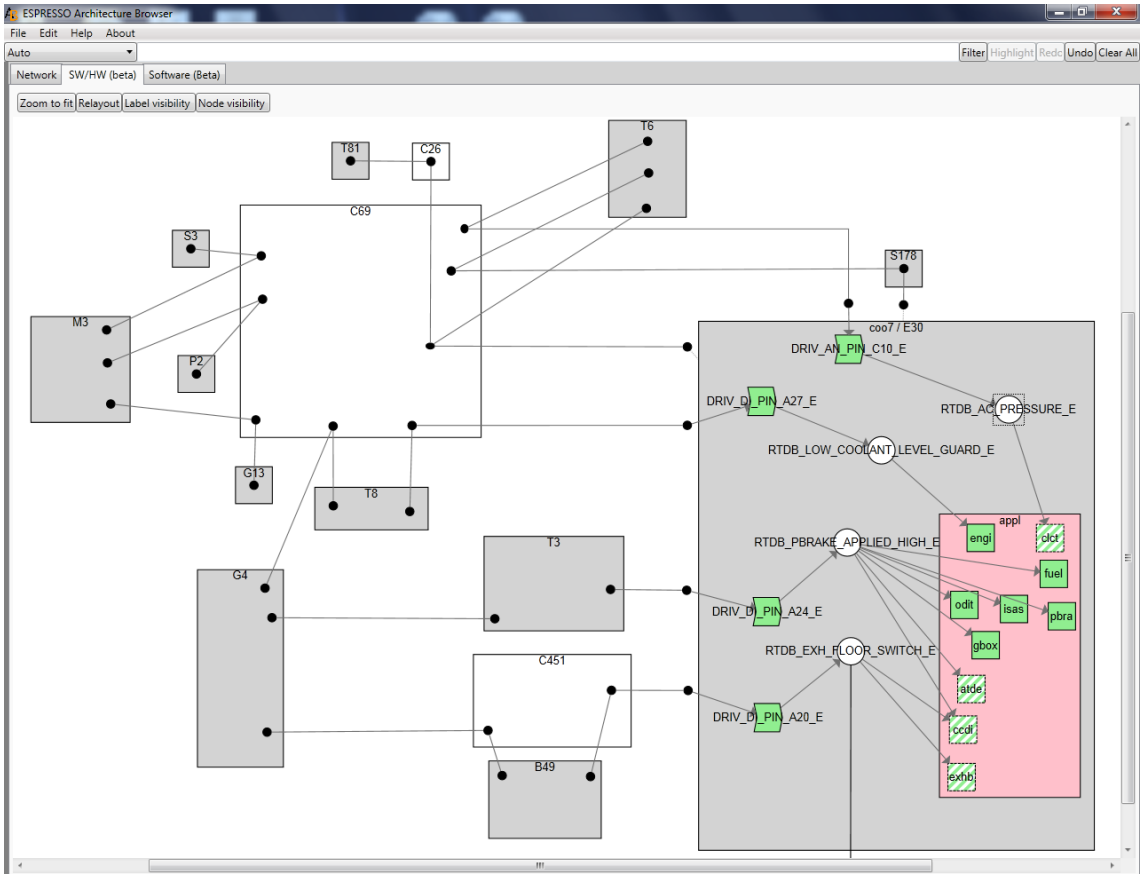

Figure 4.8: Multiple hardware connections from Architecture Browser

In order to present the results in a more general and understandable way, it was decided to analyse two particular scenarios: engine start scenario and exhaust brake scenario. This can also be viewed as an informal way of performing validation.

The two scenarios were chosen for their analysis properties and because they include the results of the thesis. For each scenario a conceptual diagram is first presented after which the actual screenshot from Architecture Browser is displayed.

In the engine start scenario, Figure 4.9, the connected hardware is:

- a switch, S4, which serves as a starter key
- a COO7 ECU which receives the information from the switch
- an EMS S8 which receives through CAN messaging a command from COO7
- the motor, M1, which receives the comand from EMS S8

