Model Development and Data Provision for Formal Analysis of Cause Effect Chains

Tejas Pravin Phadnis

Advisors

Prof. Dr. Klaus Schneider
Mr. Max Jonas Friese [Daimler AG]
Mr. Hannes Walz [Daimler AG]

Department of Commercial Vehicle Technology
and
Department of Computer Science

This project work is submitted in fulfillment for the degree of Master of Science

April 12, 2019

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this project work are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This project work is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and in acknowledgements.

Tejas Pravin Phadnis
Kaiserslautern, April 12, 2019

Acknowledgement

I would first like to thank my project work supervisor Prof.Dr. Klaus Schneider who was always present whenever I ran into some trouble or had a crazy thought. He consistently steered me in the right direction whenever he thought I needed it and at the same time provided tremendous academic support.

I would like to express my profound gratitude my enthusiastic mentors Mr.Max Jonas Friese and Mr.Hannes Walz. I am particularly indebted to them for their constant faith on my work. I thank them wholeheartedly, not only for their support, but also for giving me so many wonderful opportunities. They contributed in all possible ways in the projectwork and made sure I am on the right path.

I would also like to acknowledge University of Kaiserlautern for providing me with the opportunity for writing the project work and Daimler AG for providing me with the computation power.

Finally, but by no means least, thanks go to mom and dad for almost unbelievable support. They are the most important people in my world and I dedicate this project work to them.

Abstract

A cyber-physical system, electrified vehicles are the most emerging fields for the research and development of the automobile because of high efficiency and the zero emission. The sensors and actuators have an interaction with the physical environment in the embedded system. Therefore, they must execute at a pace determined by their environment. The performance analysis of the real-time embedded system has more significance in the designing process of the complex system and the electric and electronic (E/E) architecture. This analysis provides an overall picture of the system behavior in all situations before the start of actual implementation.

The main purpose of the project work is to create a model which is used for performance analysis and verification of the powertrain. The tool is created to visualize this model and collect all the timing attributes of hardware and software topologies. These topologies are focused on different layers of communication which includes various components such as software modules, runnables, bus, ECUs, and have different timing attributes. The path of the signal transmission from the sensor to the actuator in the cause-effect chain can be visualized in the time-augmented model. The connection between the time-augmented topological model and the formal analysis can be created by the database model.

In future, this tool will be used to find out the worst case scenario of the cause-effect chains.

Contents

List of Figures

List of Tables

Listings

Nomenclature

AUTOSAR	AUTomotive Open System ARchitecture
CPS	Cyber-Physical System
ECU	Electronic Controlled Unit
GUI	Graphical User Interface
IDE	Integrated Development Environment
OSI	Open System Interconnection
RTE	Run Time Execution
SQL	Structured Query Language
UML	Unified Modeling Language
XML	Extensible Markup Language

1 Introduction

1.1 Drive for the development

Cyber-Physical systems are systems that integrate, control and monitor the physical environment with software systems. An automobile can be treated as a cyber-physical system which has been the drivers of many concurrent innovations for several years. This is realized through distributed embedded systems, involving several controlled units, sensors and communication channels to improve safety and security of automobiles. These control units require precise monitoring and signal processing. Therefore, it is necessary to accurately model a real time-triggered scheme to implement a deterministic model of the system, emphasizing a new importance on electric and electronic (E/E) architecture.

1.2 Background of project work

Presently, automotive systems have complex embedded systems with complex architecture of control units and actuators. These embedded system have to interact with the physical environment and communicate with other control units. The performance is a key quality attribute of such a complex embedded system. It is very important to identify the reaction time between the signal initialization and actuation of actuator. In a cyber-physical system, consider one system-function example as gear shifting or braking. In the process instant, a signal is initiated, transformed and processed by many different control units. Signal follows various paths to reach the actuator and after a certain interval of time the actuator gets actuated. This is referred as the cause-effect chain [figure 1.1]. Considering safety and driving experience, the reaction time for such a cause-effect chain is critical. Therefore, it is necessary to estimate this reaction time for the worst case situation.

Major challenge in the embedded system of powertrain is to minimize reaction time and finding out the bounding of worst case reaction time. The performance of the powertrain depends on reaction time. Therefore, it is very important to predict the reaction time before start of the development and its implementation. This prediction can be achieved with the help of formal methods. Formal methods are the

techniques used for design and verification of software and hardware systems.

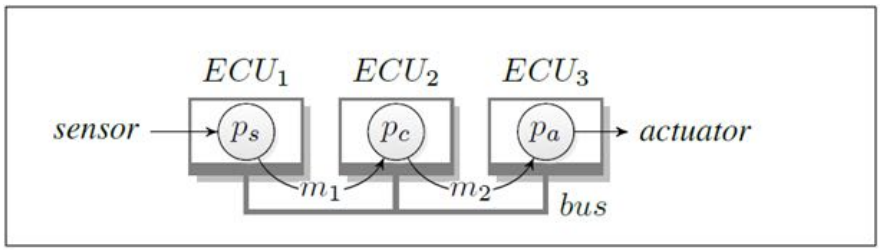

Figure 1.1: Chain of action
[LSS+12]

1.3 Purpose of the project work

The embedded structure of electric and hybrid vehicle is complex and it will further increase in near future. Therefore, the performance analysis of such architecture is very difficult; because it includes large number of input events, resource sharing, interconnections among shared units, scheduling of hardware and software control modules. For such a system it is required to create an abstract model for performance analysis. The time verification, formal analysis are also possible with the help of abstract model. But this predicting time behavior tasks always prove to be expensive in terms of computational resources. To deal with such a problem, the tool support is needed. In such tool, a software developer can work on hardware and software topologies together, understand the flow of signal, identification of resources, and perform the analysis which finally supports important decisions of cause-effect chain. The development of this tool will be discussed later.

The main aim of the project work is to support the development in terms of conceptual and implementation by means of formal methods focused on the performance analysis of the cause-effect chain. The main problems “a system integrator" deals with are as follows:

1. Which data must be collected for a continuous, formal description of the effect chain?
2. What exchange formats are available and how can information that changes with different frequencies (hardware vs. software development) be brought together in a consistent way?
3. How can system model and formally described cause effect chain be brought together? At which points is a ‘cut’ of the analysis of the chain possible?

To reduce complexity, we divide the chain into multiple parts which can be ana-

lyzed seperately. The results of the individiual analyses need to be aggregated in a following way.

The purpose of the project work is to develop a concept and a tool which brings automotive hardware and software topologies together. In research and development department, currently, these two topologies are developing independently. Therefore their worst-case scenarios, effect chains are studied separately. From the mechanical point of view there are many software systems like Ansys, HyperWorks, 3DS MAX which can be used for analysis and visualization of car model from different perspectives like air and fluid dynamics, bending force analysis etc. Hence, there is also a need to develop a tool to visualize hardware and software topologies of a car under one roof. With the help of that, developers can understand the flow of signals and latency time of software task as well as the time required through hardware components. So it is useful to analyse the worst case scenario of the cause-effect chain.

Furthermore, the basic requirement of a formal method is to consider parameters which affect the time of a signal. The hardware topology is based on the various communication bus systems. So the project work includes a detailed study of communication protocol stacks of each bus such as LIN, CAN, Ethernet, FlexRay, etc. Determination of time relevant attributes and their values of each bus and other components like ECU, connectors helped in the formal description of the cause-effect chain.

1.4 Tool Overview

The hardware and software topologies of automotive are described with the help of XML (Extensible Markup Language) file.The XML file which represents the software topology has information about ECUs, tasks, runnable entities and various types of signals. Similarly hardware topology described in XML file called as ARXML (Autosar XML). It represents communication cluster, busses, controllers, transport protocols, and their time relevant attributes and values. The code is written in C++/CLI to develop a tool which reads software XML file and depending on the selected signal and/or runnable it forms a chain of signal processing. Similar concept is used to read ARXML file. It reads complete communication cluster of a bus, frames, PDUs, resolve the references for controllers, and collect all time relevant attributes and its value of each node and child nodes. As per the internal flow structure of Software and Autosar XML file, the database structure is modelled and connection is established with developing tool. All time relevant attributes and their values are stored in database. When a powertrain engineer wants to analyze communication stack for latency timing of frames, PDUs, buses or ECUs then it will be helpful to read these values directly from database. This also be helpful to find largest path of cause effect

chain and worst-case scenario. The complete overview of the tool is as described in figure 1.2.

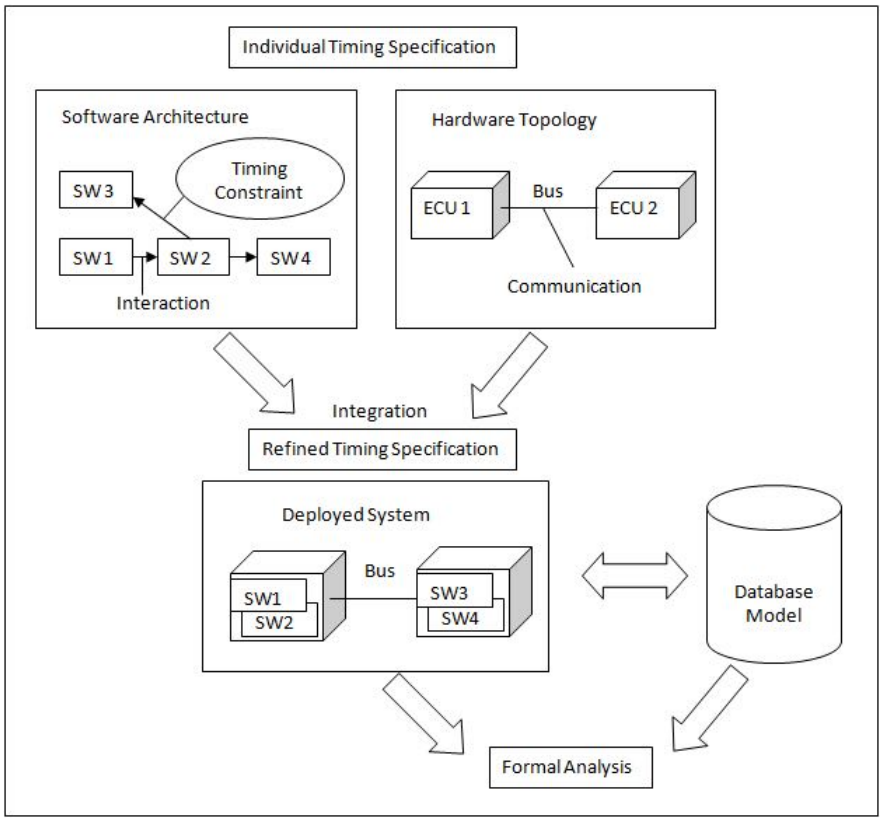

Figure 1.2: Tool overview

2 State of art

2.1 Powertrain

2.1.1 Introduction of powertrain

Basically, the powertrain is a set of components such as energy source, transmission, differentials, etc. that generates power and deliver it to the wheel and surface. In conventional powertrain, the engine is used as an energy source which requires more components than an electric motor and battery used in an electric vehicle. In the case of the hybrid powertrain, both energy sources can be used. The powertrain is the heart of a car and is the identity for high performance and efficiency of a vehicle. Moreover, other important features of a car like top speed, drivability, towing capacity, fun to drive also depend on the gradeability of a powertrain. On the other hand, society imposes environmental demand on the manufacturer to reduce the emission and noise of a vehicle. This factor was the turning point to start the electrification of a powertrain [HHGS18].

2.1.2 Electrification of powertrain

The electrification of the powertrain is started first with hybridization with the help of an electric motor. In recent years, the environmental impact of pollutants has received considerable attention from all levels of populations throughout the world. Maximum CO2 is emitted from the automobiles. These emissions must be drastically reduced to save the environment. The environmental benefits of electric and hybrid vehicles are the main endorsed for electrification of the drivetrain [LPWT15].

There is a worldwide necessity that air pollution has to be reduced, which encourages the electrification of the powertrain in all vehicular applications. Electrification of a powertrain means controlling mechanical parts with the help of electronic and/or software else replace them completely by electric part. For example, the IC engine is replaced by an electric motor/generator and battery. Loss of kinetic energy during braking, heat energy generated due to friction, waste exhaust gas energy can be recovered with the help of electric motor which acts either as an energy supplier or an energy recover component when runs in generator mode. The paradigm shift

of powertrains from conventional to electrification has paved the way for innovations. Driver assistance, Partial automation, high automation can be possible with the help of electrification. Electric powertrains are inherently more efficient and emission-free and it meets the customer demands in the best possible way. The new attraction in electric powertrain is 'just push a button and enjoy the speed in silence'.

2.2 Cyber physical system

Cyber-Physical System (CPS) is a branch of engineering which deals with interaction of cyber i.e. computational with physical systems. In other words, it is a mechanism which is controlled by the computer based algorithm and interacting with all possible ways.

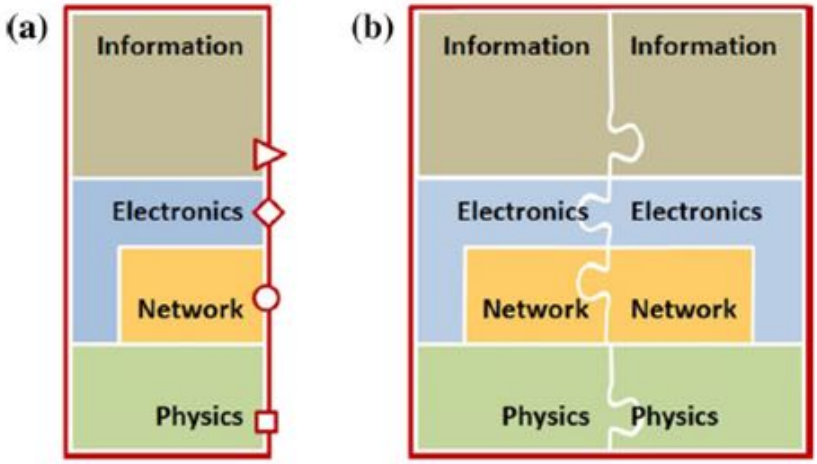

Figure 2.1: (a) Networked embedded system (b) Cyber physical system [SJ18]

Earlier, electrical, mechanical and software subsystems were treated as a standalone but now they are considered as a single system as CPS. Consider the figure A.1 as an example of an electric vehicle. It is a 'system of systems' interacting with all heterogeneous systems. In such cases, the controller represent the cyber world and the human driver represents the physical system. This CPS involves embedded system, control theory, software engineering, etc. and have hardware and software real-time requirements. Now-a-days, most of the systems are cyber-physical system like aerospace, manufacturing, transportation, etc. To improve the control and performance of the system it is very important to have a highly coupled and interactive

system. Mobility, efficiency, functionality, reliability and safety can be achieved with the help of CPS. The main advantage of the CPS is that it forms a linkage between embedded processing and network topology with the physical system. So that all perspective of the system can be analyzed and a high level of automation can be achieved. Electronic Controlled Unit (ECU) plays an important role in this system. They control the actuation of the actuators by interconnecting in the system and precise signal processing.

2.3 Complexity of network

Evolution of an automobile from the mechanical system into an electromechanical system had many challenges. In beginning, in-vehicle communication can happen with the help of sensors which senses all physical and environmental parameters and convert into the electric signal. Further, these signals are transformed to the controller for processing; as a result, the actuator gets actuated. The movement of the actuator and their environmental values are again measured by the sensor and this cycle continues further which creates a closed loop with feedback. In its native form, controllers are connected to each other with the help of wires. As we go further into this evolution of technology, the functionality of vehicle increased by increasing the number of sensors and control units, which results into an increasing number of wires and the complexity of the network. Pushing further, communication can be done between controlled units and connection between them can be established with the help of automotive communication buses, which reduced the number of wires on one hand side but increases the complexity of signal processing on other hands [Bos14].

