DEGREE PROJECT IN ELECTRICAL ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2019

A Co-Simulation Framework for Black-box Testing of In-Vehicle ECUs

YIFAN RUAN

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

A Co-Simulation Framework for Black-box Testing of In-Vehicle ECUs

YIFAN RUAN

Master in Electrical Engineering
Date: September 20, 2019
Supervisor: Matthias Becker
Examiner: Zhonghai Lu
School of Electrical Engineering and Computer Science
Host company: Volvo Construction Equipment
Swedish title: En Samsimuleringsram för Svartbox-testning av ECUs i Fordon

Abstract

With the increasing complexity of embedded systems in the automobile industry, the demand on the testing process increases simultaneously. Bus communication behavior is an important feature which is used by testers to evaluate functions of the embedded system under test. However, the existing testing method Hardware-in-the-Loop simulation is expensive and it is mainly used at the late stage of the project life cycle. Additionally, the time and resources available for the testing process are very limited.

The thesis designs a co-simulation framework for testing functions of in-vehicle Electronic Control Units (ECUs). It is dedicated to providing a realistic environment for the ECU under test at both early and late stages of the project life cycle and reducing the cost of the testing process. The framework consists of multiple ECUs that are implemented by different methods and connected to the same vehicle bus. A prototype is also implemented to evaluate the design of the co-simulation framework. Multiple simulation tools are used during the process of the prototype implementation. Limitations and future work of the thesis are discussed.

As a proof-of-concept framework, the resulting prototype can be extended to more functions and provides solutions for further research on ECU testing.

Sammanfattning

I takt med att komplexiteten hos inbäddade system i bilindustrin ökar så har även efterfrågan på testningsprocessen ökat samtidigt. Busskommunikationsbeteende är en viktig funktion som används av testningsingenjörer för att utvärdera det inbäddade systemets funktioner i samband med dess testning. Men den befintliga testningsmetoden, hårdvara-i-loop-simulering, är dyr och används främst i det senare stadiet av projektets livscykel. Dessutom är tiden och resurserna som är tillgängliga för testningsprocessen mycket begränsade.

Avhandlingen utformar ett ramverk för samsimuleringstestning för att testa funktioner för elektroniska styrenheter i fordon (ECU). Den avser tillhandahålla en realistisk miljö för ECU-testning i både de tidigare och de senare stadierna av projektets livscykel samt minska kostnaderna för testningsprocessen. testningsramverket består av flera ECU:er som implementeras genom olika metoder och är anslutna till samma fordonsbuss. En prototyp implementeras också för att utvärdera utformningen av testningsramverket. Flera simuleringsverktyg används under processen för implementeringen av prototypen. Begränsningar och förslag till framtida forskning i relation till avhandlingen diskuteras.

Som konceptvalidering testningsramverk kan den resulterande prototypen utökas till att omfatta fler funktioner och utarbeta lösningar för ytterligare forskning inom ECU-testning.

Acknowledgement

The thesis was held at Volvo Construction Equipment (VCE), Eskilstuna. I would like to thank my manager Cedric Millard and company supervisor Hjörtur Reynisson for their excellent support during the thesis. I am also thankful to other team members in the department and engineers from dSpace, ETAS who provide a lot of assistance when exploring potential possibilities.

I would also like to express my sincerest thanks to my examiner Professor Zhonghai Lu and school supervisor Dr. Matthias Becker for providing valuable reviewing, thoughtful comments on my report.

Last but not least, I wish to express my particular appreciation to my parents and friends for their support and encouragement that cannot simply expressed in words.