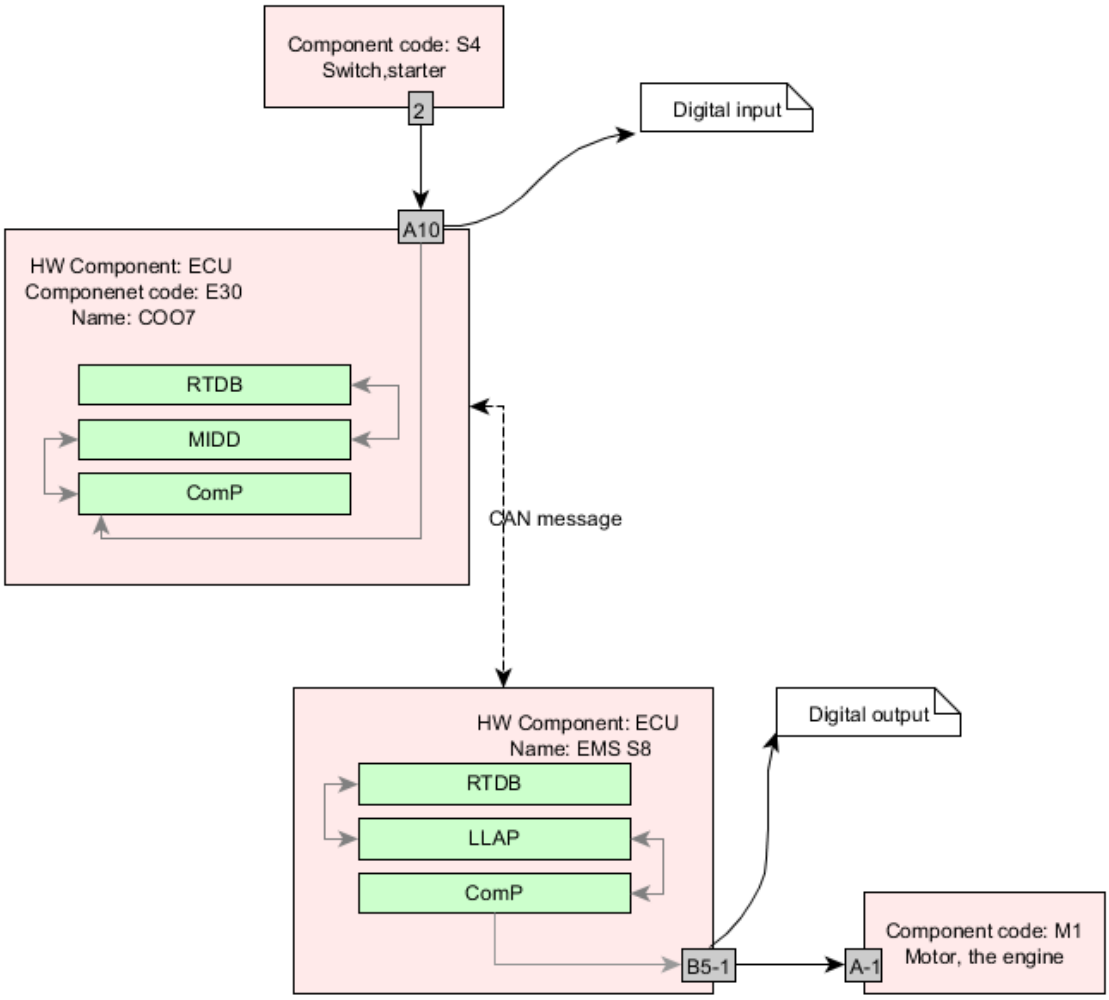

Figure 4.9: Graphical description of the engine start scenario

The result can be seen in Figure 4.10. The switch S4 is activated and the information from it is received by COO7 thought wires and pins where it is processed. From the application component that processes the signal of the starter switch, a signal is sent via CAN messaging towards the EMS S8. Here an application component deals with both processing the request and sending a command to the final actuator, the starter motor M1.