2.4 Automotive architecture

2.4.1 Embedded architecture

Automotive Electronic Control Units (ECUs) are the most dominant software controlled electronic device which overcomes the main limitations of a mechanical system like limited accuracy and efficiency. A modern car contains more than 70-100 ECUs which executes gigabytes of code. 50 to 70 % of the development regarding ECU is related to software. The functionality of all the parts are controlled by the ECUs. An increasing number of ECUs inside the vehicle accompanied by the increasing complexity of the system. ECU reads the data from the sensor, and operate within certain value to ensure the functionality of the vehicle. With the help of ECU, it has interaction with the rest of the system. ECUs have different working

requirements based on priority and bandwidth. Therefore in-vehicle communication, buses can be used differently as per the applications. Electric and electronic (E/E) architecture is a way to handle the problem of increasing complexity in a decent manner [HMG11].

2.4.2 E/E architecture

The automobile is leading to become more software and electronic intensive system. The term electrical and electronic (E/E) architecture is used in automotive technology in contemplation to implement an electrical system and their modules such as processing units like ECUs, buses, sensors or actuators and their functions. In the era of an intelligent system having functionalities like driver assistance, brake assistance, skidding control or parking aid, etc. In deliberation of incorporating these new functionalities into the car, its functional components need to be connected to ECUs and sensors have to be placed in a relevant position in the hardware. The E/E architecture is dealing with the electric and electronic network of the vehicle, their interfaces and communication between them with software architecture.

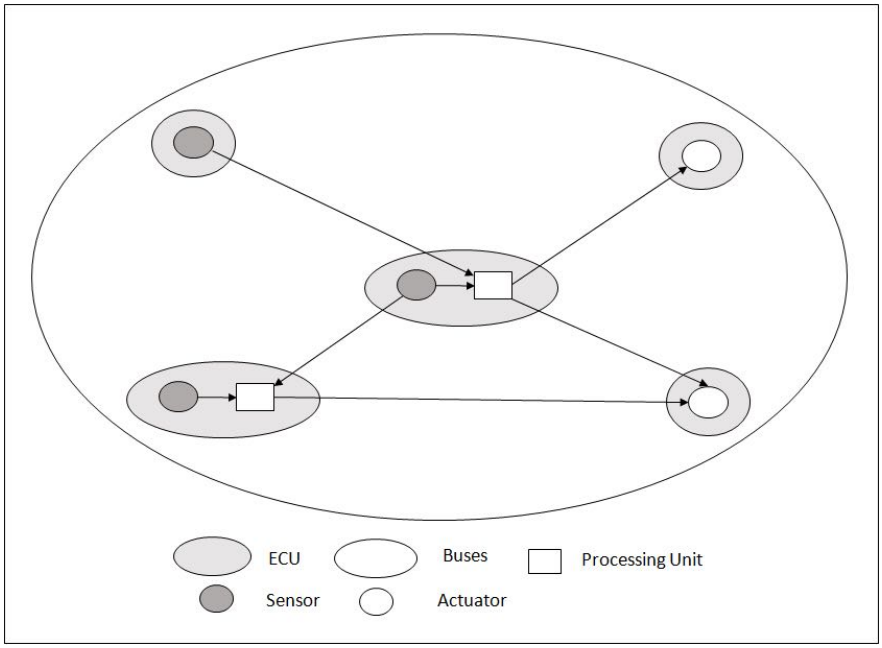

Figure 2.2: E/E architecture design example

These components are communicating with each other via signals. So as per the application suitable bus is selected for data communication and it is important to make a connection with internal topology. Consider an architectural design example given in figure 2.2; E/E incorporates the optimum position of electric compo-

nents and ensures the signal and data distribution of all components. It involves transmitting actuator and sensors values to the control units via communication buses. In other words, E/E architecture is not only handling the complexity of the network but also optimizing the system. It plays a significant role in improving the efficiency of an electrified powertrain. Based on the connections and data transmission path in network architecture, the system is becoming intelligent to take some decisions regarding operating parameters such as the speed of the vehicle, optimum battery charge, engine ON/OFF, and maximum torque. As a result, the E/E system and architecture reduce emissions and increases the vehicle range and efficiency. Therefore, the importance of E/E architecture increases at an astonishing pace in electric and hybrid vehicles [KCG+17].

2.5 Types of ECUs in powertrain

For the design of an automotive embedded architecture and design communication between parts, the vehicle electronic system is divided into four functional parts as Chassis, Powertrain or Drivetrain, Body, and Telematics. The sensors and ECUs are also grouped into these categories. These 4 categories have many different sensors and ECUs which placed at different locations inside the vehicle and have different purposes (refer figure 2.3). As per their functionality, the abbreviation of ECU varies.

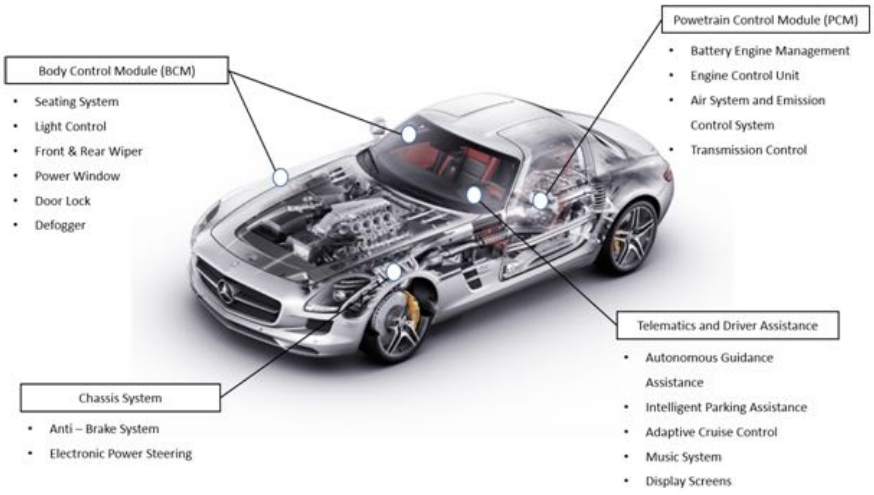

Figure 2.3: Automotive ECUs
[MB]

The features of a car can be achieved by sharing a signal and data through many sensors and control unit connections. For different functionality requires different ECUs. There is an interconnection between control units and these control units

are sharing the network resources in one or other way. Some common examples of control units with their functionality are given in table 2.1 and their names differ as per the OEMs convention.

Abbreviation	Control Unit	Main Function
ECM	Engine Control Module	Responsible for controlling engine performance
EBCU	Electronic Brake Control Unit	Controls braking of a car
PCU	Powertrain Control Unit	Responsible for controlling powertrain performance
TCM	Transmission Control Module	Responsible for gear transmission
VCU	Vehicle Control Unit	Responsible for driver assistance
BCM	Body Control Module	Responsible for suspension, temperature control

Table 2.1: Different types of control units and their functionalities

Earlier only one functionality is incorporated in an ECU. But now many functions are assigned to the single ECU. The different sensor data is gathered at the ECUs. These ECUs uses sensor data for further processing of a signal. And output data of ECU can be used by other ECUs, or actuators.

ECUs are controlling all functions of the automotive like opening and closing of air intake valve till adjusting the side mirror, from the battery management system till front and rear wipers. Overview of cause effect chain and inside part of a control unit is shown in figure 2.4. The main part of the control unit is a microcontroller (EPROM). The function code is stored in this part. ECUs gathered all relevant data and signals. The sensor sends analogous signals which get converted into digital form in ADC (Analog to Digital Converter) module. This data acts as an input for the controller functional code or processor which decides the behaviour of the unit. Within a microsecond, it performs millions of calculations on this data and generates the output which decides the best result for the further process. Therefore, one can say that, though the body of a car is made up of metal, but the brain and its heart are processors and control units that regulate the entire system.

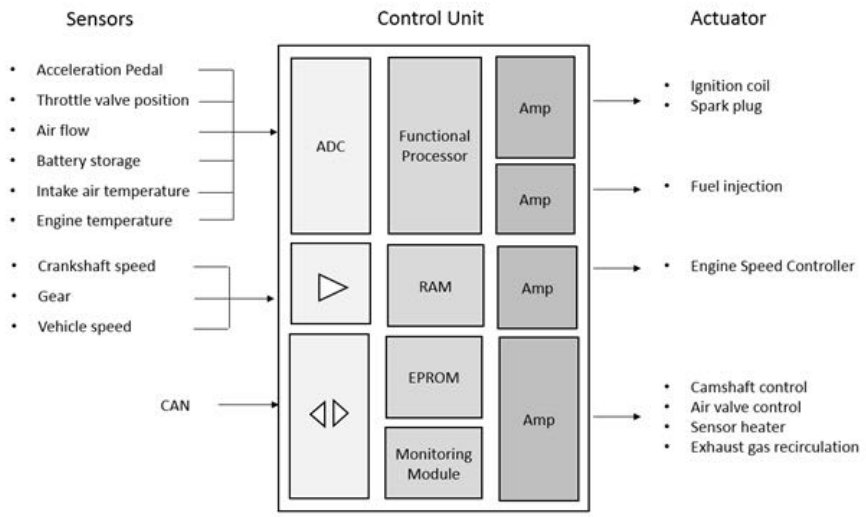

Figure 2.4: Functional module of an electronic system

2.6 Network Communication

Many systems in a car are dependent on each other, many times they used the same data as an input. Or the output of one module could be the input of others. E.g. if the driver gives the command to apply a break. So at the time, the same signal need to be given to the engine management unit to reduce the engine speed by reducing fuel intake and air mass flow. Meanwhile, the same signal has to be sent to gear module to reduce the gear and if it is a hybrid vehicle then motor needs to operate in generator mode to recover energy and store it in the battery. In this way, all are dependent on each other in order to increase performance, efficiency and safety. To this end, the system is networked with each other. For this purpose, they need to communicate with each other in an efficient way and shared the data. The required secure, accurate and fast communication between all resources can be achieved via automotive communication buses (e.g. LIN, CAN).

2.7 Bus Topology

In topological methodology, ECUs, sensors are called nodes and their connection via wire or bus is called as the transportation medium. A network is a system in which a group of nodes can exchange information via a means of transport. The network topology is the structure of nodes which represent how nodes are interconnected with each other. Every node subscriber must have at least one connection to

the other network node. Then only communication is possible within the network. Many varieties of network topology are possible by connecting different nodes and using various communication buses. It will help to create optimize network and selection of suitable bus for communication is possible.

In bus network topology, communication buses such as LIN, CAN, and Ethernet are used to connect nodes. The data transmission is taking place via messages. A message contains useful data (called payload) and data transfer information. Two types of communication can be possible; one is via subscriber oriented method. In this method, to transmit data, each node must have to subscribe to a channel and the connected bus. The transportation information is attached to the message. It is used to recognize the recipient and transfer the information to the correct destination. Another method is Message-Oriented-Method; each message is accompanied by the address. This message is sent to all the receiver nodes and the receiver decides to process this message or not. Because of these two methods, the latency, time triggering events, periodic functions to trigger the data comes into the picture, which are the relevant considerations in the cause-effect chain [Bos14]

2.8 Automotive Communication Buses and Network Protocols

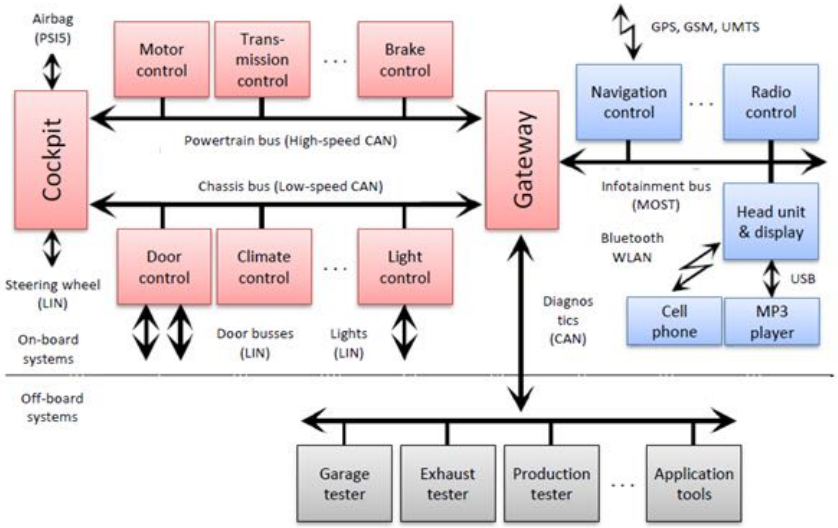

Figure 2.5: Vehicle architecture example

Each bus has a different communication method which is referred as data transfer protocols. These protocols have accompanied by a different layer called an OSI

model (Open System Interconnection Model). Each layer represents the different information of the communication data. The different application has different constraints of communication like safety and priority. On other hands, all buses (like LIN, CAN, FlexRay, Ethernet, MOST) have different properties like data transfer rate, interference immunity, real-time capability, length of data transfer capacity, etc. Therefore different buses and protocols are used for different applications. This is explained in the table 2.2 in more detail. Moreover, the maximum number of nodes are interconnected to each other by a bus in the different part of the car (refer figure 2.5).

	Class A	Class B	Class C	Class D
Transfer Rate	Low data rate (up to 10 kb/s)	Medium data rate (up to 125 kb/s)	High date rate (up to 1 Mb/s)	Very high date rate (up to or >10 Mb/s)
Purpose	Actuator sensor control	Information sharing	Real time control	Real time control and Networking in telematics
Applications	Body domain i.e. comfort functions, wiper, light, door, seats, mirrors, climate control	Transfer information as vehicle speed, emission, mechanism for error handling	Powertrain control of engine, chassis, suspension	Control of steering, braking, multimedia functions
Representative	LIN, TTP/A	Low speed CAN	High speed CAN	FlexRay, MOST
Priority Level	Low (for non-real time)	Low (for non-real time)	High (real time)	High (real time)

Table 2.2: Comparison of functional classes of communication protocols

3 Helping tools

The project work contains variety of tasks from software development and electronic point of view such as reading and visualization of XML file, creating UML (Unified Modeling Language) diagram, version control to develop a software, and database management using SQL (Structured Query Language).

3.1 XML Notepad

XML Notepad is the Microsoft XML editor used for the visualization of large XML files. This editor is very intuitive allowed to present data in both tree and text view (refer figure 3.1 and 3.2). Therefore, it is very user-friendly to visualize all main/-top nodes, their sub-nodes, child, and their attributes, etc. It has very good performance measure on validation of XML schema. It also has incremental navigation search which is helpful to navigate through all relevant nodes till their last child nodes. So finding out the path of the signal processing is easy. The XML schema also supports the XPath concept which is used to give the reference to the other nodes.