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

ICT Information and Communication Technology

ECU Electronic Control Unit

MiL Model-in-the-Loop

HiL Hardware-in-the-Loop

SiL Software-in-the-Loop

CAN Control Area Network

ISO International Organization for Standardization

AUTOSAR AUTomotive Open System ARchitecture

UDS Unified Diagnostic Services

OSI Open System Interconnection

CAPL Communication Access Programming Language

TCU Telematic Control Unit

GNSS Global Navigation Satellite System

GPS Global Position System

DSP Digital Signal Processing

FPGA Field Programming Gate Array

VCE Volvo Construction Equipment

FMI Functional Mock-up Interface

FMU Functional Mock-up Unit

MBD Model based development

OEM Original Equipment Manufacturer

Chapter 1

Introduction

Embedded systems are considered as one of the most important application areas of Information and Communication Technology (ICT) and are widely used in automotive electronics, avionics, telecommunication, smart buildings and robotics [1]. Embedded software, which controls the major functions of current embedded systems, plays a significant role in the development of embedded systems, A market analysis showed that software-based implementations account for more than 80% of system development work in the domain of embedded systems [2].

As the complexity and interconnection of embedded software continue to increase in the automotive industry, the demand on the testing process increases at the same time [3]. High pressure imposed by the time to market and limited resources for testing are two main problems in the testing process.

Meanwhile, it becomes imperative to distribute different parts of an application to different embedded systems with messages being exchanged among those nodes [4]. The embedded system under test is often treated as a black box by testers. One reason is that Original Equipment Manufacturers (OEMs) tend more and more to purchase components from outside suppliers, so testers do not have access to the source code. The other reason is that security requirements and the limited number of test cases are the focus of the testing process rather than the detailed implementation. Then the input and output of the embedded system, where sensor data and bus communication are two major sources, are investigated to verify the embedded system under test.

This thesis focuses on designing a co-simulation framework and implementing a prototype based on an actual industrial product. The co-simulation framework prototype should provide a comparable realistic environment for the ECU under test and have the ability to monitor the bus communication

behavior of the ECU network through which the ECU can be tested. The prototype is a proof-of-concept which gives solutions for further research in the testing process of the automotive industry. The implemented prototype in the thesis turns out that the framework can be used for ECU testing and multiple forms of ECUs implemented by different tools can be included in the framework.

This chapter presents a general introduction to the area that this thesis lies in, explains the context of the problem and goals this thesis should achieve. The research methodology and ethic considerations are shown as well. It also outlines the structure of the thesis.

1.1 Problem Statement

The demand on the embedded system testing increases dramatically in the automotive industry. The testing process is defined as ”the demonstration of consistency, completeness, and correctness” [5] of the system under test. One of the most popular testing methods in the automotive industry is to use Hardware-in-the-Loop (HiL) simulation which emulates some subsystems of the overall system by immersing faithful physical replicas within a closed-loop simulation of the remaining subsystems [6]. The same ECU that will be used in the final product is used so testing results are quite accurate. The HiL simulation system is also called HiL rig. Engineers always monitor the bus communication behavior inside the HiL rig in order to verify the ECU under test because the physical ECU, which is under test during the HiL simulation, is treated as a black box.

However, the cost to establish and maintain such a HiL rig is quite high because the electric and electronic systems of the rig are partially built following the one in the real vehicle in order to achieve the same functions and performance as the real one and ensure the accuracy of the rig. Whenever there are changes in the vehicle, the HiL rig should be adapted simultaneously which leads to the difficulty of maintaining the rig. As a result, a company could only keep a small number of HiL rigs at the same time which means that the existing rig resources are limited. When there comes a key release or update, multiple departments or engineers may compete for the same rig which will delay the time to market. At the same time, there exists another situation where a new software release for old products needs to be tested, but the corresponding rig has been removed due to considerations regarding maintenance budget and space occupancy. And the HiL rig is mainly used at the late stage of the project development life cycle.

So one vital demand is to have a framework which can be used in both early and late stages of the project development life cycle in the automotive industry. The new framework should keep a relatively low cost in both time and resources and be convenient to be distributed among testers. The framework should have the ability to monitor bus communication status as well. This thesis will investigate how to design such a framework and implement a prototype as a proof-of-concept. The scope of the thesis includes the design of the framework, choice of implementation tools as well as the comparison with the old platform.

1.2 Goals

The thesis aims to propose a co-simulation framework for the automotive industry. The goal is not to provide a final solution or product, but a start point for the future development of the framework, as a proof-of-concept.