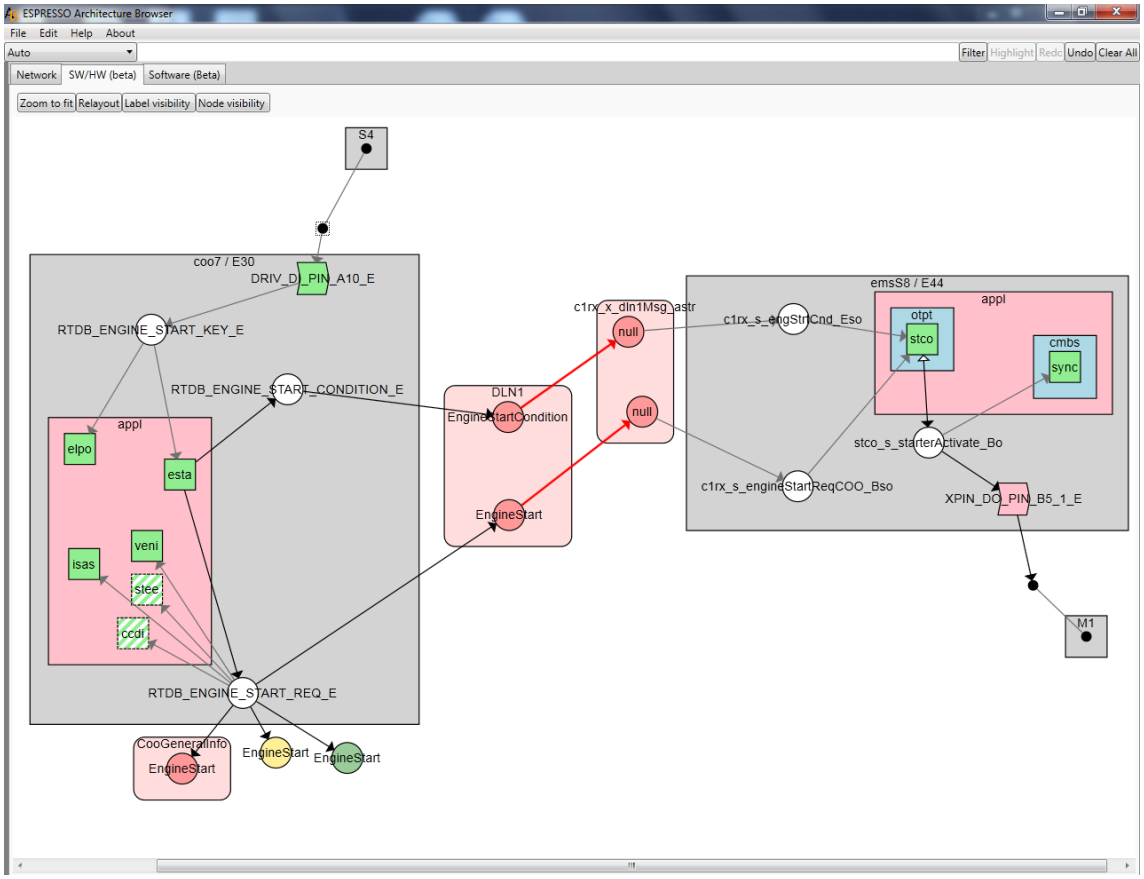

Figure 4.10: Engine start scenario from AB

In the exhaust brake scenario, Figure 4.11, the connected hardware is:

- two switches which act as inputs in regards to the exhaust brake
- a COO7 ECU which receives the information from the two switches
- an EMS S8 which receives through CAN messaging a command from COO7
- the valve which is actuated by EMS S8