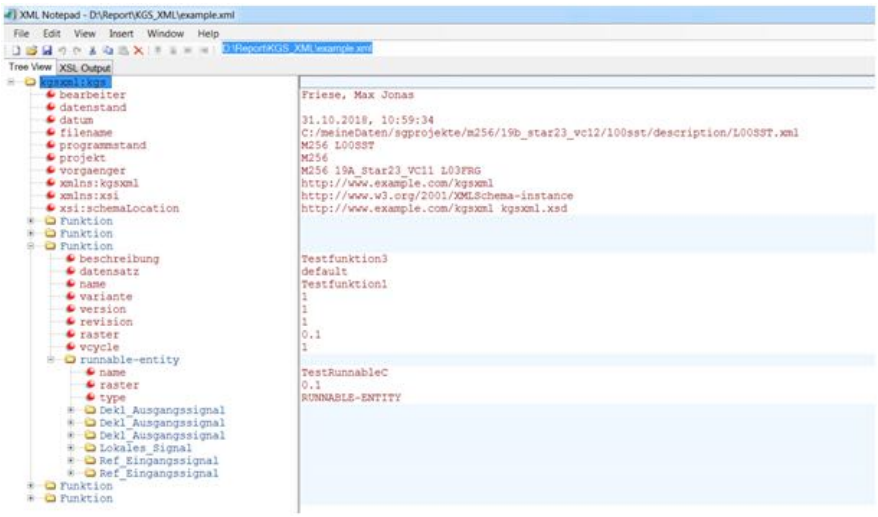

Figure 3.1: Tree view of the XML file

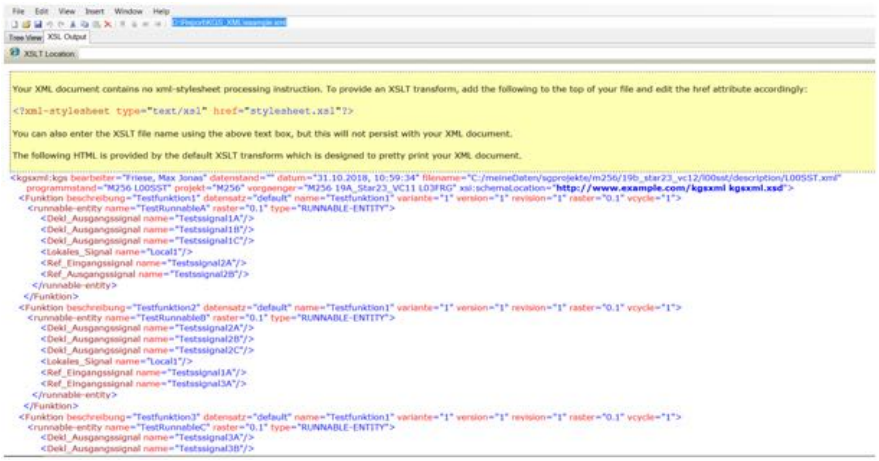

Figure 3.2: Text view of the XML file

3.2 Microsoft Visual Studio

It is an IDE (Integrated Development Environment) from Microsoft. This environment is used to develop a program, software and Apps. It works on the windows platform. It supports many programming languages such as C, C++, C++/CLI, .NET, XML, C#, Visual Basics. This visual studio includes code editor along with debugger. Moreover, it provides a compiler, code completion tool, graphical designer, etc. It also supports GUI applications and database connection. The latest version 2017 of Microsoft Visual Studio is used in the development of tool during the project work.

3.3 Git

Git is one of the version control systems. It creates one repository on the server that keeps the track of changes made in the code over time. This system is important in the software industry when many developers are working on the same project. In such cases, the individual developer is writing different functions of code. So each time for further development, it would require for everyone to access the latest version of source code to save changes. This global repository is accessible from any geographical location within the respective industry network. The continuous improvement in the project can be possible by Git. Each time after doing any changes in code, it is necessary to push those changes in the Git repository. So it will store all versions of the source code. It is possible to access any version of a source file at any point in time. Push, pull, add, commit are some examples of Git command to reflect the changes into the repository.

3.4 ARXML Visualizer

ARXML visualizer is the software used to view the Autosar XML file. This tool is used to load and identify cluster and other nodes within the ECU architecture represented by ARXML. This software is similar to XML notepad tool. But the only difference is editing, the creation of any node points, saving is not allowed.

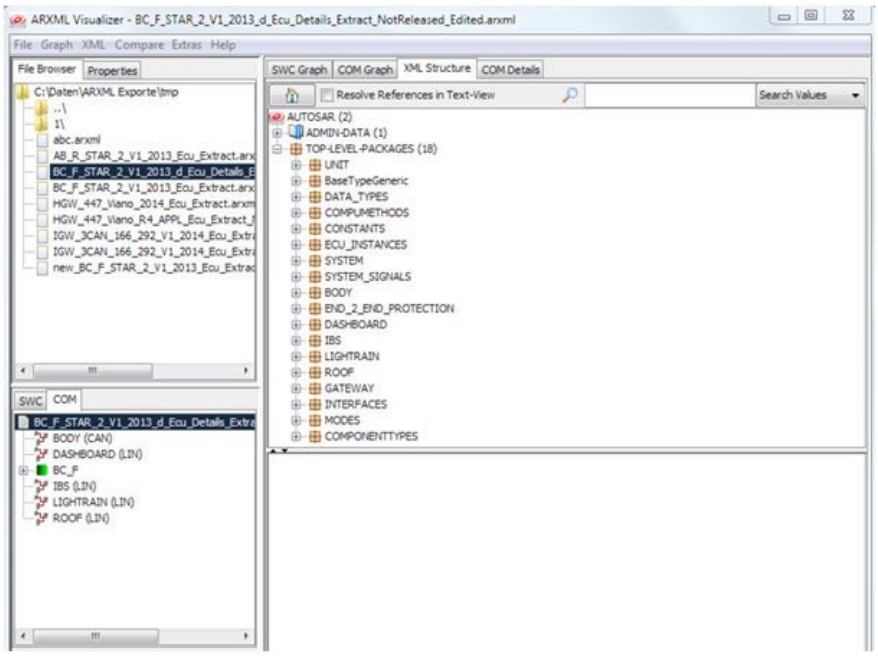

Figure 3.3: ARXML Visualizer

This tool is used to visualize complete electronic architecture of automobile. Software and hardware connections can be viewed with this tool (Refer figure 3.3).

3.5 SQL Server Management Studio

Microsoft SQL database server is a database management system used to storing and/or retrieving the data as requested by the other software. The database can be generated on the same machine or the different machine across the network. SQL server management studio is used to managing the data of the database server. This includes script editor and result window. Data is stored in the form of a table. Attributes of data entries are stored as the table column. It is a very user-friendly tool to view and analyze entire data storage and connections between them.

4 Methodology

4.1 Software Development Concepts

Now the world is shifting more towards technology, as a result software becoming heart of the all development process. Software development is the process of developing software through progressive steps in specific manner with certain objectives. The process contains not only writing a source code but also includes the preparation and analysis of requirements, system design, documentation and testing whether it has met the objective or not. The phases of software development are 1.Identification of customer requirements, 2.Analysis of software requirements, 3.Feasibility study 4.Detailed specification of software requirements, 5.Software design, 6.Programming, 7.Testing, 8.Integration and 9.Maintenance. Software development can be done by various methodologies such as waterfall model, iterative development, agile process, etc. In this project work, the software development is mainly focused on agile process [Lap17].

4.2 Agile Method

Agile is the methodology of software development based on the incremental and iterative development where requirements and solutions evolved in each step. In this agile method, the planning is incremental and design and delivery are iterative.

4.2.1 Implementation of Agile in project

Incremental delivery is the key feature of the agile method. The project work has 3 major sessions, which are independent from each other in terms of requirement, understanding of input data, and coding environment. Therefore, as per agile development, the complete software development of project work is divided into 3 iterative parts. In first step, the software topology is taken into consideration. The entire design cycle is carried out on this topology and executable tool is generated. In second step, the software development for hardware topology is taken into consideration. In this case, again executable code is written to read AUTOSAR file and

time relevant data are identified in communication cluster. This code is integrated with the first part. In third section, the data based model is created with top-down approach and complete data is stored in database. In such way all three sections are bring together to form an entire executable software which is used for performance analysis of cause effect chain in accordance with real time embedded system of powertrain. The overall agile process is shown in figure 4.1.

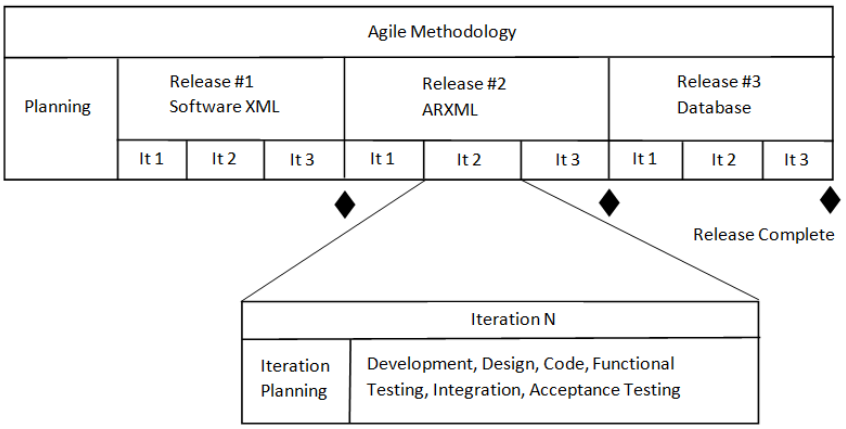

Figure 4.1: Agile methodology : software release

Each iteration is accompanied with specific functionality of the specific software part and after each release, the size of tool is gradually increased. These releases represents the checkpoints in the project and also used to decide the timeline of the project work. Therefore the first step is to collection of data; e.g. which information exchange formats are available and analysis of current existing tools in market. Pros and cons of these tools is studied and need to our tool is discussed with development in further sessions.

5 Data Exchange Formats

5.1 Industry Standards

Data exchange is the process of transforming one schema or structure of the data into another structure without changing the meaning of the source data. In our case, the source data is the architecture of distributed embedded system. So it is very important to represent all control units, signals, their connections in a very accurate manner in one file or in a software. Therefore, data exchange formats and data exchange languages plays an important role in the project work.

In the automotive industry, software is developed in a distributed manner. On one hand side, the functionalities inside the car are increases, on other hand side complexity of software architecture also increases. The end automotive software is growing day by day. Therefore, OEM divided software development into subparts and given it to the suppliers for the development. At the end, OEM has to integrate all these parts. To make this emerging process manageable, OEM has to follow some method and format for data, architecture and their interface representation. So that OEMs can harmonize exchange of the information between the suppliers and carried out the development in a secured manner. There are many data exchange formats and modelling formats are available which are described in further sections.

5.2 Available exchange formats

There are many data exchange formats are available and are used for different applications. Some examples are shown in table 5.1. XML is found to be the most versatile and universally compatible method of representation. Moreover, it is open and extensible method. It is more secure than any other formats of data transfer. Therefore, many organizations follow XML [for].

File Extension	Name of the format	Description of format
CSV	Values separated by comma	In this document, data is represented in a tabular format where columns are represented by commas.
JSON	Notation od JavaScript objects	Lightweight data exchange format, JavaScript is used for the programming, easy for machines to interpret data.
RDF-XML	Infrastructure of Description of Resources	Used for modeling of a web resources in the form of objects. It is used in representation of data in web pages.
API	Application Programming Interfaces	API are runs through HTTP protocols. This format is used for machine-to-machine interaction.
XLS	Microsoft Office Excel	It has cells in rows and columns structure. Each cell represents a data. It is used to represent informatory structure but does not follow any tree structure.
XML	eXtensible Markup Language	It has a tree structures to represent data, all properties/attributes of the data can be include in this method, easy for representing complex data structure, representation of interconnections and linkages within the data are also possible

Table 5.1: Data Exchange Formats

5.3 Proprietary format

It is the file format specific to the company or organization which stored the data according to particular encoding-decoding scheme. This scheme is designed by the company and only the employees who are working on that file can decode the meaning of the data. This is done to achieved high level safety secretes. The interpretation of stored data in such file is only possible with the help of particular hardware and software developed internally by that company. In the working de-

partment of project work, XML file format has been used as a proprietary data storage format.

The data model approach of any hardware and software system has main 4 problems such as identification, data retrieval, storage and exchange. These problems are effectively handled by the XML format which has elevated its importance. Using xml it is possible to exchange the data between incompatible systems. Many data structures are complex, therefore converting the data into xml format greatly reduces the complexity. Xml also creates a data which can be read and used by many different types of applications. XML allows to structure the data into standardized schema. Therefore it is easier for the combined development of hardware and software. It also creates the platform independent way of storing data. Therefore, it becomes widely accepted 'self-describing' open standard for data exchange format [XML].

5.4 Available exchange formats for data model

Data models are defined as how the data is stored in a particular format and structured. For modelling of complex data, formats like AUTOSAR, AMALTHEA can be used. The main big problems in the automotive industry are to handle parallelism exploitation of the embedded system. This problem can be solved by the AUTOSAR and AMALTHEA. These both methods also provide suitable processes for various applications.

5.4.1 Amalthea

AMALTHEA is one of the open source toolchain platform and/or environment. The main goal of this platform is to do data exchange in a scalable, modular and comprehensive way, which is also a platform independent. It enables the creation and modelling of a complex chain including simulation and validation. Amalthea represents the software and hardware models. The software model is more detailed than the AUTOSAR Timex model. It also supports chains of large model size, their mapping, and code generation. In this, the mapping section is used to join hardware and software parts.

In software model part, it defines the functional behavior in terms of runnable and tasks. Runnable elements are the software unit that defines the behavior of communication. Labels define the shared data and runnable are mapped to the tasks. Interrupts are also taken into consideration during runtime. The hardware model describes the system which contains ECUs, control units, microcontrollers, etc. Moreover, in the constraint module, event and event chain based constraint are taken

into consideration. For example, order, synchronization, repetition, age constraints, etc. With reference to figure 5.1, the consideration of all parts can create the system model [SSH+16].

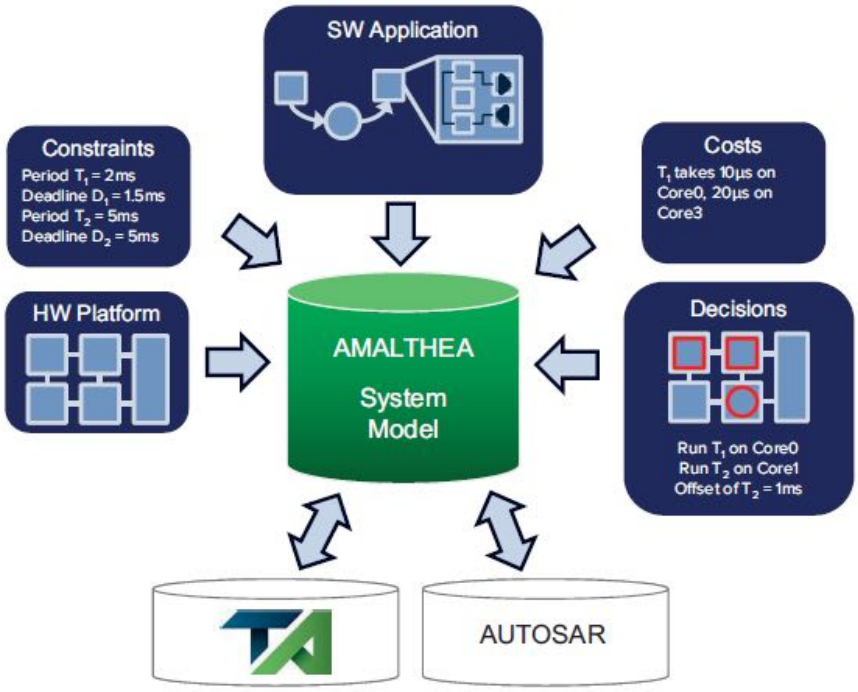

Figure 5.1: Amalthea system model [SSH+16]

Overall, the advantages of Amalthea system model are consideration of static and dynamic software architecture with runtime functions, event chain consideration based on timing requirements, software design constraints for execution and whole mapping. But the drawback of the Amalthea system model has a limitation in consideration of hardware topology. In the hardware model, it covers only a single ECU. It is not considering cores, memories and peripherals. Furthermore, it ignores the data network i.e. the communication between ECUs via buses. Therefore, neglecting buses means neglecting physical channels, ports, their signals, latencies and many more time-dependent modules. This leads to the only partly correct abstract model of the entire system. These drawbacks are the call for the new development in data exchange format.