The proposed framework should contain different forms of ECUs, including ECU model, virtual ECU, physical ECU and ECU behavior model. The model, software and hardware of an ECU are delivered sequentially in normal conditions but the maturity of different vehicular subsystems is not always synchronized. Some subsystems cannot even be abstracted in the form of a model, then behavior models are useful in this case, thus the framework should support multiple forms of ECUs. The bus communication behavior among different ECUs is investigated to test the ECU under test. One goal is to investigate on-the-shelf simulation tools and analyze their benefits and limitations. Below is a list of goals for the thesis.

- Design of a co-simulation framework
  - The framework should contain physical ECU loaded with real production code
  - The framework should contain virtual ECU loaded with real production code
  - The framework should emulate the real bus communication behavior for each Electronic Control Unit (ECU)
- Implementation of a co-simulation framework prototype
  - The prototype should achieve similar performance to the existing HiL rig in testing ECUs

- The prototype should be compatible with other popular simulation tools, such as Mathworks Simulink, National Instrument Labview
- The prototype should reduce the cost of time and resources in the testing process

The result of the thesis provides a framework where different forms of ECUs communicates with each other as in the real working scenarios.

1.3 Research Methods

In order to meet the goals of the thesis, existing HiL rigs at VCE are firstly explored. Literature study based on current testing technology and on-the-shelf tools is then carried out. The main research method used in this thesis is qualitative, as the proposed framework is generated through analysis based on current testing methods rather than numerical analysis.

The case study method [7] is chosen as the most appropriate research strategy for the thesis because the proposed co-simulation framework is going to be evaluated through a specific use case at VCE. Interviews with experienced engineers at VCE and suppliers of different simulation tools and documentation reviews of the vehicle system requirements and ECU specifications are used to narrow down the scope, as well as investigate more potential solutions in regard to the data collection. An extensive literature study, whose main topics are virtual ECU, AUTOSAR and testing technology,is performed in parallel.

As there is no hypothesis that needs to be proven at the beginning of the thesis, the research approach is inductive. The design of the proposed framework is based on findings that are received at different stages of research. The performance and efficiency of the proposed co-simulation framework in different specific use cases will be compared to those existing frameworks, benefits and limitations of different testing methods will be analyzed.

1.4 Ethics considerations and sustainability

The thesis is carried out at Volvo Construction Equipment in Eskilstuna, Sweden. No confidential information of the company is included in the thesis report. Since interviews were used for data collection, ethics need to be considered before holding interviews. It is important not to force interviewees to

answer questions. Also, the consent of the interviewee is required before any recordings.

The benefits of the proposed co-simulation framework are to reduce the cost of time and resources in the testing process and speed up the project development life cycle. In addition, the framework is able to be distributed to multiple engineers simultaneously allowing parallel work. If the framework can be put into real usage, the number of HiL rigs can be decreased which makes the testing process more sustainable.

1.5 Structure of the Thesis

The rest of the thesis report is structured as follows. Chapter 2 gives an introduction to the necessary theoretical background to help understand the problem and goals of the thesis. Chapter 3 presents the design of the proposed co-simulation framework. The design and implementation of the prototype are described in chapter 4 and 5. Then in chapter 6, the process of the evaluation of the framework prototype is shown. The last chapter, chapter 6, gives a summary of the thesis and discusses the completion level of goals and future work which can be extended based on current results.

Chapter 2

Background

This chapter aims to present the knowledge background of the thesis which is needed for increasing understanding of how this thesis was developed. Firstly this chapter will introduce what an ECU is and several significant theories in its area which will be later used in the thesis work. Following different testing technologies are presented. The next section describes the virtual ECU. Then the concept of V-model in project development is introduced which integrates the testing technologies into different stages of the project life cycle. This section is followed by a brief description of communication protocols used during the thesis. The Functional Mock-up Interface (FMI) standard is introduced additionally. Furthermore, industrial tools and their features are briefly presented. The last section of this chapter will introduce some related work to the thesis.