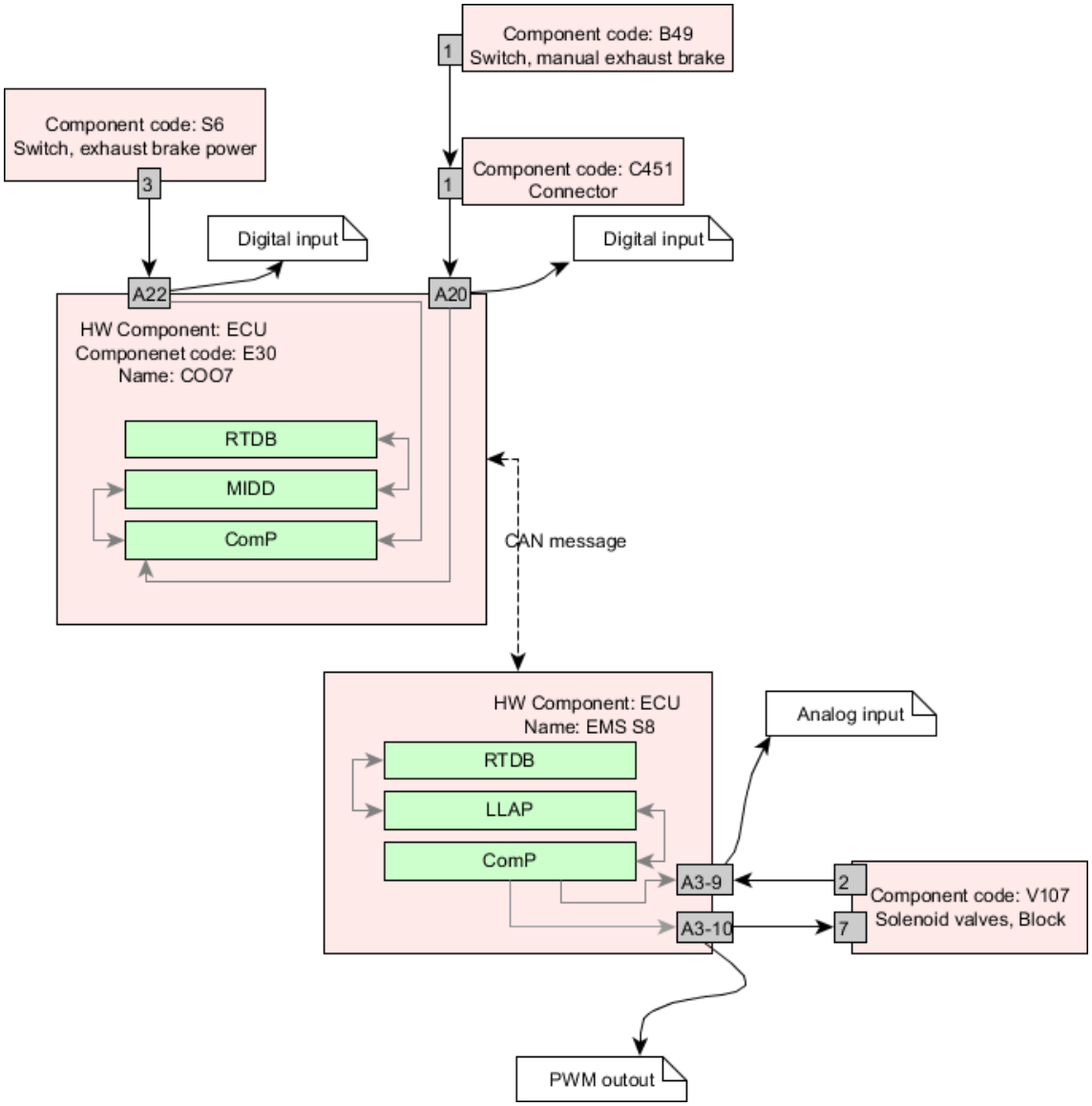

Figure 4.11: Graphical description of the exhaust brake scenario

In this case, Figure 4.12, the input for COO7 can come from multiple sources. The information from both sources is processed and a CAN massage is used as in the previous case to send a control command to the EMS. The control command is processed in the the same application component that will send the signal to the actuator. The actuator in this case is also interesting to analyse since it also seems to give feedback. Since the actuator is a valve, it can be assumed that the feedback is the sensor for position of the valve.

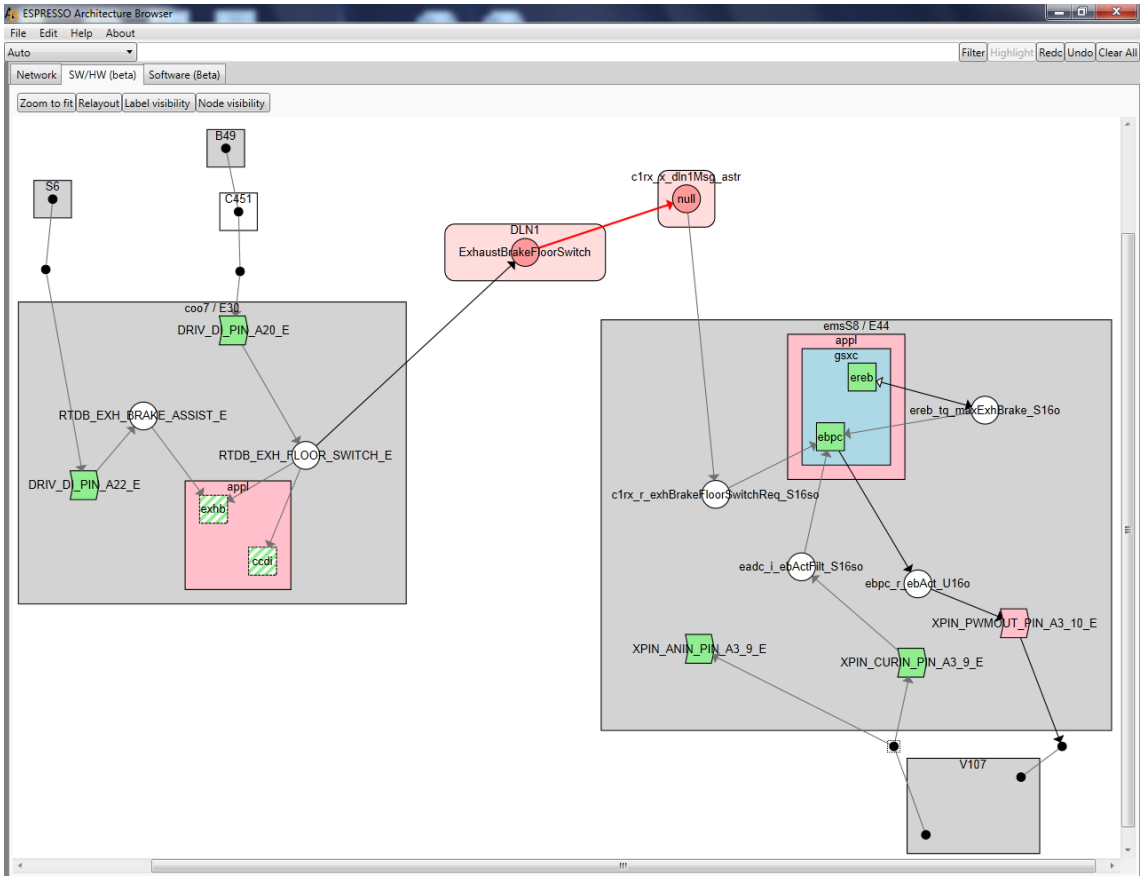

Figure 4.12: Exhaust brake scenario from AB

5. Discussion

This chapter discusses the accomplished results and argues about the techniques used.

The concept of an architecture description that can accommodate both software and hardware elements interconnected in a logical manner has been analysed and has been proved feasible. This has been achieved by collecting multiple data sources, processing them and composing a hardware model by fusing the data gathered. Although a fully automated tool could not be created, the resulting implementation suited the objectives. It needs to mentioned that both constructing the solution and using it, depended on the previous work which was and is being done in the Scania Espresso project.