5.4.2 AUTOSAR Timex

In automotive development, the subsystems are divided as ECUs, software, and communication buses. The different development is done at different suppliers. But the overall system design and integration are done at the car manufacturer i.e. at the end of OEMs. The structure of the subsystems, as well as the timing requirements, are known as time model, are modelled using TIMEX. AUTOSAR Timing Extension (TIMEX) depends on the AUTOSAR standard. It provides predefined events and event chains. Whereas event is the predefined behavior of the system and event chain represents stimulus and responses. With the help of TIMEX, it is possible to do the analysis of timing behavior and validation of the system with respect to time constraints. The timing properties of PDUs, frames, software tasks are taken from the AUTOSAR files. It also includes functional bus timing (VfbTiming), software component timing (SwcTiming), system timing, electronic control unit timing (ECUTiming), etc. and all are brought together in TIMEX. Therefore this is another purpose of the TIMEX to support temporal behavior by providing sufficient timing information. The overview of the TIMEX is shown in figure 5.2.

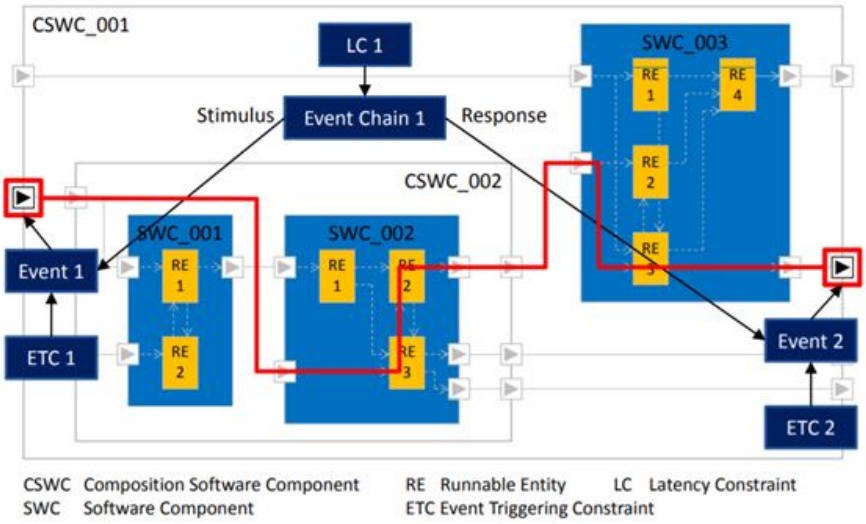

Figure 5.2: AUTOSAR TIMEX
[Con]

AUTOSAR TIMEX depends on the top-down development process i.e. timing requirements are considered at the system development level. But in actual world applications, time attributes and their guarantees are relevant in the implementation and integration phase of different subsystems i.e. it needs a bottom-up approach. Therefore, this exchange format is not completely suitable for model-based development of the effect chain [AUT].

Therefore, the limitations of AMALTHEA and AUTOSAR TIMEX highlights the need

for a sophisticated framework for exchange formats. By taking into consideration the need for formal verification by using the abstract model of the entire system and overcoming the limitations of currently available formats, the new framework need to develop. The sections 5.4.1 and 5.4.2 also represents the first step in the software development process as a requirement analysis which contains identification of customer's requirements and feasibility study.

6 Software Topology - Tool Release #1

6.1 Concept of software topology

ECU has many functionalities as shown in figure 2.4 and it has to execute those functionalities during runtime. The functional behavior of the ECU which is defined by the software component. The software components contain the piece of code wrapped in a unit called as runnable shown in figure 6.1. One software module has many runnables and are stored at the application layer in the architecture (refer figure A.2). The mapping of the instances of the runnables, Operating system, and/or sensor-actuator hardware system is undergone during ECU configuration and it followed during Run Time Execution (RTE).

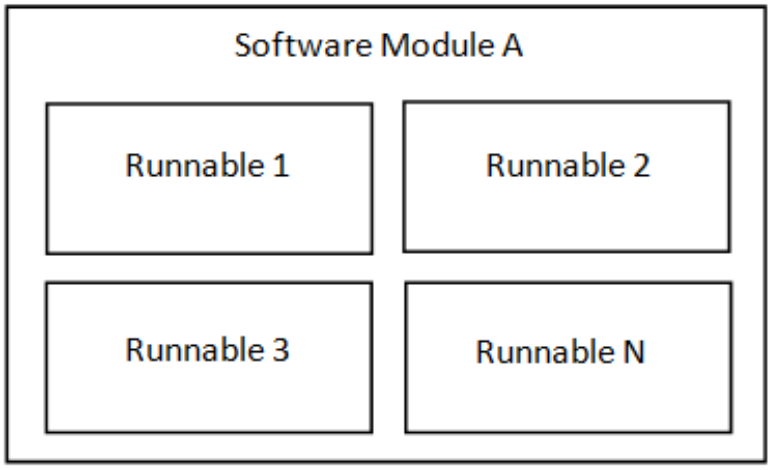

Figure 6.1: Software module

Therefore, during run time execution, the runnable entities are assembled into the tasks and these tasks are assigned to the ECU. And the system has many ECUs. Each runnable has an individual execution semantics. Depending on the implementations, runnable are divided into 2 categories; periodically running and spontaneously running. Periodic runnables have a definite executable time period and

they run after a certain interval of time. The execution of spontaneous runnables are depending on signal processing. Depending on the time required for an execution, an instance of the runnables are group together into the tasks and have a definite schedule in the multicore system. At RTE, according to the semantics, when any task is activated, it receives the data then it executes the operation and returns the data. The overview of the ECUs containing runnables and connected over the network are shown in figure 6.2.

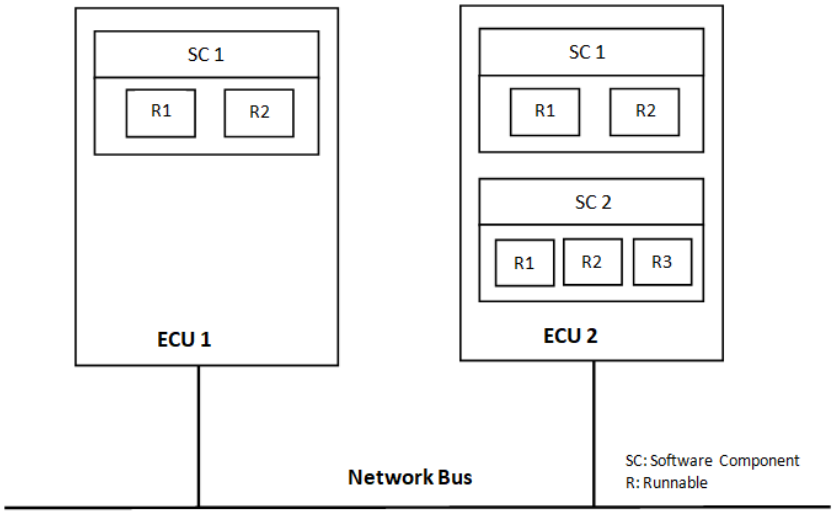

Figure 6.2: Runnables incorporated in ECUs

6.2 Challenges

The software topology is extracted in an XML file with reference to the advantages mentioned in section 5.3. The hierarchical structure of the XML schema represents the tasks (called as a function), runnables and their local, input and output signals in terms of nodes, sub-nodes and childs. The signal is coming from the network i.e. sensors and is transmitted through various ECUs. When it arrives at the ECU, different tasks are getting activated inside an ECU as per the schedule and time triggered activities. The signal is transmitting through the runnable in the activated task. There are many input signals and many output signals for a runnable. The output of the one signal can act as an input for one or many runnable. Similarly, the signal gets transmitted and form the chain of runnable. Many times signal changes during transmission. So the output signal of one runnable may or may not be the same as the input signal.

The tasks, runnables and signal transmission are explained in figure 6.3. The first challenge is to identify the properties or description of a signal and path during transmission. Secondly, it is important to find out which runnable is using which signal and finally how it is propagating and forming a chain. To solve this problem it is important to develop a tool in which one can visualize the signal processing in a runnable chain. This development is discussed in the next sections. Another challenge is with an XML file. Each XML file is focused on one ECU. Therefore, the number of ECUs used in powertrain is equal to the number of XML files need to be considered in the development of the tool. It is important to handle this problem in a data management system which will be discussed in further sections.

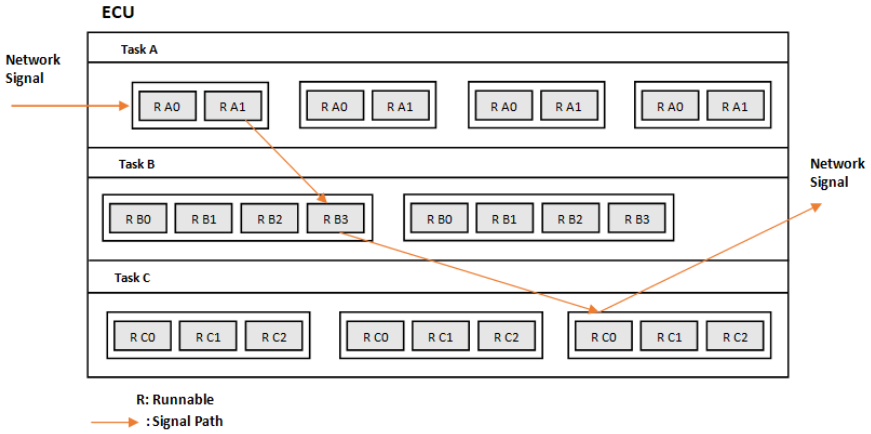

Figure 6.3: Tasks and runnables in ECUs

7 Development of tool for SW-XML

7.1 Analysis of SW-XML

The XML file represents the structure of software topology of one of the ECU. XML has particular schema and syntax to represent the interconnections. The sections are represented in the start and end Tag. For example, <section> called as start-tag and </section> indicates the end-tag. As per the explanation of the previous section, tasks contains runnables and runnables contains signals. This structure is shown in figure 7.1 of an XML file.

Figure 7.1: Text view of the XML file

In this case, tasks are represented by the <Funktion> which is also called as the node and <runnable-entity> are the sub-nodes. Similarly, output, local and input signals are known as child nodes, represented as <Dekl_Ausgangssignal>, <Lokales_SIgnal>, <Ref_Eingangssignal> respectively. The 'beschreibung', 'datensatz', 'name', 'variante', 'version', 'vcycle', 'type', etc. are called as the properties or attributes of the respected elements. These properties represent the identification of the elements,

data carried by them and its own value. By considering the confidentiality and security, the file data elements are modified by keeping the schema and purpose the same. Referring to figure 7.1, the runnable with name 'TestRunnableB' has 'Testssignal2A' as an output signal which act as an input of 'TestRunnableD'. Similarly, its output signal is the input for 'TestRunnableC'. In such a way the chain of runnable continues. Therefore, in such a case, the main aim is to write a code to read identifiable attributes of the respective entities. Secondly, read all signals with distinguishable properties and relevant data. Stored these value and types in the database. Then identify the linkage of the runnables based on the signals and display the formation of the chain.

7.2 Programming environment

7.2.1 Purpose of programming

XML visualization is very difficult for the analysis of big data files. Therefore, there is a need to create a user-friendly tool which converts XML data into the easily understandable form and developed a GUI (Graphical User Interface) where the linkages between runnables can be visualized and chain between them can be formed. The Microsoft Visual Studio version 2013 and 2017 are used for the GUI application. Its advantages are discussed in section 3.2. It provides entire support to create windows application features such as buttons, textbox, empty box, scroll bars, tables, etc.

7.2.2 Modeling in object-oriented programming

The C++/CLI is the best programming language to migrate from native C++ into the .NET platform. It has many advantages of using the features of this language. It can also preferably interoperate with others. It is a unique language to build code for user interface and powerful language to read the data from the XML file. Therefore, this language can act as a bridge between the worlds of applications. The other features like 'for each', 'list' are used for reading the large schema. One of the dominant features of the C++/CLI is the allocation of managed objects on the unmanaged heap is possible with '*' and 'new'. Furthermore, semantical allocation of unmanaged objects on the managed heap can be achieved by '^' and gcnew. This language gives the incomparable possibilities to develop a tool and handled the big data files. This language and environment also provide a wider scope to deploy the application on a secure database, for example, SQL server manager.

7.3 Automotive software development concepts

7.3.1 UML Diagram

A UML (Unified Modeling Language) diagram is the virtual representation of the system. It includes main actors, roles, actions, and classes which are used for the better documentation, understanding, maintaining and improvements of the system. UML is not a standalone programming language like C++ or JAVA. It is used to create the blueprint or pseudo programming of the system. Therefore, when input data and requirements are large then it is required to design UML diagrams. It will create an overview of the system such as various parts in the system, interconnections between the data, inheritance, dependencies, etc. All aspects of the system will not be possible to implement in one UML diagram. Therefore, UML encompasses into different categories which include different diagrams and that can be used for different purposes throughout the development [Tut]. UML divided into 2 broad categories such as Structural and Behavioral UML and their subcategories are shown in the figure below. The class diagram and use case diagrams are used in the development of project work and are explained in section 7.3.2 and 7.3.1 respectively.

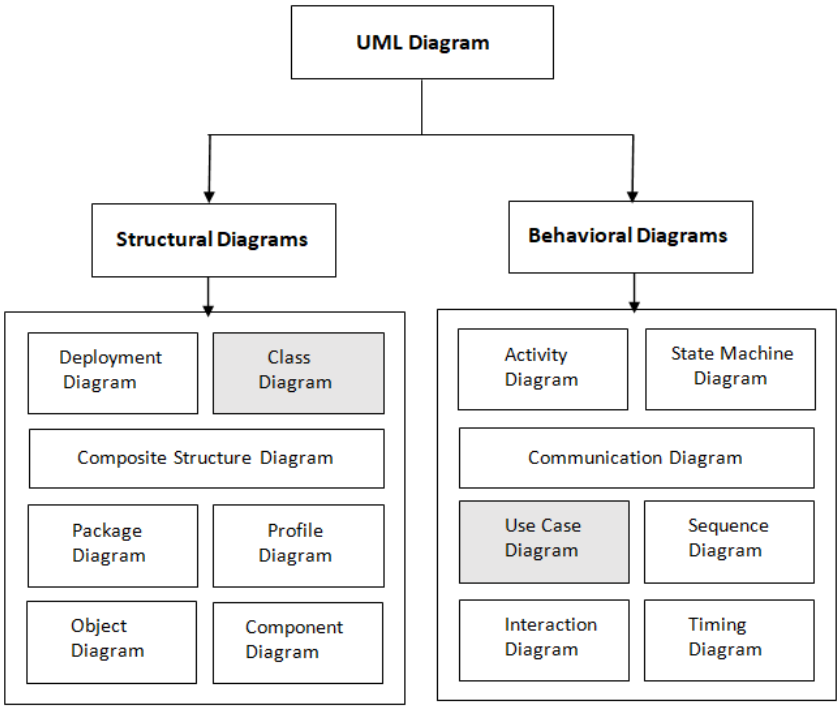

Figure 7.2: UML class types

7.3.2 Class Diagram

A class diagram represents the static view of the application. The functions or objects which have similar role or purpose are groups together into a class. Different classes together can form a system. So the class diagram helped for the better understanding of the schematics of the application. It states information about the interconnections, data type, inheritance, associations, etc. The concepts of class diagram helped in project work to understand the structure of XML. Therefore, the XML tree data (nodes, childs, attributes) is converted to the different classes and interactions between them can be identified. The class diagram is drawn by following the standard procedure and syntax of the UML in the open source platform named as 'DIA' which described in figure 7.3.