2.1 Electronic Control Unit

Electronic Control Unit (ECU) is an embedded system that controls one or more electrical system or subsystem in the vehicle [8]. From this definition, systems that contain microprocessors, microcontrollers, Digital Signal Processing (DSP) and Field Programming Gate Array (FPGA) are able to be recognized as ECUs. ECUs are usually distinguished according to their functions in the automotive industry, there are engine control unit, powertrain control unit, general control unit and so on.

The idea of ECU was firstly introduced in the automotive industry in the 1970s and ECUs played a fundamental role in the evolution of the vehicles from being completely mechanical to being an electronic dominant device [9]. The number of ECUs in the vehicle has continued increasing since the day it

was put into vehicles and they are affecting everything in the vehicle from energy consumption to safety features. In 2009, premier cars have more than 70 ECUs with 5 communication buses and up to 2500 signals which are used for inter-ECU communication [10].

Basically, one ECU is able to be divided into software and hardware and different functions of the ECU are deployed to the two parts separately. The ECU hardware will be connected with sensors and actuators while the ECU software controls the behavior of the ECU and processes the incoming and outgoing data. ECU software, which is also called embedded software, occupies an increasingly significant position over time and it will be discussed in the following content.

2.1.1 Embedded Software

Embedded software is the typical software that will be executed on devices, known as embedded systems and running on dedicated operating systems. One main reason that differs the embedded software with the regular PC software is the time, memory and speed limitations of embedded systems [11]. Meanwhile, the embedded software is not controlled through a graphical user interface anymore but directly through the machine.

The term ”real-time” is a widely used feature in the embedded software area. It means that the embedded system which runs the software has explicit deterministic or probabilistic timing requirements [12]. If the embedded system is required to run multiple tasks at the same time and each task requests a certain amount of resources, the embedded software should schedule all these tasks according to their deadlines in order to fulfill the system timing constraints. The real-time scheduling theory was originally made for cyclic tasks with hard deadlines. After years of development, there are five important areas of real-time scheduling theory, including fixed-priority scheduling, dynamic-priority scheduling, soft real-time applications [12].

Embedded software increases in line count, complexity, and sophistication [13] while the number of ECUs and functions waiting for implementation keeps growing. In order to manage the situation and keep the development costs feasible, a general standard needs to be set up.

2.1.2 AUTOSAR Classic Platform

AUTomotive Open System ARchitecture (AUTOSAR) is a worldwide development partnership of vehicle manufacturers, suppliers, service providers and

companies from the automotive electronics, semiconductor and software industry since 2003. The objective of the AUTOSAR is to create and establish a software architecture which is both standardized and open for the automotive ECU. Its goals contain the modularity, scalability, transferability, re-usability and many other topics through the entire life cycle of the project. [14][15]. In 2017, the AUTOSAR partnership added a new standard to its line-up which is based on POSIX operating systems [16]: the AUTOSAR Adaptive Platform, then the previous one is continued under the name AUTOSAR Classic Platform which is discussed in this thesis.

The AUTOSAR uses a software architecture which can be divided into three main layers [17]:

- Application layer: Encapsulate an application running on the AUTOSAR infrastructure
- Runtime Environment: Abstract from the network topology for the inter- and intra-ECU data exchange between different layers and within one layer
- Basic Software: Provide the infrastructure for execution on an ECU

The overview of the three layers can be seen in figure 2.1. It is worth mentioning that blocks that are under the runtime environment block and above the microcontroller block in the figure belong to the layer basic software.