The first part of the solution involved the use of the source code to extract low level software details by means of software architecture recovery. This part, that also maps to the first goal, has been accomplished to a basic level. The result was the extension of the existing meta-model and database with complementing entities called xpins. These had the role of linking global variables to external connection points. The work in this part was limited due to time constrains and a compromise was made to only recover the essential information about xpins, such as type, location, direction and description. Although it suffices the needs in this context, a more detailed level of information could have showed voltage level scaling, filtering functions and so on.

The second part of the solution had the objectives of finding data sources for hardware components external to the ECU, retrieving the data and fusing it in order to create a network of hardware components linked between them through pins and cables. This in turn maps to the second goal of creating a meta-model and implemanting based on it. A first problem encountered in this case was the difficulty of finding the right information. This was a result of the lack of communication between different departments and different interpretations of the same concepts. The hardware components retrieved described the general functionality of the elements and didn't refer to precise physical entities. One level deeper would have meant to involve variability since the technical specifications for a sensor or actuator depended on the precise truck configuration. Although some impediments were encountered, the objectives were reached and the solution was satisfactory. As an observation, the authors note that the sources used for the hardware have proved to contain only production information from NGS trucks. These are truck generations already in production, which means that they are being delivered to customers. In contrast, the new generation of trucks are called NCG. Those trucks are still under development and as such in prototype stage. If prototypes, NCG trucks, are to be described, the sources of information found in this thesis are not sufficient.

All the work was based on a meta-model which was used to both describe what information is need, how the information should be connected and how it should be stored in a database. The contribution from the authors referred to the introduction

of the hardware region in the meta-model.

The authors also noted some downsides of the meta-model. A first issue observed was the difficulty of describing conditional relations. Further on, advantages and disadvantages could be noted in using a meta-model as a template for storing architecture related data in a graph database. Although the meta-model can be directly modeled over the database, in some scenarios it seems more logical to use a relational database. For example the many different types of components like sensors and actuators together with their properties can be queried more efficiently in a relational database. This is also true whenever the goal is to retrieve statistical information from the data and not to find hierarchical connections between entities.

The benefits of having a database with the system architecture that contains both software and electrical hardware elements would not only be of interest to Scania. In terms of possible uses of this tool, it is thinkable that this could be employed as a form of architecture checking and validation in environments where model based development is already used.

The validation of the results was limited. The reason for this is mainly because the methodology was based on a case study. The Scania supervisor provided the main confirmation that the work is accepted. Besides this, the qualitative nature of the work meant that within the meetings, with contacts from different departments, some informal interviews were carried out. This interviews were not the main way of validation and there was not enough time to analyse them in detail, but they occurred with developers and engineers in order to receive limited feedback on how correct and useful they found the results. The subjects of the interviews were also included as stakeholders because of their interest in the project. Before the meetings, comparative observations were done based on the technical documentation found in Scania. Technical scenarios were extracted from the documents and compared against the database respectively the user interface.

6. Conclusion and future work

This chapter presents the conclusion of the work which has been performed in this thesis by summarising each critical point. The chapter closes with suggestions towards future work.

This project is part of the ESPRESSO initiative and as such identifies itself with the main goals, but in the same time it can be seen as a separate work.

As a separate work, the project's goals relate to the design of a model for the hardware elements in the system's architecture. This hardware model is implicitly linked to the software layers which influence it. Together with this, a database was populated based on the new HW/SW architecture. This was done on a proof of concept level.

The results of the thesis are in first place the meta-model that accommodates connections between hardware and software; the developed tool-chain for parsing source code and forgathering information from different data sources and the populated database that can be used together with AB. This shows that the goals can be accomplished, although the current work done only in a proof of concept level based on a case study. Fully validated results will provide developers with a better systems understanding which will improve analysis and debugging. Another strong point will be the traceability between HW/SW and requirements. Of course, the higher level motivation is also to investigate the safety standard ISO 26262 within this specific context.

In other words, the problems that were solved relate to the issue of outspread data sources in Scania, which sometimes have incomplete data and the lack of a database that contains the overall architecture of a truck with the connections between hardware and software.

The reason of why these are considered problems and worthy to being investigated is because the architecture of a Scania truck is very prone to being modified; it can change with each configuration and it also changes with time. Another reason is that information on these changes are spread throughout different databases in different departments depending on what each department's area of focus is. What we are trying to reason here is that although the concept of an overall system architecture exists, the components for it are separated within each department's perspective. Each department is working with a particular view of the architecture or a specific part and has, in this sense, its own data specifically related to its needs.

To return to the problem statement and research question it can be said that:

- For Question 1, the system's architecture focused on the connections between hardware and software was retrieved partially from the source code that deals with low level interaction with external elements. The best software recovery technique for accomplishing was decided to be lexical analysis and specific patterns matching. Complementary to this, other data source were used to describe the E/E hardware components like electrical schematics and so on.