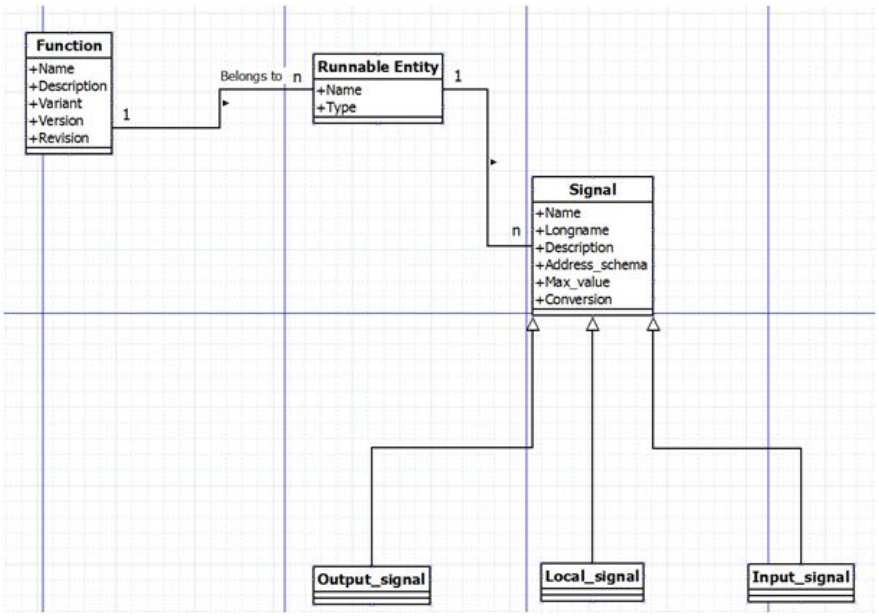

Figure 7.3: Class diagram

The relation between tasks, runnables and signals are explained in previous sections. The same relation can be explained in a class diagram in a sophisticated manner with all the details. The rectangular blocks are known as the classes and the lines or arrows connecting to them implied the relationship. The tasks have a direct relation to the runnable, known as simple association with 1:n relationship which is denoted by a solid line in the figure. It also means that one task has many runnables. Similarly, one runnable have multiple signal association (1:n relation). In this case, the signal is considered as a broad category of the class called a parent class. The signals internally have an 'is-a' relationship with 3 more subclasses such as Output_signal, Local_signal, Input_signal. These 3 subclasses are inherited

by the parent signal class which are shown by a solid line with a hollow arrowhead pointing from child to the parent class.

The rectangular blocks of the class have 3 portions. The first portion is the class name. The second portion is the class attributes. These attributes defined the data members of the class in the code. The third portion is the class operations. They are the services or the functions provided by the class (not shown in the diagram for the confidential purpose). Therefore, the class diagrams help to start the implementation of the coding. It provides the background data required for the declaration of classes which are the main building blocks of the object-oriented programming. The number of rectangular blocks in this diagram is equal to the number of classes declared in the code. It also states that the signal class can act as an abstract class where the attributes of this class can be shared by the subclasses inherited by it.

7.3.3 Modularization

Modularization is one of the method of programming in which the dividing a program into separate subprograms called as a module. As per the desired functionality, each module has some variety of applications and functions. Modular programming has an emphasis on the small parts of the program to achieve reliability, reusability, and readability. It also helped with debugging the complete program. When the requirement changes, then it is suitable to implement changes into the code at any point in the development process. Object-oriented programming supports modular programming in all aspects. Therefore, modular programming concepts are used in the development of this project work.

7.4 Code development

Considering the modular programming and class diagrams, tool development is divided into 3 sections. First is collecting data types called as attributes, second is reading all nodes and storing it in lists and third is displaying the data in the form of the chain as per the user requirements.

The first section of collecting the data types includes 3 classes called a Module, RunnableEntity and Signal class. Module and RunnableEntity classes have private, public data members as variables and functions. Variables are the attributes of the entity i.e. the attributes which are shown in the class diagram block in figure 7.3. The getter and setter functions are declared in a public access control level which is used to access the private variables. The data variables of a Signal class are defined in a protected access specifiers because it is an abstract parent class for 3 types of signals. The body of the class is shown in listing 7.1 below as an example.

```
ref class Module;
ref class RunnableEntity;
ref class Signal;

public ref class Module {
public:
        property String^ Name {
                String^ get() { return _name; }
                void set(System::String^ name) {
                this->_name = name; }
        }
private:
        String^ _name;
};
```

Listing 7.1: Example of class body of data types

```
public ref class Reader {
public:
        Reader(System::String^ filename);

        List<Module^>^ ReadAllModules() :
        List<RunnableEntity^>^ ReadAllRunnableEntity();
        List<Signal^>^ REadAllSignal();

private:
        Module^ ModuleFromNode(XmlNode^ functionNode);
        RunnableEntity^ RunnableFromNode(XmlNode^ runnableNode);
        Signal^ SignalFromNode(XmlNode^ signalNode);

private:
        System::String^ _filename;
};
```

Listing 7.2: Example of class body of reader

The second section is the Reader class, which includes the loading of the XML file and reading each node line by line. All nodes are encapsulated with some entities and have a relation with the child nodes. Therefore, 3 functions are made to distinguish the nodes as module, runnable and signals and stored them in different lists with their attributes. As per the class diagram, the associations can be established between these lists. The header file includes the declaration of read function which is shown in the listing 7.2 and .cpp file includes its implementation. The pseudo

code associated with the implantation of the read functions with the help of XML libraries is shown in listing 7.3.

```
// To read all functions
List<Module^>^ Reader::ReadAllModules() {

        Xml::XmlNodeList^ functionNodes =
            xmlDoc->SelectNodes(xpathForFunction);

        // For reading all nodes
        for each(Xml::XmlNode^ functionNode in functionNodes) {

        }
}

//To read all attributes of function
Module^ Reader::ModuleFromNode(Xml::XmlNode^ functionNode) {

        Xml::XmlAttributeCollection^ nodeAttributes =
            functionNode->Attributes;

        // To read attributes of node
        for each(XmlAttribute^ nodeAttribute in nodeAttributes) {

        }
}
```

Listing 7.3: Example of read function

In the first and second sections, all the required data is collected in a sophisticated manner i.e. all tasks, runnables, different signals are collected in a separate list with their attributes. Now the remaining third task is to display this data in a proper manner to form a chain of runnable. Therefore, a window structure is created in a visual studio which serves the purpose of the user interface of the application. The outline of the main window is shown in figure 7.6. Each button on the window UI represents one function in the .cpp file. The functionality of that button or text box or a panel is defined in that function implementation.

Two inputs are mandatory to start the application. The first input is the XML file which data want to analysis and the second one is the name of the signal from the file to start the chain. The UI is divided into 2 panels. The centered panel is used to show all the possible connections between the runnables. As per the selected signal, the centered runnable is decided. The input and output signal of the cen-

tered runnable accompanied by the successor and predecessors of the respected runnable. When the user is selected one of the successor or predecessor runnable, then it gets updated in the chain of the bottom panel. Some buttons are provided to perform predefined operations and ease user interaction. For example,

1. 'Select file' button, which is used to ask the user to select the XML file from the directory.
2. 'Filter by the signal name' and 'Filter by the runnable name'- this search box is used to filter the runnable by the signal name or the runnable name. For example, in many cases approximately, 20 to 30 runnable are present as predecessors or successors. So these search options are saved the efforts of finding out the required runnable.
3. Vertical and horizontal scroll bars adds the flexibility to the predetermined direction. The tool overview is shown in figure 7.6.

7.4.1 Use cases

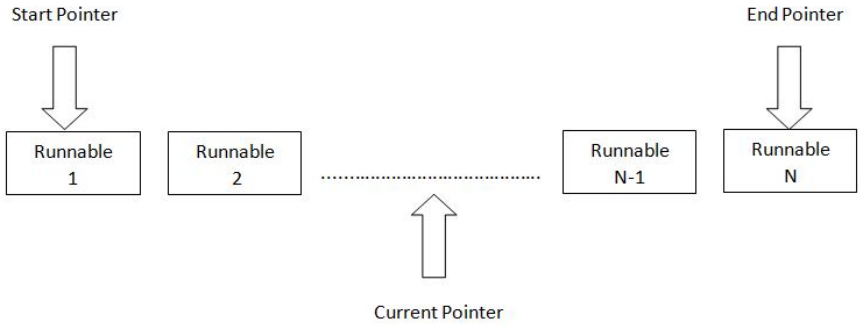

Figure 7.4: Runnable on the bottom panel

Use cases are a methodology that describes how a user uses a system or application to accomplish a certain goal. The use case is consists of the set of interactions between the system and the user in a particular environment. It can be the collection of scenarios. The 'actors' which are the users of the application, 'system' which defines the functional requirement and the 'goal' is the event or the procedure to achieve a certain function are the main members of the use cases. The entire process of how to open an application till how to achieve a certain goal is mentioned in the use cases. It also helped to set the constraints on the use of some options. It is used for the developer to write code according to the situation described in the use cases. The detailed use cases for the application is given below. It illustrates how to form a chain on the bottom panel.

Usecase 1-1: Adding a runnable to the front of the chain	
Precondition	The current pointer is equal to the start pointer of the chain.
Trigger	The user clicks on a predecessor in the upper panel.
Postcondition	The predecessor selected added to the front of the chain. The start pointer is now pointing at the new start. The current pointer is now pointing at the new start. The newly selected runnable is the one in the center of upper panel. Its predecessors are shown. Its successors are shown but faded out but only the previous start of the chain is highlighted.
Usecase 1-2: Navigating through the chain to left (upper panel)	
Precondition	The current pointer is not equal to the start pointer and the current pointer is not equal to the end pointer.
Trigger	The user clicks the highlighted predecessor in the upper panel.
Postcondition	The current pointer is now pointing to the selected predecessor. The selected predecessor is the one in the center of upper panel. Its predecessors are shown but faded out. Its successors are shown but faded out. The selected successor and predecessor is highlighted.
Usecase 1-3: Navigating through the chain to right (upper panel)	
Precondition	The current pointer is not equal to the start pointer and the current pointer is not equal to the end pointer.
Trigger	The userUsecases clicks the highlighted successor in the upper panel.
Postcondition	The predecessor selected added to the front of the chain. The start pointer is now pointing at the new start. The current pointer is now pointing at the new start. The newly selected runnable is the one in the center of upper panel. Its predecessors are shown. Its successors are shown but faded out but only the previous start of the chain is highlighted.

Table 7.1: Usecases for top panel

Usecase 1-4: Navigating through the chain in the lower panel	
Precondition	
Trigger	The user clicks on a segment of the chain in the lower panel.
Postcondition	The runnable which was clicked on is the centered one in the upper panel. The current pointer points to the position of the runnable in the chain. If the current pointer is now the start pointer then all predecessors of the selected runnable are shown and all successors are shown but are faded out except the selected one. If the current pointer is now the end pointer then all successors of the selected runnable are shown and all predecessors are shown but are faded out except the selected one. If the current pointer is greater than start pointer and less than end pointer, the successors are predecessors are shown but faded out and the selected oneâs which are on the chain are highlighted.
Usecase 1-5: Adding a runnable to the end of the chain	
Precondition	The current pointer is equal to the end pointer of the chain.
Trigger	The user clicks on the successor on the panel.
Postcondition	The selected successor is added to the end of the chain. The end pointer is now pointing to the new end of the chain. The current pointer is now pointing to the new end of the chain. The newly selected runnable is located at the center of the upper panel. Its successors are shown. Its predecessors are shown but faded out.

Table 7.2: Usecases for bottom panel

Usecase2-1: Deleting the last runnable from the bottom panel	
Precondition	The current pointer is equal to the end pointer of the chain.
Trigger	The user clicks on the delete button on the runnable entity.
Postcondition	The selected runnable entity is deleted i.e. removed from the bottom panel. The second last runnable entity is newly become the last entity. The end pointer is pointing to new last entity. The current pointer is pointing to new last entity. The newly becoming last is shown on the center of the upper panel. The successors are shown. The predecessors are shown but faded out, only the previously selected is highlighted.
Usecase2-2: Deleting intermediate runnable from bottom panel	
Precondition	The current pointer is greater than start pointer and less than end pointer.
Trigger	The user clicks on the delete button on the runnable entity.
Postcondition	The selected runnable entity and all further entities are deleted i.e. removed from the bottom panel. The end pointer is pointing to newly becoming last entity. The current pointer is pointing to newly becoming last entity. The newly becoming last entity is at the center of upper panel. All predecessors are shown but faded out and only the previously selected one is highlighted. All successors are shown.

Table 7.3: Usecases for breaking chain

7.4.2 Output of runnable chain formation

The procedure discussed in the use cases is helpful to achieve the required output in terms of the runnable chain.

Step 1: Select and input the XML file from the directory.
After providing the XML file, the program will read all runnables and ask the user about preference to select one runnable from where the chain has to start. The preference box is shown in the figure 7.5 and the start of the chain is explained in figure 7.6.

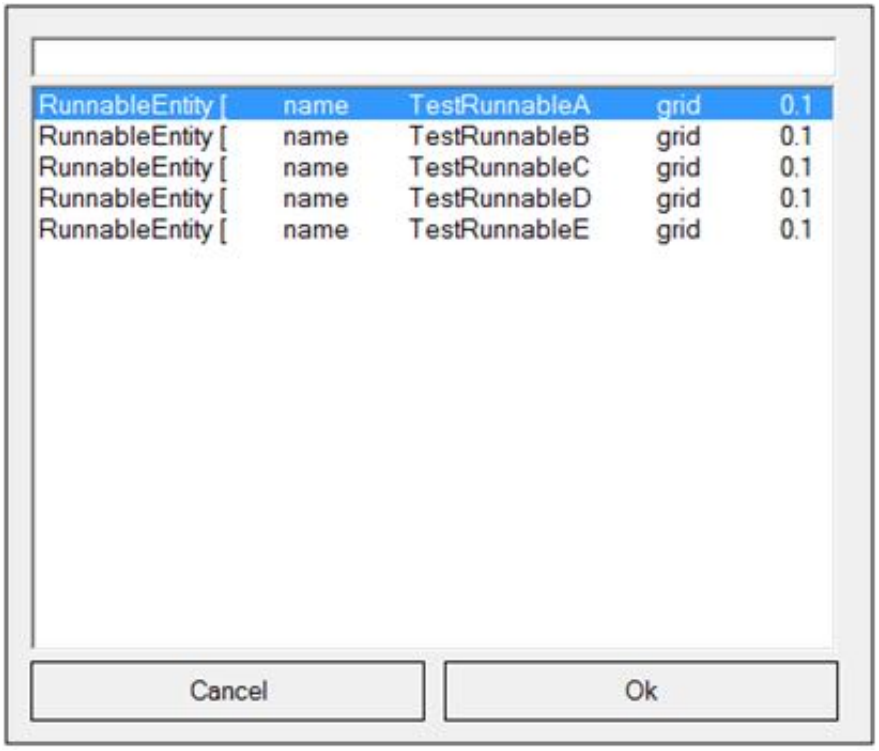

Figure 7.5: Preference box to select starting point of the chain

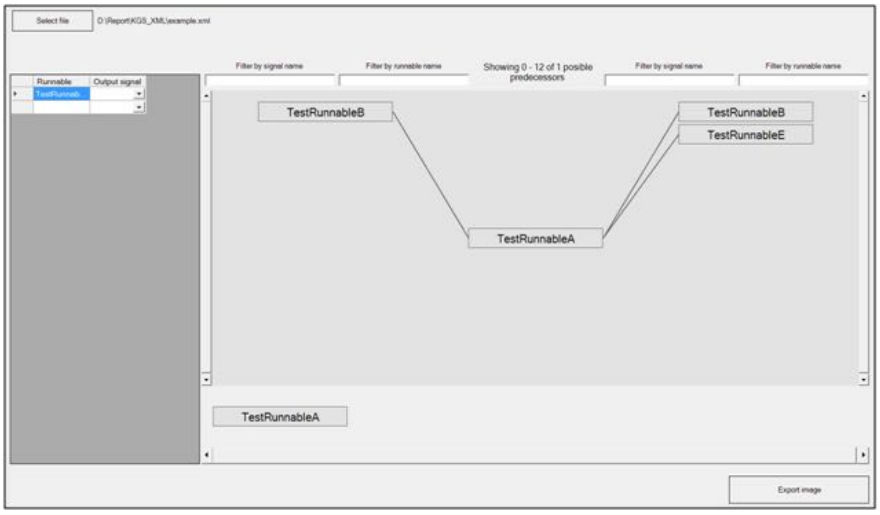

Figure 7.6: User interface of the tool

Step 2: Adding the successors For each runnable successors and predecessors are shown on the middle panel by the solid lines. As per the requirement user can click on one of the available successors and the same will get added in the bottom panel

and chain continues further on the right side. At this situation, the user is not allowed to add predecessor in between the chain. It has to add at the first part of the chain so the chain can continue on the left side. The hidden blocks indicate that the user is not allowed to click on such runnable.