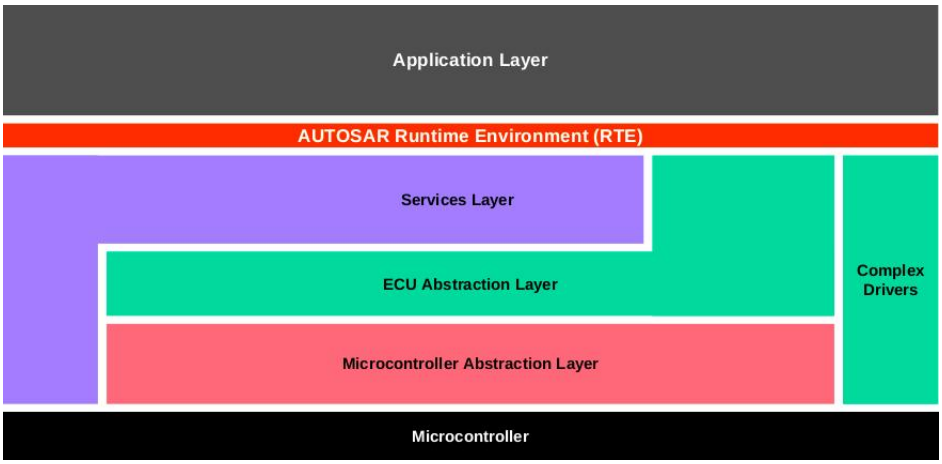

Figure 2.1: The AUTOSAR layered software architecture [14]

In order to implement a piece of embedded software following the AUTOSAR standard, three types of configuration description files (.arxml) are required.

The system configuration description includes all the system information and explains how basic software components are mapped to different ECUs. The ECU extract includes the information from the system configuration description for each ECU. The ECU configuration description contains the configuration of all basic software components for every single ECU.

2.2 Embedded system testing

Embedded system testing is similar to other testing types. One difference is that the embedded system testing is always related to software as well as hardware. The aim of testing is to detect errors in the system, and, if no errors are found during comprehensive testing, to convey confidence in the correct functioning of the system [18]. So the embedded system is tested for its performance, consistency and validated as per the requirements defined at the beginning of the project. Embedded system testing checks and ensures that the embedded system under test is of high quality and complies with all the requirements it should meet [19].

Embedded system test teams normally focus on black-box testing, which is also called functional testing, as they are often independent from development teams and have no easy access to precise design information [20]. In black-box testing, inputs and outputs of the system are focused without bothering about its internal knowledge. Black-box testing aids in overall functionality validation of the system and is done based on the requirements of the embedded system [21].

There is an important aspect of testing embedded systems, which is the need to provide observability of system behavior sufficient to allow engineers to detect failures [22]. Regarding ECUs in the automotive industry, an ECU takes a myriad of signals monitoring the vehicle as well as its environment, so appropriate test system resources emulate the various input signals in a controlled manner and load and check the outputs for correct response in ECU black-box testing [23]. Bus communication behavior is a major source of its inputs and outputs where time constraints, packet loss, content of message frames can be tested.

2.3 Testing Technology

At different stages of the ECU development life cycle, different resources become available in sequence and it is impossible to test the whole system after

all resources are well set up due to the high pressure on the product time to market and the complexity of the distributed automobile system. Then there come the requirements for the testing technologies used in different phases of the project life cycle.

This section presents concepts of Model-in-the-Loop, Hardware-in-the-Loop simulation and Software-in-the-Loop simulation. Those X-in-the-loop testing methods simulate part of the physical plants and embedded system architectures and allow engineers to test the ECU on the software and hardware level respectively [24].

2.3.1 Model-in-the-Loop Simulation

Model-in-the-Loop (MiL) simulation is used to test a model which is an abstract of a system or subsystem. The goal of MiL simulation is to test the architecture of the model and its functions in a hardware-independent manner [25], so there are no physical hardware components in the MiL simulation. With regard to the field of ECUs, ECUs will communicate with its surrounding environment and receive sensor data, as well as control actuators. MiL simulation is mainly used at the early stage of development where the ECU environment is still supposed to be simulated in order to work properly.

2.3.2 Software-in-the-Loop Simulation

In Software-in-the-Loop (SiL) simulation, the actual production code is incorporated into the mathematical simulation containing the models of the physical subsystems [26]. Compared to the HiL simulation that will be presented later in section 2.3.3, the SiL simulation happens at an earlier stage. The implementation of hardware is usually later than the implementation of the production software, this is the reason why requirements for SiL simulation rose up. At the same time, some control algorithms may even not have an existing model, then SiL simulation brings up the possibility of testing them directly.

The architecture of the SiL simulation is shown in figure 2.2. The overall simulation will only happen on a host PC which is a ”pure-virtual” environment and no I/O ports are required.