This was handled in an semi-automatic way and fused with the rest of collected information.

- For Question 2 it can be said that the model was a fusion of the existing views and models. Although the model leaves space for improvements, it was demonstrated from the results that it can be used as a template of how to store information in the graph database. The model can be improved by further iterating and refining it based on the results of this work and by performing additional interviews with the engineers from the involved departments.

The points that were achieved are the following:

- gathering hardware related information from source code
- gathering information from other sources and models in Scania and processing each of them in an appropriate way
- composing a model that could describe the overall system, including hardware elements
- describing and suggesting a possible approach to graphically represent hardware entities

In order to achieve this, a meta-model was created and used to understand what information was needed, how it should be processed and how it should be persisted in a graph database so that the architecture can be provided also in a visual format to the end users. The contribution of this work was to fuse the hardware part to the software elements of the meta-model.

Suggestions for future work, that have been derived through the current work are:

- Parse all ECUs that are developed in Scania to improve the coverage of the database. It is to be noted that some ECUs are purchased from external companies, or in other cases only part of code is developed by Scania.
- Counting in variability, in future the entirety of the cable lists should be persisted in the graph database so that it can be also used for variability modeling.
- The current level of detail for hardware stops at the functionality of the hardware component: the next step is to connect the physical hardware parts with technical specification and supplier. This will on the other hand unavoidably involve variability.
- Design a method for efficient validation of the resulting hardware models.
- Completely automate the tool chain environment without any manual input.

Other aspects that are out of scope, but can be improved would be testing and simulation functionality for the architecture. Related to this, the possibility of evaluating different scheduling algorithms might also be of interest to Scania together with the possible extension of this. The last point could refer to a more

detailed study that can be conducted in regard to the data-flow in the software layers.

List of Figures

Bibliography

[1] F. Liu, A. Narayanan, and Q. Bai, "Real-time systems," 2000.

[2] I. O. for Standardization, in *ISO/FDIS 26262*, May 2011.

[3] X. Zhang, M. Persson, M. Nyberg, B. Mokhtari, A. Einarson, H. Linder, J. Westman, D. Chen, and M. Torngren, "Experience on applying software architecture recovery to automotive embedded systems," in *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on.* IEEE, 2014, pp. 379–382.

[4] S. Voget, "Safe rtp: An open source reference tool platform for the safety modeling and analysis," http://www.erts2014.org/Site/0R4UXE94/Fichier/erts2014_5A2.pdf, 2014, [Last Visited: 2014-06-01].

[5] C. Ebert and C. Jones, "Embedded software: Facts, figures, and future," *Computer*, vol. 42, no. 4, pp. 42–52, Apr. 2009. [Online]. Available: http://dx.doi.org/10.1109/MC.2009.118

[6] M. Broy, "Challenges in automotive software engineering," in *Proceedings of the 28th international conference on Software engineering.* ACM, 2006, pp. 33–42.

[7] "Scanias flexible production system to suit specific customer needs," [Last Visited: 2014-05-31]. [Online]. Available: http://www.motorindiaonline.in/commercial-vehicles/scanias-flexible-production-system-to-suit-specific-customer-needs/

[8] O. Molin, "Design verification through software architecture recovery," Master's thesis, Uppsala University, 2013.

[9] R. Hilliard, "Iso/iec/ieee 42010-2011: Systems and software engineering - architecture description," *International Organization for Standardization, Geneva, Switzerland.*

[10] J. Ulke, "Generating modelica-models of ecu hw for functional safety verification," Master's thesis, Luleå University of Technology, 2013.

[11] B. M. Juan Greco, "Network architecture recovery in the context of automotive can communication," Master's thesis, Royal Institute of Technology, 2013.

[12] A. Z. OUSSAMA CHAMMAM, "Towards automated recovery of embedded system functional architecture from source code and product data," Master's thesis, Royal Institute of Technology, 2013.

[13] “Scania in brief - scania.com,” http://scania.com/scania-group/scania-in-brief/, [Last Visited: 2014-05-31].

[14] “Scania global introduction - scania.com,” http://scania.com/scania-group/scania-in-brief/scania-global-introduction/, [Last Visited: 2014-05-31].

[15] P. O. YP, “Product development process,” Scania CV AB, Tech. Rep., 2011, scania internal document.

[16] J. W. Julius Reineck, “Gap analysis on software development between iso 26262 and scania,” Master’s thesis, UPPSALA University, 2011.

[17] M. Al Hayani, “Modeling bus load on can,” Master’s thesis, Halmstad University, School of Information Science, Computer and Electrical Engineering (IDE), 2012.

[18] “Ecu designing and testing using national instruments products,” [Last Visited: 2014-02-18]. [Online]. Available: http://www.ni.com/white-paper/3312/en/

[19] D. Holmgren and P. Madsen, “System description ems s8,” Scania, Tech. Rep., 2010, 1924977, Scania internal document.