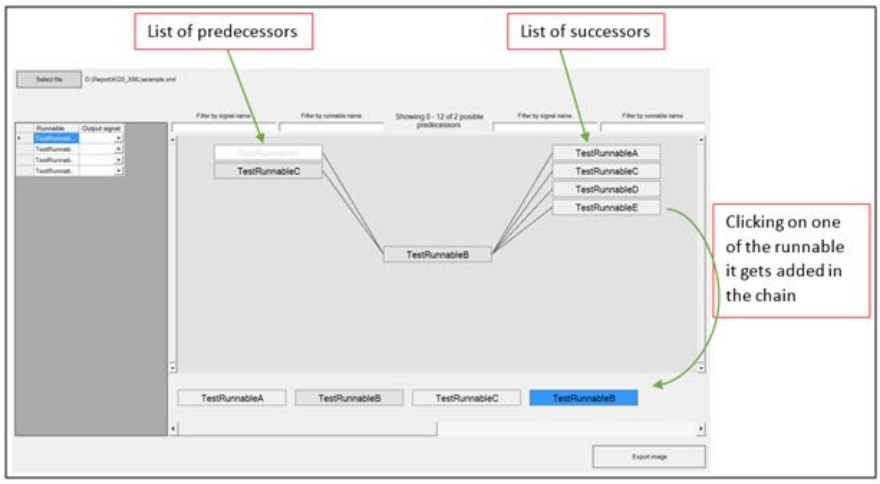

Figure 7.7: Adding the successor

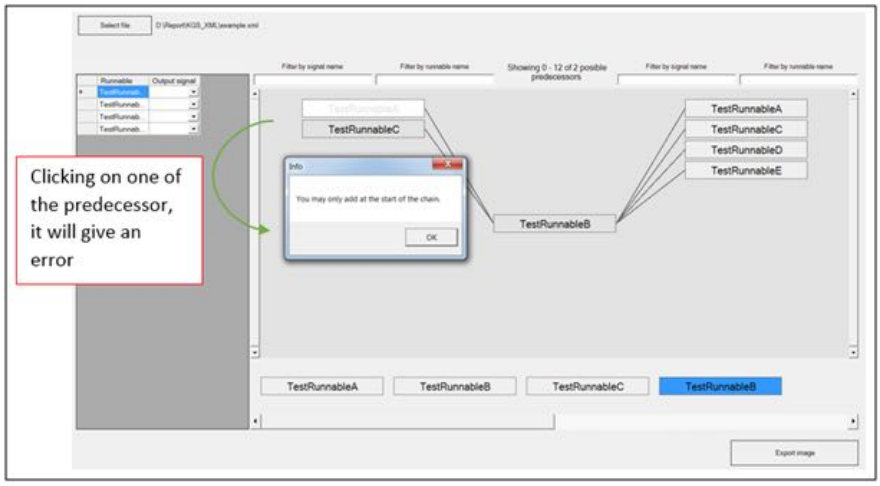

Figure 7.8: Not allowed to add predecessor in between

Step 3: Adding a predecessor

Predecessors can be added at the left side of the chain. Therefore, the user has to select first runnable, then all available successors and predecessors will be visible. And at such a point user is not supposed to add successors. The addition of predecessors is illustrated in figure 7.9 and 7.10.

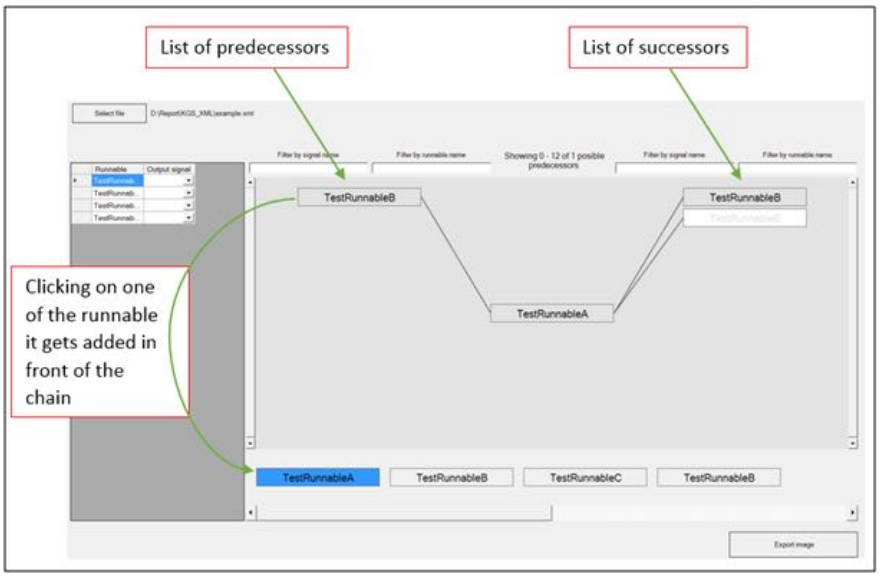

Figure 7.9: Adding the predecessor

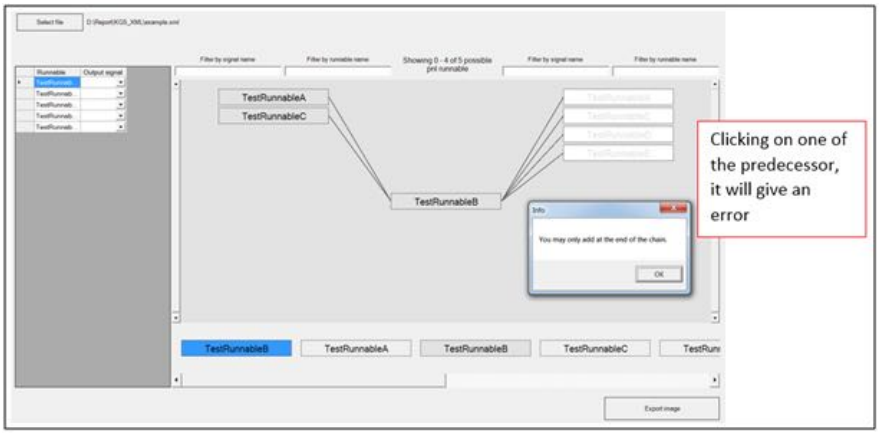

Figure 7.10: Not allowed to add successor in between

Step 4: Deleting the chain

The chain can be deleted completely or partially. If the user required to change or replace certain runnable then it is permitted to delete the runnable. But the constraints are applied to this command. If one runnable is want to delete then it must have to delete all the successors or all predecessors. Because, all runnables are connected to each other, so deleting a single entity in between can cause the loss of reference for the entire chain.

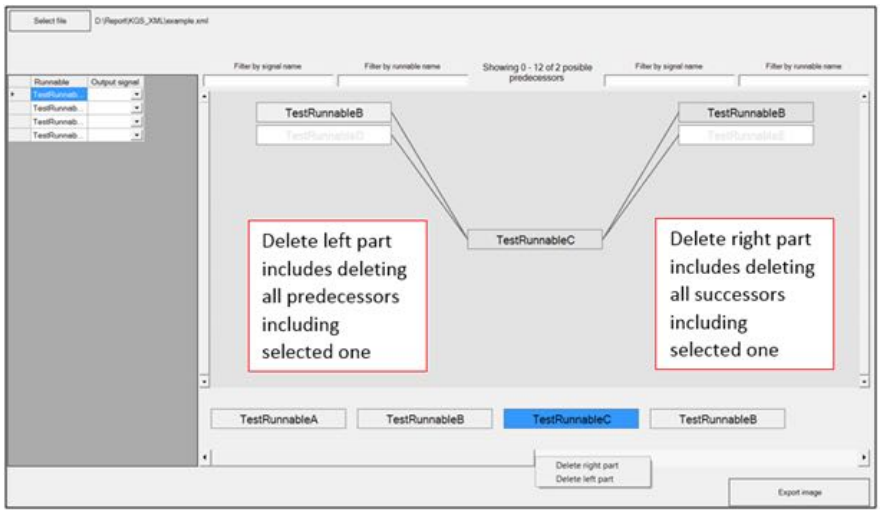

Figure 7.11: Deleting the chain partially

8 Hardware Topology - Tool Release #2

8.1 Concept of hardware topology

The E/E architecture as explained in section 2.4.2 have more complexity because of interconnections between ECUs, sensors, actuators and the communication media. The description of all these networking components is broadly named as a hardware topology. The powertrain contains different types of control units such as ECUs, powertrain control units, torque control unit and they are transmitting data between each other via communication buses like LIN, CAN, Ethernet, FlexRay, etc. The overview of the different ECUs and their connections via buses are illustrated in figure 8.1.

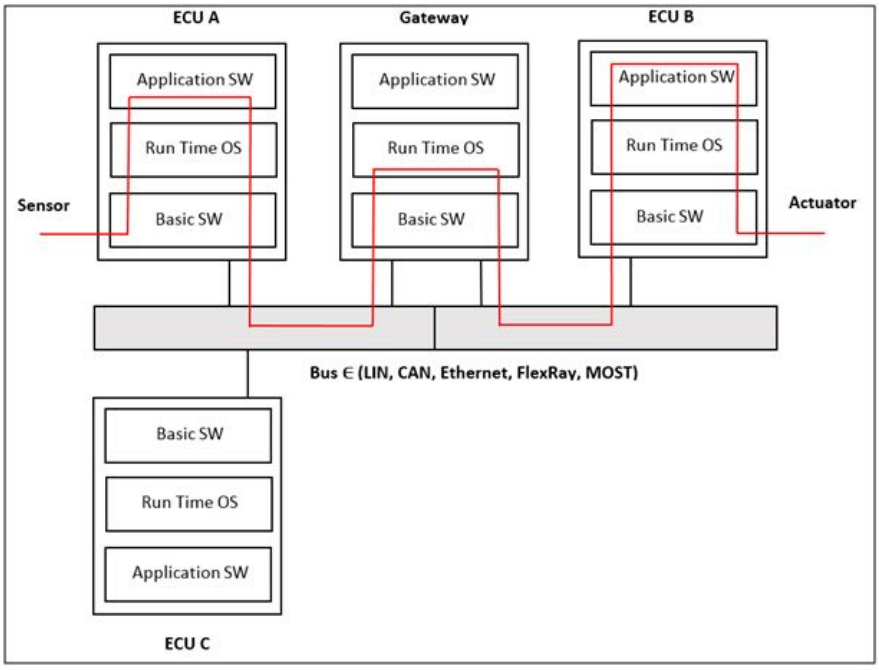

Figure 8.1: Overview of ECU and gateway connection

The signal starting from the network has to transmitted through different ECUs as per the determined path and finally, again it becomes the network signal at the ac-

tuator. While transmitting via ECUs, it has to process by all the layers of ECUs, such as application software, run-time operating system and basic software, etc. The inside parts of the ECU software modules are already discussed in section 6. In this section, the main focus is on the communication network. The connection between two ECUs is carried out by the buses and special control unit called a gateway. In general, the gateway is defined as a link between different elements. In hardware topology, it is a control unit which denotes the linkage between two different networks or different communication protocols. It is often used for the converter between different formats. Therefore, it can be possible in powertrain architecture that the connection between ECU A and gateway is via CAN bus and gateway to ECU B with the help of LIN bus. Therefore, signal processing from ECU A to ECU B has to pass through CAN and LIN. Moreover, the communication protocols of all the buses are different. Hence to analyze a hardware topology it is necessary to consider physical channels, connectors at ECU nodes, OSI layers, physical layers of buses and their communication protocols.

AUTOSAR (AUTomotive Open System ARchitecture) is a standard an open automotive architecture. It helped to analyze software and hardware architecture independently. The different layers of AUTOSAR are shown in figure A.2. AUTOSAR helped to manage the complexity of the interconnections and mapping between all components and their communication network. Therefore, it is a key enabling the hardware topology of the electric architecture of the automobile. It is used to extract the entire ECU or architecture in one file named as ARXML (AUTOSAR XML). It mainly focused on how ECUs are communicating with each other i.e. the entire communication cluster including signals and buses and their connections.

8.2 Definitions of communication cluster

1. Cluster: It is the group of ECU nodes which are connected by a communication medium like LIN, CAN, etc.
2. Channel: A Communication network provides the channels for data transmission.
3. Controller: It is a hardware component that allows to ECU to send and receive the data on the communication medium.
4. Connector: It is the bus interface of the ECU which represent send and receive behavior of the node.
5. Frame: It is a pack of information that shared over a communication network.
6. PDU: It is a Protocol Data Unit which is a collection of signals.

7. Frame Triggering: It represents the condition on which the frame is transmitted on the network [Bos14]

8.3 Focus of ARXML

Section 2.8 discussed the different buses and their communication protocols. The communication protocols are the methods by which two electronic units can exchange the data with each other. To ensure the runtime accurate exchange of data and timely delivered the commands to the other units or actuators, it is crucial that the protocol must be ensured the response time of the frame. Therefore, the timing relevant attributes of the protocols are very important in data transfer. Each unit of the communication cluster such as PDUs, frames, signals, buses, ECUs, controllers, etc. have the time relevant attributes. The main focus of hardware topology is shown in figure 8.2.

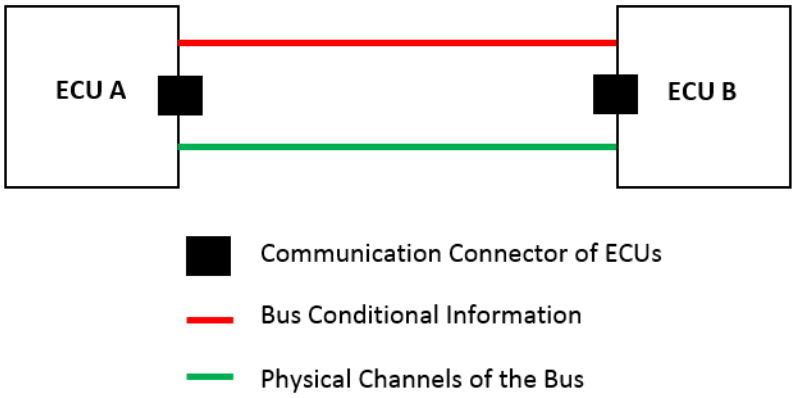

Figure 8.2: Focused of hardware topology

The connection between 2 ECUs is mainly focused on communication connectors between them, information about the bus and the physical channels. Therefore the main aim of consideration of hardware topology is to find out the timing attributes of these components. All the information of these components will be available in the ARXML file in terms of packages. The packages example is shown in figure 3.3. The main tasks are to deal with the ARXML file to extract the relevant information regarding figure 8.2. It includes understanding the hierarchy of the file, contents of each package, references and xpaths in the file, etc. The main challenging part is to determine the bus conditional information i.e. how the frames or services, PDUs and signals are encapsulated in the packages of different buses. Secondly, it is very important to find out the significant time relevant attributes which affect the data

transfer. The third interested part is the physical channel of the bus. It includes the transfer protocols of address, channels and nodes. Moreover, it is also related to the frame, PDU and signal triggering.