2.3.3 Hardware-in-the-Loop Simulation

Hardware-in-the-Loop (HiL) simulation is a real-time simulation methodology where real subsystems of a complex machine are coupled together with

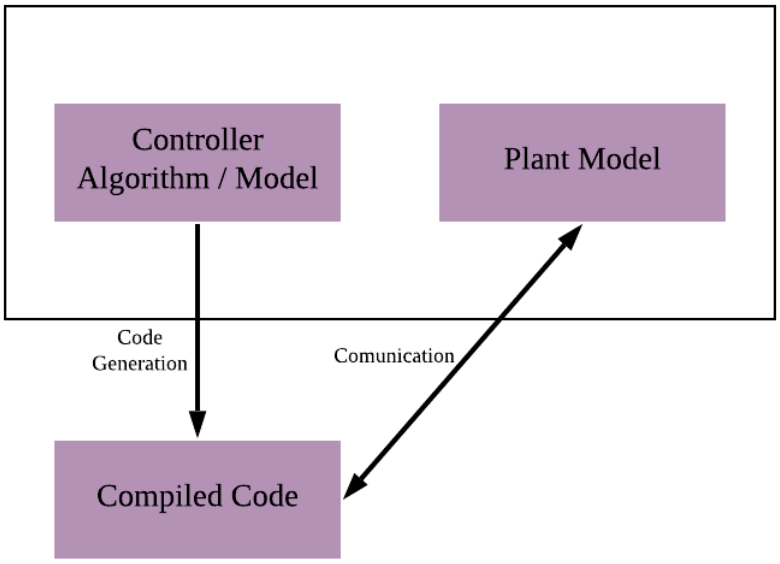

Figure 2.2: SiL simulation

the simulated models of the remaining subsystems to form its complete representation [27]. In this case, some difficult-to-model subsystems are able to be kept in the simulation environment while the rest of the system are modeled by other tools.

HiL test rigs are widely used in the vehicle industry as rigs are regarded as accounted real operational conditions when compared with the traditional pure software-based simulation [28]. The structure of the HiL rig is shown in figure 2.3. The stimuli in the figure can be predefined Simulink or Labview signals simulating I/Os and part of the bus message frames. And the simulated subsystems are actual ECUs loading the production code, as well as mathematical models.

The features of the HiL rig are summarized as follows [28],

- Single HiL platform enabling parallel inclusion of multiple subsystems with actual full scale hardware components
- Availability of injecting sensor data and controlling actuators
- Implementation of in-vehicle independent communication network protocols
- Interfaces for simulation tools generating signals