[20] H. Persson and J. Gimholt, “Requirements specification ems s8, ecu,” Scania, Tech. Rep., 2010, 1851829, Scania internal document.

[21] F. Wilkstrom and J. Danielsson, “System description coordinator7,” Scania, Tech. Rep., 2007, 1783496, Scania internal document.

[22] P. Jansson and J. Gimholt, “Requirements specification part specification coo7,” Scania, Tech. Rep., 2008, 1728139, Scania internal document.

[23] J. Johansson, “Requirements specification ecu coo8 hw,” Scania, Tech. Rep., 2013, 2079359, Scania internal document.

[24] M. Eriksson, “System description bci1 for ncg,” Scania, Tech. Rep., 2012, 2166053, Scania internal document.

[25] J. Karlsson, “System description cms1,” Scania, Tech. Rep., 2014, 2184921, Scania internal document.

[26] H. Huang and Z. Dong, “Research on architecture and query performance based on distributed graph database neo4j,” in *Consumer Electronics, Communications and Networks (CECNet), 2013 3rd International Conference on.* IEEE, 2013, pp. 533–536.

[27] “An introduction to functional safety and iec 61508,” http://www.mtl-inst.com/images/uploads/datasheets/App_Notes/AN9025.pdf, 2002, [Last Visited: 2014-06-01].

[28] “V model,” [Last Visited : 2014-05-31]. [Online]. Available: http://www.waterfall-model.com/v-model-waterfall-model/

[29] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Addison-Wesley Professional, 2003.

[30] M. Nenad and T. R. N, "Software architecture: foundations, theory, and practice," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*. ACM, 2010, pp. 471–472.

[31] S. Ducasse and D. Pollet, "Software architecture reconstruction: A process-oriented taxonomy," *Software Engineering, IEEE Transactions on*, vol. 35, no. 4, pp. 573–591, 2009.

[32] G. Rasool and N. Asif, "Software architecture recovery." *International Journal of Computer, Information & Systems Science & Engineering*, vol. 1, no. 3, 2007.

[33] D. Ganesan, M. Lindvall, R. Cleaveland, R. Jetley, P. Jones, and Y. Zhang, "Architecture reconstruction and analysis of medical device software," in *Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on*. IEEE, 2011, pp. 194–203.

[34] P. D. Guido Di Campli, "Architectural reconstruction and its approaches," http://www.idt.mdh.se/kurser/ct3340/archives/ht08/papersRM08/9.pdf, 2008, [Last Visited: 2014-06-01].

[35] A. Vasconcelos and C. Werner, "Software architecture recovery based on dynamic analysis," in *Brazilian Symp. on Softw. Engineering*. Citeseer, 2004.

[36] K. Sartipi and N. Dezhkam, "An amalgamated dynamic and static architecture reconstruction framework to control component interactions 259," in *Reverse Engineering, 2007. WCRE 2007. 14th Working Conference on*. IEEE, 2007, pp. 259–268.

[37] P. H. Feiler and D. P. Gluch, *Model-based engineering with aadl: An introduction to the sae architecture analysis & design language*. Addison-Wesley, 2012.

[38] J.-M. Favre, "Cacophony: Metamodel-driven software architecture reconstruction," in *Reverse Engineering, 2004. Proceedings. 11th Working Conference on*. IEEE, 2004, pp. 204–213.

[39] S. Mubeen, J. Maki-Turja, M. Sjodin, and J. Carlson, "Analyzable modeling of legacy communication in component-based distributed embedded systems," in *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*. IEEE, 2011, pp. 229–238.

[40] S. Björnander, "Architecture description languages," http://www.mrtc.mdh.se/han/FoPlan/ass2-bjornander.pdf, [Last Visited: 2014-06-01].

[41] H. Blom, H. Lönn, F. Hagl, Y. Papadopoulos, M.-O. Reiser, C.-J. Sjöstedt, D.-J. Chen, F. Tagliabò, S. Torchiaro, S. Tucci *et al.*, "East-adl: An architecture description language for automotive software-intensive systems," *URL http://www. maenad. eu/public_pw/conceptpresentations/EAST-ADL_WhitePaper_M2*, vol. 1, 2013.

[42] S. D. Zilio, "Survey report on modeling languages, components technologies and validation technologies for real-time safety critical systems v 2.0," http://www.cesarproject.eu/fileadmin/user_upload/CESAR_D_SP3_R1.2_M2_v1.000_PU.pdf, 2010, [Last Visited: 2014-06-01].

[43] "Modelica and the modelica association," [Last Visited: 2014-06-01]. [Online]. Available: https://www.modelica.org/

[44] "EAST-ADL association," [Last Visited: 2014-06-01]. [Online]. Available: http://www.east-adl.info/Specification.html

[45] P. Cuenot, P. Frey, R. Johansson, H. Lönn, Y. Papadopoulos, M.-O. Reiser, A. Sandberg, D. Servat, R. T. Kolagari, M. Törngren *et al.*, "The east-adl architecture description language for automotive embedded software," in *Model-Based Engineering of Embedded Real-Time Systems*. Springer, 2011, pp. 297–307.