8.4 Communication Cluster

8.4.1 ARXML: OSI Layers

The Open System Interconnection Models (OSI model) is a standardized communication functions which define the framework to implement the protocols on different layers. OSI model is composed of 7 layers where application layer is closest to the user and physical layer is at the down where actual data transfer will take place. In the OSI model, the data is passed from one layer to another starting from the top layer (application layer: layer 7) at one node and proceeding to the bottom layer of physical layer and leads to the another node, then again follow the hierarchy in opposite direction of layers. OSI layers are helped to divide the large data into smaller segments. At each layer, the some part of data gets added in a protocol. The detailed information about the OSI layer is shown in the table 8.1.

Layer	Layer Name	PDU	Function
7	Application Layer	Data	
6	Presentation Layer	Data	
5	Session Layer	Data	
4	Transport Layer	Segment	Responsible for delivery of a message from one process to another
3	Network Layer	Packet	Responsible for the delivery of packets from the source host to the destination host
2	Data Link Layer	Frame	Responsible for moving the frame from one node to another
1	Physical Layer	Bits	Responsible for transmission of Bits

Table 8.1: OSI Model

Each layer is associated with the PDU of different information. PDUs are used for the peer to peer contact between corresponding layers. Data is handled by the top

2 layers and then segmented by the transport layer. Network layer places the information into the packet and data link layer frames the packets for transmission.

8.4.2 Aim of communication cluster consideration

The figure 2.5 showed that powertrain or the entire vehicle have different ECUs located at different places in vehicle architecture. The different ECUs are connected via different buses. In a field of the project work, time estimation of different buses for data transfer is concerned.

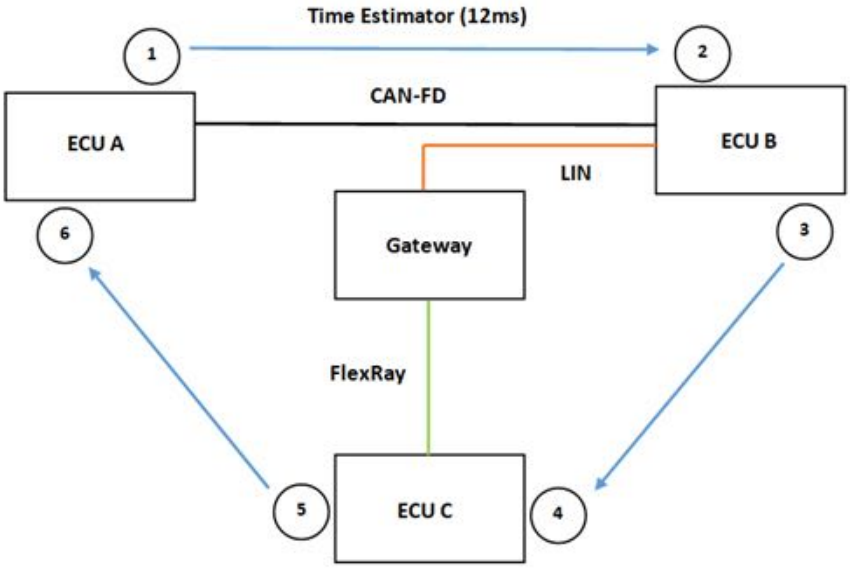

Figure 8.3: Requirement of time estimation by considering bus and ECU nodes

The main focus of the work is to estimate the time required to communicate between 2 ECU nodes via a specific bus. For such estimation of time, it is important to know how the message is transforming, which components are involved in the process, description of bus protocols, and their latencies, etc. This background information is used to find out the timing attributes of the specific bus, which are further used by the formal analysis methods of a system for the creation of an abstract model. In such cases, it is required to consider the protocol stack of the communication medium. Furthermore, the work is concern about OSI layers to find out which layer is significantly important at a certain level of communication. Hence, accordingly, the timing attributes related to the layer will come into the picture in runtime execution. For example, when horizontal communication between 2 nodes at communication controller is estimated then the timing attributes related to OSI layer 1 and 2 and data unit called as frames need to be considered, which is explained by

the figure 8.4 below. Such type of consideration is used to create an abstract model of the protocol stack.

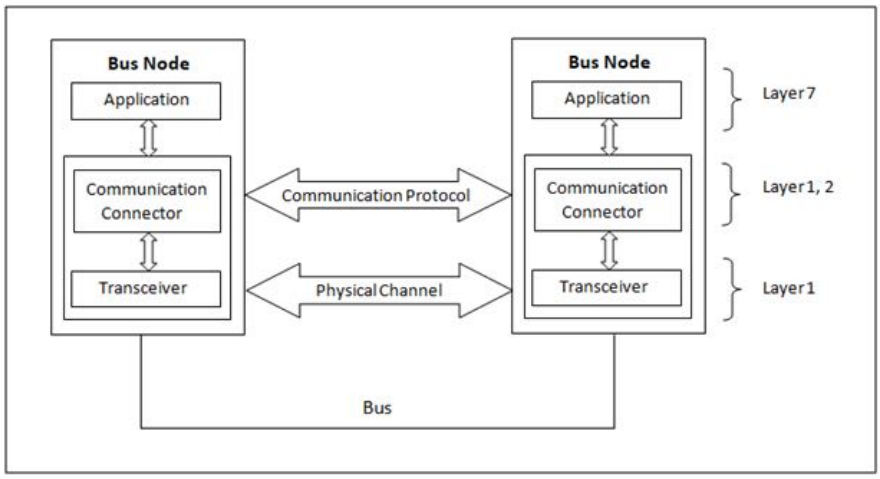

Figure 8.4: Bus node with OSI layers

Secondly, when talking about abstract generalize model, it is important to consider all the buses and their communication properties. As discussed earlier, different protocols are followed by different buses. Therefore, it is necessary to find out common layers used by the buses during different levels of communication. It is used to write a code for reading communication cluster of ARXML files. The detailed study of all bus protocols are carried out and common properties are noted down.

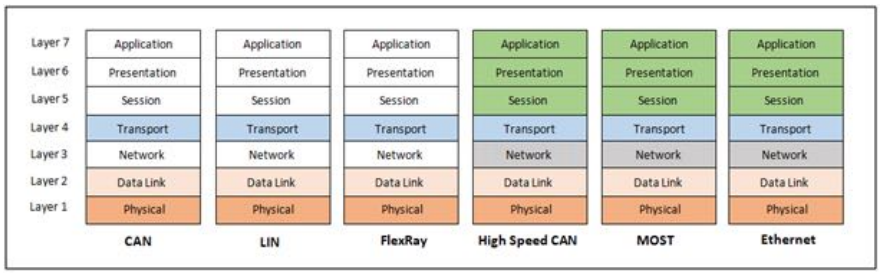

Figure 8.5: Overview of common layers in communication stack within all buses

For example, by considering LIN protocols as a case study, the LIN specification package consist of 3 important parts. First part is LIN protocol specification which is described by the Physical and Data-link layer. Second is LIN configuration description which is described by the Network layer. It includes information about the number of nodes, frames, baud rate (is the rate at which the information is transferred in a communication channel), etc. The last part is LIN API (Application Program Interface) which is the software implementation encapsulated in layer 7. A

similar study is carried out on all respective buses and the common parts are determined which are shown in figure 8.5. It results as Can, LIN and FlexRay buses need physical, data link, transport and application layer. MOST, High Speed CAN, and Ethernet bus covers all the 7 OSI layers for communication. All the buses used the transport layer for the diagnostic services. The network layer is used for the routing support for IP, Car2X and In-Car Wireless Communication.

9 Code Development for ARXML

9.1 Analysis of ARXML

ARXML file has the representation of hardware topology in the form of packages. These packages represent different views of the system. It shows how the control units are communicating with each other. Moreover, the entire topological information is stored in the form of XML nodes, subnodes and child. The ARXML is focused on either specific bus or the ECU like central ECU of the powertrain. Therefore, in a similar way of the SW-XML, the analysis of ARXML is carried out in 2 steps.

The first part is to debug the encapsulated information of the hardware topology from all the packages. It includes an understanding of the information available in the respected package and its attributes. For example, there is a package called I-Signal, which is the instance of the signal which sends via buses. Under work of interest, it is required to collect all signal names and their time relevant attributes. Furthermore, I-Signals are referred to the system signals which describe the system value. For example, value of the engine torque. When such a value is send via bus then it become the instance. These system signals are also encapsulated in a different package. Therefore, collecting the system signals and finding out the reference of system signal from I-Signal is important task in analysis. The second part is writing a code for reading the ARXML file, and collecting the timing attributes of each components in consideration.

9.2 Challenges in development

The first step in the code development is to analyze the ARXML file and its structure. Accordingly, the strategy is made to write a code in C++/CLI with the help of XML to read the ARXML and collect the required data. Consideration of all versions of ARXML file such as version 3.1, 3.2, 4.2, etc. focused on all buses is the key point in creating the generalize software. The different versions of ARXML have different package names. For example, package name could be ISignals in version 3.1, I_SIGNALS in version 4.1, etc. Moreover, the hierarchy of the file is also different. For example PDU package in at level 1 in version 4.1 but at level 3 in version 3.1, etc.

The different hierarchy and package abbreviation are shown in figure 9.1. Therefore, it is necessary to develop a program which takes all such cases under consideration.

When the file is specific to the bus then it has variation in communication cluster. Because the protocols are different for Ethernet and LIN, CAN, etc. Therefore the requirement in a program is to recognize the communication protocol and accordingly collect the timing attributes with its value. Frames of LIN, CAN, FlexRay are statically generated and encapsulated in the ARXML but the frames of Ethernet are dynamically generated hence not directly specified in the ARXML file. They are indirectly specified as services. Such type of variation in data transfer is the challenging task in creating the generalized code for communication.

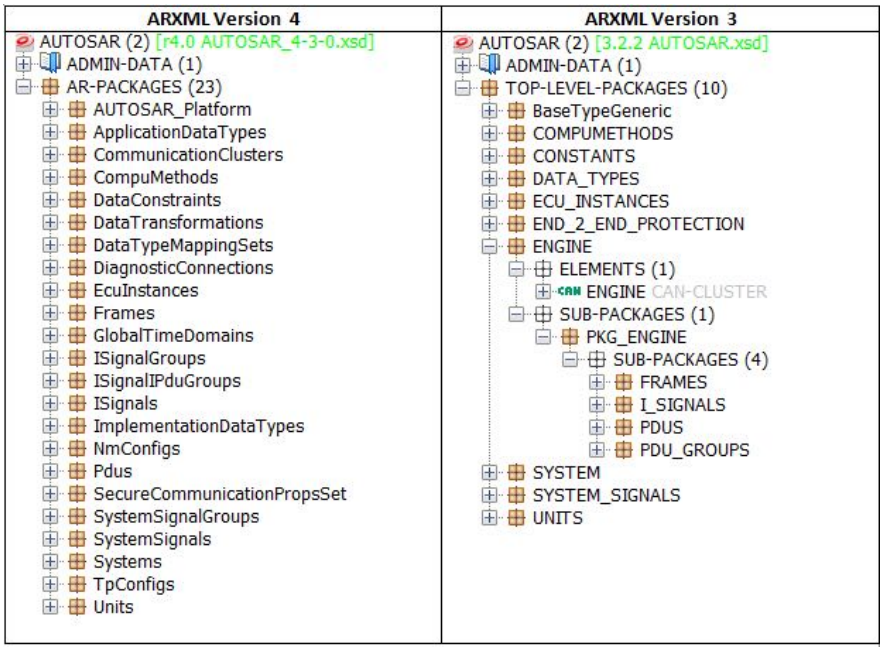

Figure 9.1: Different versions of ARXML

9.3 Development strategy

The program development strategy is divided into 2 parts. Initially in the first part, all communication protocols are studied on one hand side and on other hand timing attributes of all components from ARXML file are found out. These timing attributes are separately stored in the notepad file. One notepad file is created for timing attributes of one component and it is specific to the ARXML version. These files are stored in one of the folder in the system. The second main part of the strategy is to write a code in C++/CLI and XML to read all nodes of the ARXML file and their attributes are collected. These attributes are compared with the respective notepad

file. For example, the part of code is used to read the PDU package, then attributes of PDUs are compare with PDU_version3/4.txt file. When the match is found then those attributes and its value are selected and stored.

The following pieces of pseudo codes are written and build together to collect timing attributes of ARXML file. Firstly, different classes are created for different components which are shown in the listing 9.1. Each class represents public, private and protected data members which are the attributes of their respective packages and xpaths in ARXML file.

```
namespace NetDatatype {

                ref class ARXML_ECU;
                ref class ARXML_Port;
                ref class ARXML_Connector;
                ref class ARXML_Controller;
                ref class ARXML_Bus;
                ref class ARXML_Channel;
                ref class ARXML_Frame;
                ref class ARXML_Socket;
                ref class ARXML_PDU;
                ref class ARXML_SignalGroup;
                ref class ARXML_Signal;
        }
};
```

Listing 9.1: Overview of the all classes

Another important class is the TimingAttributeType class in which the information related to timing attributes and its value type are collected. The pseudo code is given in listing 9.2. The data members of this class are used to identify name of the attribute, the path of that attribute in ARXML file, relevant bus where the attribute referred to and its value.

```
public ref class TimingAttributeType {
public:
        TimingAttributeType(System : String ^ attributeName,
                System::String^ arxmlRef) {
                this->Name = attributeName;
                this->ARXML_Ref = arxmlRef;
        }

        property TimingAttributeParent Parent;
```

```
        property System::String^ Name;
                    /*the name of the attribute*/
        property System::String^ ARXML_Ref;
                    /*where to find it in ARXML*/
        property System::String^ ParentVariant;
                    /*e.g. CAN/FlexRay for bus*/
        property System::String^ ValueType;
                    /*e.g. int, float or string*/
};
```

Listing 9.2: Collection of timing attributes information from separate class

In the communication protocols, the main parts are frames, PDUs and signals. So as per the flow of data, frame has PDUs and PDU has signals. Therefore, the same concepts are implemented in the code. The frame class has PDU collection function and PDU class has the signal collection function. The pseudo code of frame class is represented in listing 9.3.

```
/*Frame, PDUs, SIgnals and Signalgroups*/
public ref class ARXML_Frame : ARXML_Identifiable {
public:
        ARXML_Frame(System::String^ uid, System::String^ name) :
                ARXML_Identifiable(uid, name) {
                _pdus = gcnew List<ARXML_PDU^>();
        }

        /*setter*/
        void AddPDU(ARXML_PDU^ pdu) { _pdus->Add(pdu); }

        /*getter*/
        List<ARXML_PDU^>^ GetPDUs() { return _pdus; }

private:
        List<ARXML_PDU^>^ _pdus;
};
```

Listing 9.3: Example of frame class representation

Similarly when talked about communication cluster, the class member contains ECU, bus, frames, socket, channel and connector, etc. These above classes are used to build the data types of communication cluster of ARXML. Now another type of class required is the reader class, which reads the file nodes and attributes. From the figure 9.1, it is clear that package names, and hierarchy is different for different

versions. Therefore, separate reader files are created for different versions especially for version 3 and version 4. This file has special functions to find the packages which are specific to the XML node in hierarchy. Moreover dictionaries are also created to fill all the types at the beginning (refer to listing 9.4) and it can be used throughout the program for various purposes and make the program general.

```
Dictionary<String^, ARXML_Frame^>^ frames =
        gcnew Dictionary<String^, ARXML_Frame^>();

array<System::String^>^ frametypes =
gcnew array<System::String^>{
        "ETHERNET-FRAME", "CAN-FRAME", "FLEXRAY-FRAME",
        "FRAME", "LIN-FRAME"
};
```

Listing 9.4: Dictionary of the frame

Furthermore, separate functions are required to read the timing attribute file line by line and to compare the specific XML node attributes with the file. These two functions are shown in the listing 9.5.

```
array<String^>^ lines = System::IO::File::ReadAllLines(
        ECT::Resources::GetFileByName("BUS_V3", ".txt"));

_busTimingAttributeTypes =
        ParseTimingAttributes(lines, TimingAttributeParent);
```

Listing 9.5: Reading and comparing file with attributes

9.4 ARXML Output

When certain ARXML file of any version is given as an input to the program then the program determines participation of communication components with names and their timing attributes with the value. The sample output is shown in the figure 9.2. It is helpful to visualize the complete communication cluster. From this output developer can acquire a data of many cases or scenarios of states such as it is used to determine the value of attributes when signal is passing through certain bus or ECU, etc then which connector or controller is taking a part and what are the values of their timing attributes. Furthermore, information about all layers of communication can be known from physical channel, Frames and PDUs, etc. In other way, the

entire information of real time embedded system can be known from this ARXML reader program.