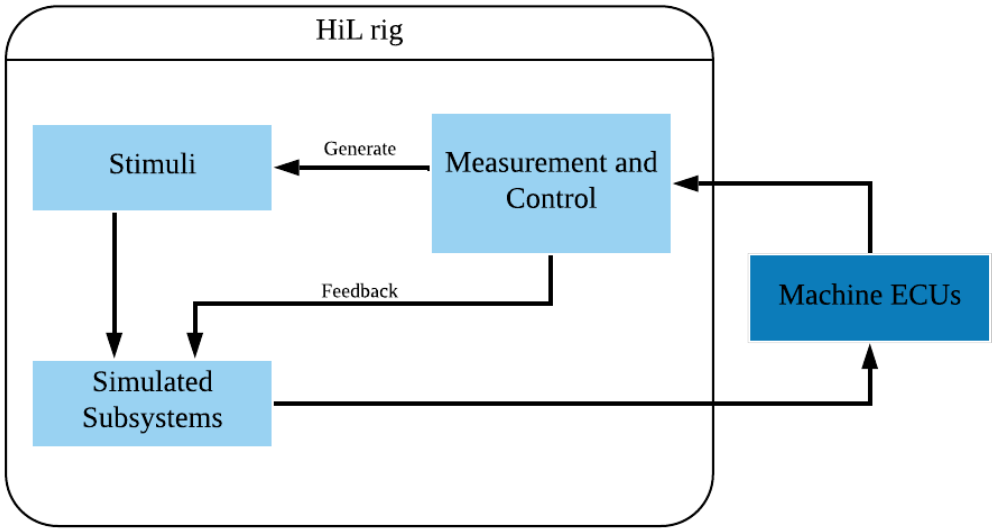

Figure 2.3: Architecture of HiL test rig

2.4 Virtual ECU

The term virtual ECU in this thesis is regarded as any software functionality in the ECU that can be executed without the existence of ECU hardware [29]. One key feature of the virtual ECU is containing unmodified production code which is identical to the final ECU. It can be recognized as an advanced SiL simulation through which the production code is tested in a more realistic environment. In addition to control algorithms, interface compatibility, missing data, AUTOSAR compliance of the code can also be tested.

The virtual ECU will repeat the same realistic scheduling behavior while the same load and outside stimulus are put on it as the physical ECU. And the virtual ECU can be connected at the bus level which brings the integration testing early on. The advantages of the virtual ECU could be summarized as [30],

- Decoupling and parallelizing large development work across different teams or even organizations
- Duplicating and backing up a specific developed snapshot simply with a lower budget

- Validating separate changes to the software without having an impact on the independent changes made by other developers
- Debugging the embedded software in a realistic environment

In this case, virtual ECUs provide the possibility of testing on the embedded software at an earlier stage of the project life cycle rather than waiting for the implementation of ECU hardware.

2.5 Validation & Verification model

The Validation & Verification model (V-model) [31] was originally proposed as a modified version of the Waterfall model in the 1980s and is still widely used in project development today. The workflow of a project following V-model is presented in figure 2.4, where the relationship between development activities and verification activities is clearly reflected.

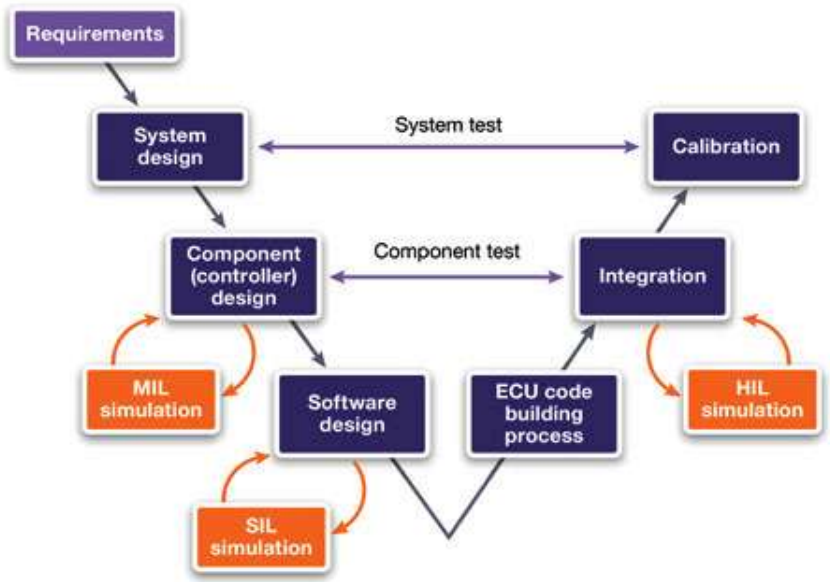

Figure 2.4: V-model in ECU development [32]

This development process in the V-model is balanced and greatly relies on the verification from the previous steps before proceeding forward [33]. Developers and testers work parallel which is a feature that differs the V-model from its previous development methodology. Before the project moves forward, the product from every phase needs to be tested and the verification methods are technologies mentioned in section 2.3.

Although requirements of the project can be changed at any stage, the requirements documents and the test documents should be updated at the same time which enlarge the project maintenance costs.

2.6 Communication protocols

Automotive systems are complex distributed systems with increasing demands on the system network capabilities because of the system heterogeneity and increasing number of applications relying on the network. This section will not present all popular protocols used in industry today, but ones used in the thesis.