[46] A. Consortium *et al.*, "East-adl domain model specification," *Project Deliverable*, 2010.

[47] "AUTOSAR," [Last Visited: 2014-06-01]. [Online]. Available: http://www.autosar.org/

[48] S. Bunzel, "Autosar–the standardized software architecture," *Informatik-Spektrum*, vol. 34, no. 1, pp. 79–83, 2011.

[49] "ATESST," [Last Visited: 2014-06-01]. [Online]. Available: http://www.atesst.org/scripts/home/publigen/content/templates/show.asp?L=EN&P=111

[50] B. Baudry, F. Fleurey, R. France, and R. Reddy, "Exploring the relationship between model composition and model transformation," in *7th International Workshop on Aspect-Oriented Modeling, co-located with (MODELS'05), Montego Bay, Jamaica*, 2005.

[51] J. Bézivin, S. Bouzitouna, M. D. Del Fabro, M.-P. Gervais, F. Jouault, D. Kolovos, I. Kurtev, and R. F. Paige, "A canonical scheme for model composition," in *Model Driven Architecture–Foundations and Applications*. Springer, 2006, pp. 346–360.

[52] A. Ledeczi, P. Volgyesi, and G. Karsai, "Metamodel composition in the generic modeling environment," in *Comm. at workshop on Adaptive Object-Models and Metamodeling Techniques, Ecoop*, vol. 1, 2001.

[53] F. Fleurey, B. Baudry, R. France, and S. Ghosh, "A generic approach for automatic model composition," in *Models in software engineering*. Springer, 2008, pp. 7–15.

[54] R. France, F. Fleurey, R. Reddy, B. Baudry, and S. Ghosh, "Providing support for model composition in metamodels," in *Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International*. IEEE, 2007, pp. 253–253.

[55] R. Reddy, R. France, S. Ghosh, F. Fleurey, and B. Baudry, "Model composition-a signature-based approach," in *Aspect Oriented Modeling (AOM) Workshop*, 2005.

[56] C. Jeanneret, R. France, and B. Baudry, "A reference process for model composition," in *Proceedings of the 2008 AOSD workshop on Aspect-oriented modeling*. ACM, 2008, pp. 1–6.

[57] Á. Lédeczi, G. Nordstrom, G. Karsai, P. Volgyesi, and M. Maroti, "On metamodel composition," in *Control Applications, 2001.(CCA'01). Proceedings of the 2001 IEEE International Conference on*. IEEE, 2001, pp. 756–760.

[58] MoCAA - models composition, aspects and analysis. [Last Visited: 2014-06-01]. [Online]. Available: http://www.irisa.fr/triskell/matt/

[59] M. Emerson and J. Sztipanovits, "Techniques for metamodel composition," in *OOPSLA–6th Workshop on Domain Specific Modeling*, 2006, pp. 123–139.

[60] R. von Hanxleden, E. A. Lee, C. Motika, and H. Fuhrmann, "Multi-view modeling and pragmatics in 2020," in *Large-Scale Complex IT Systems. Development, Operation and Management*. Springer, 2012, pp. 209–223.

[61] A. Cicchetti, F. Ciccozzi, and T. Leveque, "A hybrid approach for multi-view modeling," *Electronic Communications of the EASST*, vol. 50, 2012.

[62] A. A. Shah, A. A. Kerzhner, D. Schaefer, and C. J. Paredis, "Multi-view modeling to support embedded systems engineering in sysml," in *Graph transformations and model-driven engineering*. Springer, 2010, pp. 580–601.

[63] I. Robinson, J. Webber, and E. Eifrem, *Graph databases*. " O'Reilly Media, Inc.", 2013.

[64] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins, "A comparison of a graph database and a relational database: a data provenance perspective," in *Proceedings of the 48th annual Southeast regional conference*. ACM, 2010, p. 42.

[65] "Dictionary, encyclopedia and thesaurus - the free dictionary," [Last Visited: 2014-06-01]. [Online]. Available: http://www.thefreedictionary.com/

[66] M.-O. Reiser, "Managing complex variability in automotive software product lines with subscoping and configuration links," *PhD, TU Berlin*, 2008.

[67] "Scania software development and testing," http://www.ida.liu.se/~snt/teaching/REAP/HT13/Scania.pdf, 2013, [Last Visited: 2014-06-01].

[68] C. Mengi, C. Fuß, R. Zimmermann, and I. Aktas, "Model-driven support for source code variability in automotive software engineering," in *1st MAPLE Workshop*, 2009, pp. 44–50.

[69] S. Thiel and A. Hein, "Modeling and using product line variability in automotive systems," *IEEE software*, vol. 19, no. 4, pp. 66–72, 2002.

[70] H. F. Cervone, "Understanding agile project management methods using scrum," *OCLC Systems & Services*, vol. 27, no. 1, pp. 18–22, 2011.