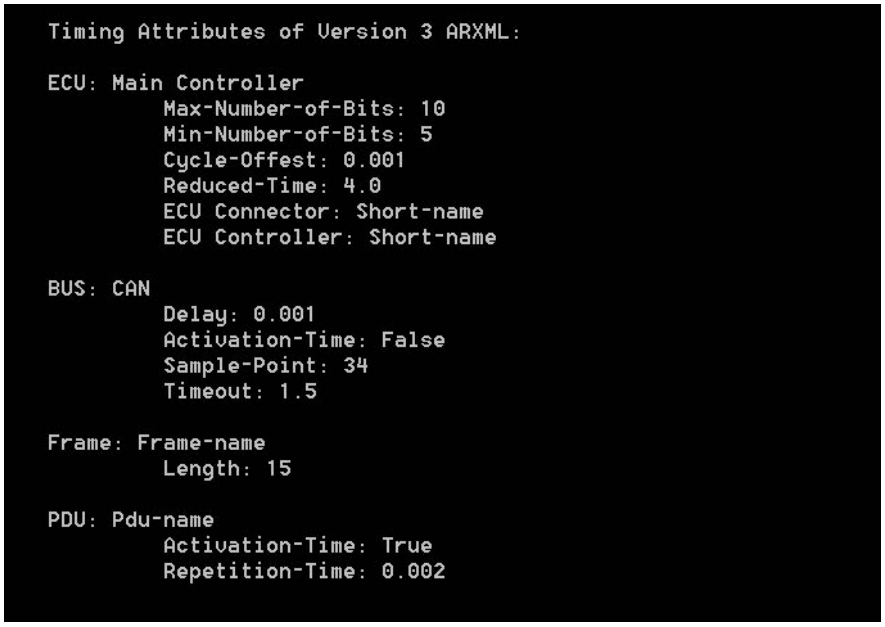

Figure 9.2: Output of ARXML code

10 Database Management-Tool Release #3

10.1 Necessity of database management

In the cyber physical system, the project is dealing with 2 different topologies and that are considered as a real time embedded and software systems where parameters are continuously changing. On the other hand side, the timing relevant data which is collected by software and hardware topologies are used for the formal analysis methods to find out the worst case scenario. Moreover, in automotive, both the topologies have huge data to deal with. Therefore, it is required to manage the data in a structural way between real time embedded system and the formal analysis method.

10.2 Database management strategy

The database is the organized collection of data stored in same or different computer system. The database is developed using design and modeling techniques. The relational database is used where model the data in terms of rows and columns of the table. The first step in the database is to distinguish between the components storage. Each component relationship gets timing relevant attributes. Therefore, separate table is created for each component. Next step is the creation of data based model by considering the relationship between the components. The connections between frames or services, PDUs and signals are established in the database also. The whole database model is shown in the figure 10.2. In this model, for a simplification of collecting the data, the timing attributes and connecting data are collected at channel, ECU and communication connectors.

To achieve the collection at these components, the SQL database with key value design paradigm is used. With the help of this concept the interconnection between the tables can be easily established. Database stores the data as a collection of key value pair. The key in the key-value pair must be unique and this is the unique identifier to access the data associated with it [KVD]. So it is possible to trace the

interconnected data. For example, the signal table is referred to the PDU table by specifying the key id in it. The example of SQL key-value table for signal attribute is shown in the listing 10.1.

```
SELECT TOP (1000) [prj_id]
          ,[ncd_id]
          ,[nw_id]
          ,[ch_id]
          ,[res_id]
          ,[pdu_id]
          ,[signal_id]
          ,[attr_id]
          ,[attr_parent]
          ,[attr_value]
          ,[attr_value_type]
From [DB].[TejasTestData].[NWSignal_TimingAttributes]
```

Listing 10.1: Parameters of signal attribute table with key-value implementation

10.3 Database connection

It is very important to establish the connection between output of software XML and ARXML code to the database. The structure of XML output and database are different. Therefore, the intermediate layer of data types is required which convert both topological outputs in a suitable form to store in a data base. It also used to establish the window connection to the database. Therefore, in windows programming, the retrieving or merging the data after analysis of files can be possible. Therefore both way communications can be possible in a database management. The overview of the database connection is shown in the figure 10.1

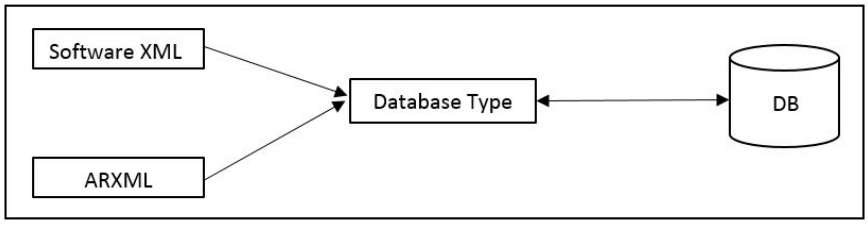

Figure 10.1: Database connection

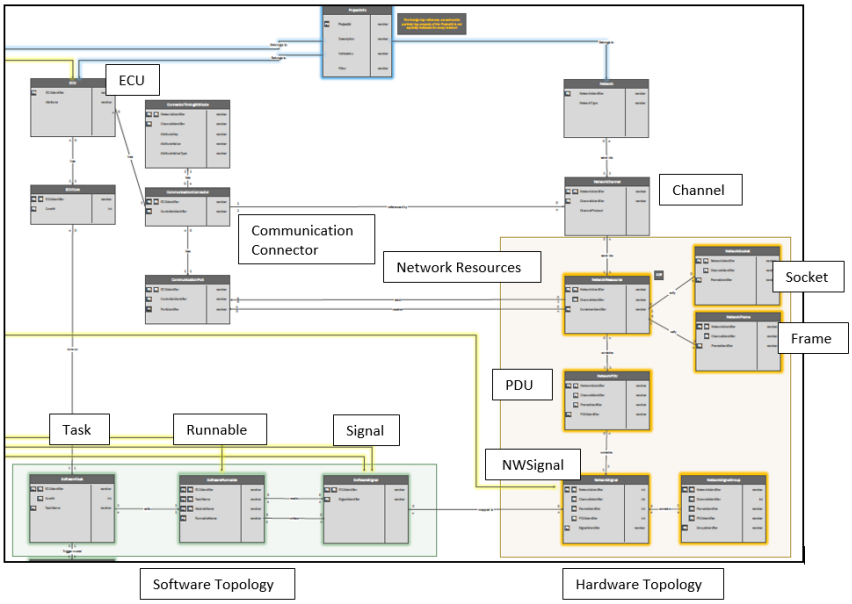

Figure 10.2: Database model of complete project

11 Conclusion

Failure of the real-time embedded system may endanger human life or cause the economic loss for the company. It is very difficult to find out the behavior of the embedded system and predict the performance. The path followed by the signal in all the layers of the communication plays an important role in the performance.

The performance analysis is an important key in the design process of the distributed embedded system. It is used to analyze the performance characteristics of the system in the early phase of the design. Furthermore, this analysis helps the designer to find out the path of the signal, the time required for execution and artifacts required in the system. The tool created in the project work helped to deal with the complexity of ECU architecture and communication network by bringing hardware and software topologies together. The developer can visualize the complete data model of the communication network. The system model and the formally described cause effect chains are brought together using database. All the timing attributes along with their values are stored in the database. This shows the complete data provision which is required for the formal analysis of cause effect chain is achieved in the tool. These data models are further used to find out the worst time scenarios in cause-effect chains.

In the future, there is a possibility of changes in software modules, runnable names, etc. Therefore, consideration of such variation in the tool can be the future work.

A Apendix

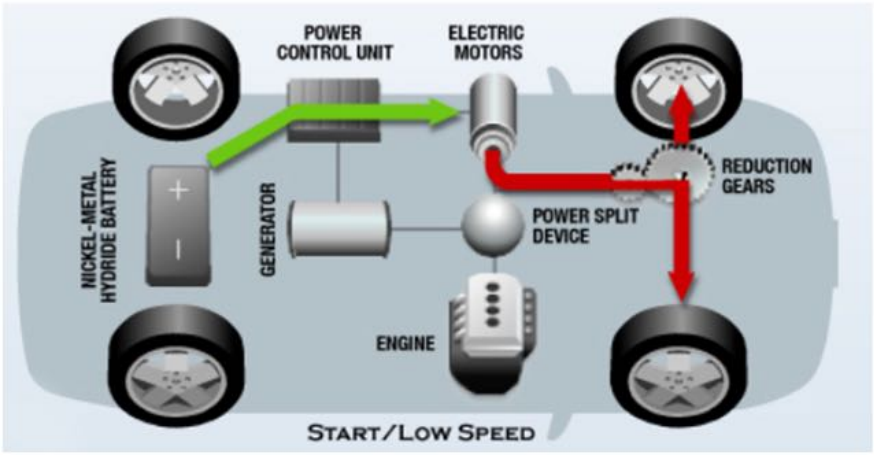

Figure A.1: Electric/Hybrid Vehicle
[EHV]

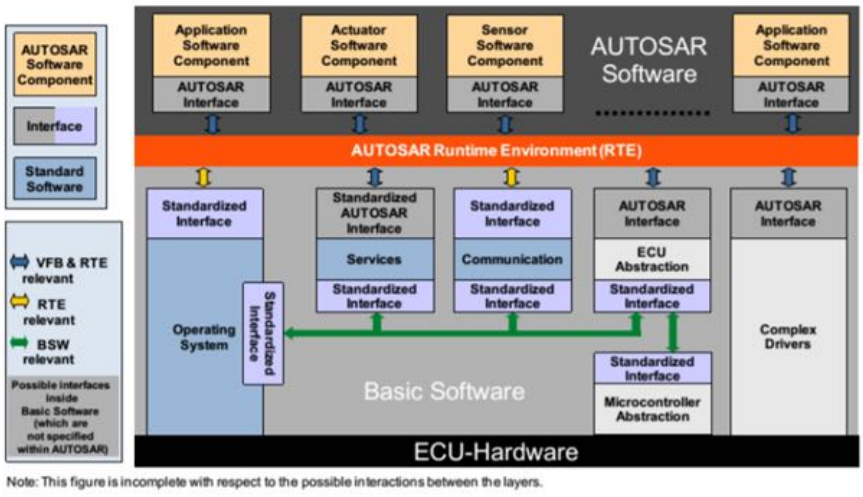

Figure A.2: AUTOSAR Layers
[A.D]

Bibliography

[A.D] A.D., Corporation: *AUTOSAR*. – https://www.global-greenhouse-warming.com/hybrid-electric-vehicle.html

[AUT] AUTOSAR: *AUTOSAR development process*. – https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_TR_TimingAnalysis.pdf

[Bos14] BOSCH, Robert: Bosch Automotive Electrics and Automotive Electronics: Systems and Components. In: *Networking and Hybrid Drive*. Springer Vieweg, 2014

[Con] CONTINENTAL AUTOMOTIVE GMBH (Hrsg.): *Timing Modelling with AUTOSAR*. Continental Automotive GmbH

[EHV] EHV: *Electric and Hybrid vehicle*. – https://www.global-greenhouse-warming.com/hybrid-electric-vehicle.html

[for] FORMATS, Exchange: *Examples of data exchange formats*. – http://www.para-agua.net/en/explorar/herramientas/sistemas-redes-informacion/p3-publicaciones/tipo-de-formato/formatos-de-intercambio-de-datos (n.d)

[HHGS18] HOFSTETTER, Martin ; HIRZ, Mario ; GINTZEL, Martin ; SCHMIDHOFER, Andreas: Multi-objective system design synthesis for electric powertrain development. In: *2018 IEEE Transportation Electrification Conference and Expo (ITEC)* IEEE, 2018, S. 286–292

[HMG11] HEGDE, Rajeshwari ; MISHRA, Geetishree ; GURUMURTHY, KS: Software and Hardware Design Challenges in Automotive Embedded System. In: *International Journal of VLSI Design & Communication Systems* 2 (2011), Nr. 3, S. 165

[KCG+17] KUGELE, Stefan ; CEBOTARI, Vadim ; GLEIRSCHER, Mario ; HASHEMI, Morteza ; SEGLER, Christoph ; SHAFAEI, Sina ; VÖGEL, Hans-Jörg ; BAUER, Fridolin ; KNOLL, Alois ; MARMSOLER, Diego u. a.: Research challenges for a future-proof e/e architecture-a project statement. In: *INFORMATIK*

2017 (2017)

[KVD] KEY-VALUE DATABASE?, What is a.: *Database-guide.* – https://database.guide/what-is-a-key-value-database/

[Lap17] LAPLANTE, Phillip A.: *Requirements engineering for software and systems.* Auerbach Publications, 2017

[LPWT15] LOMONOVA, EA ; PAULIDES, JJH ; WILKINS, S ; TEGENBOSCH, J: ADEPT:"ADvanced electric powertrain technology"-Virtual and hardware platforms. In: *2015 Tenth International Conference on Ecological Vehicles and Renewable Energies (EVER)* IEEE, 2015, S. 1–10

[LSS+12] LUKASIEWYCZ, Martin ; STEINHORST, Sebastian ; SAGSTETTER, Florian ; CHANG, Wanli ; WASZECKI, Peter ; KAUER, Matthias ; CHAKRABORTY, Samarjit: Cyber-physical systems design for electric vehicles. In: *2012 15th Euromicro Conference on Digital System Design* IEEE, 2012, S. 477–484

[MB] MERCEDES-BENZ: *Powertrain Engine Technology.* – https://www.mercedes-benz.com/en/mercedes-benz/vehicles/aggregates/powertrain-engines/(2016, September 14)

[SJ18] SINGH, Ajeet ; JAIN, Anurag: Study of Cyber Attacks on Cyber-Physical System. In: *Communication and Technology* (2018)

[SSH+16] SAILER, Andreas ; SCHMIDHUBER, Stefan ; HEMPE, Maximilian ; DEUBZER, Michael ; MOTTOK, Juergen: Distributed Multi-Core Development in the Automotive Domain-A Practical Comparison of ASAM MDX vs. AUTOSAR vs. AMALTHEA. In: *ARCS 2016; 29th International Conference on Architecture of Computing Systems* VDE, 2016, S. 1–8

[Tut] TUTORIALSPOINT.COM.: *UML - Class Diagram..* – https://www.tutorialspoint.com/uml/uml_class_diagram.htm

[XML] XML: *Managing Data Exchange/Introduction to XML..* – https://en.wikibooks.org/wiki/XML_-_Managing_Data_Exchange/Introduction_to_XML (n.d)