2.6.1 Control Area Network

The Control Area Network (CAN) communication protocol is the most common protocol in the automotive industry since the 1990s and was developed by Robert Bosch GmbH as a multi-master, message broadcast, collision detection bus system [34]. Through the CAN bus communication, different electrical devices and environments are able to talk to each other. Both ”High Speed” (up to 1Mbit/s) and ”Low Speed” (40 to 125kbit/s) CAN buses are described in International Organization for Standardization (ISO) standards ISO 11898 and ISO 11519. The bit fields of standard CAN messages are shown in figure 2.5. And it is worth mention that in the ISO11898, the CAN identifier is extended to 29 bits while the standard CAN uses the 11-bit identifier.

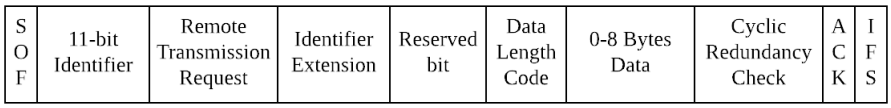

Figure 2.5: Bit fields of standard CAN messages

The priority of each CAN message is determined by its identifier which can be seen in figure 2.5. The lower the message identifier number, the higher its priority. The CAN allows only one node to send messages at one time, and each node will continuously monitor its own transmissions. So if two nodes attempt to send messages simultaneously, the message with the higher identifier owns the bus and the other node will attempt to transmit its messages again once the bus is released by the node which is currently sending messages.

There are four types of frames that can be transmitted on the CAN protocol: the data frame, the error frame, the remote frame and the overload frame. Within the four types of message frames, only the data frames are used to transport messages, the other three types of frames are used for fault detection, message request and synchronization.

Nodes in the CAN network are connected to each other through a twisted pair with a characteristic impedance of 120 Ω. The twisted pair of cables greatly reduces the noise and provides a higher speed communication. Although some other wire cable construction techniques offer similar functionality as well, like shielded cable, the twisted pair has a lower manufacturing cost.

2.6.2 Unified Diagnostic Services

Diagnostic systems in the vehicle automotive industry are developed to detect the faults in the vehicles and reprogram ECUs if required. Unified Diagnostic Services (UDS) protocol is a popular diagnostic communication protocol in the vehicle networks which is standardized in the ISO 14229-1 [35]. The reason why it is called unified is that the protocol can be combined with the other standards, like Keyword Protocol 2000 and be an international standard rather than a company-specific standard.

The UDS protocol is implemented on the application layer of the Open System Interconnection (OSI) reference model because ISO 14229 was set up with the purpose of specifying data link independent requirements for diagnostic systems [35]. It regulates the generic services which allow the diagnostic tester, who acts as a client, to control the non-diagnostic message transmission on the data link, below are three main diagnostic architectures:

- Servers connect to an internal data link and indirectly connect to the diagnostic data link through a gateway
- Servers directly connect to the diagnostic data link
- Servers directly connect to the diagnostic data link through a gateway

The UDS request is indicated by its service identifier and the response service identifier is specified as the request service identifier plus hex value 0x40. The common UDS service identifiers are listed in table 2.1.

Table 2.1: Common Service ID in UDS [35]

Request SID	Response SID	Service
0x10	0x50	Diagnostic Session Control
0x11	0x51	ECU Reset
0x28	0x68	Communication Control
0x22	0x62	Read Data by Identifier
0x23	0x63	Read Memory by Address

2.7 Functional Mock-up Interface

Functional Mock-up Interface (FMI) is a tool independent standard to support both model exchange and co-simulation of dynamic models for complex cyber-physical system development [36]. The first version of FMI was posted in 2010, followed by the second version in 2014. FMI has already been supported by more than 100 tools and is widely used in the area of automotive industry. It helps engineers integrate component models from different suppliers and simulation tools.

For highly heterogeneous systems, such as vehicles nowadays, multi-domain simulation is required to connect the heterogeneous subsystems inside the vehicle so that an extensive vehicle simulation can be achieved. The intentions for FMI are to utilize the blocks of one model in one another simulation and modeling environment, and to couple two or more simulation and modeling tools in a co-simulation environment [37]. The thesis uses the co-simulation perspective of FMI and its working theory is shown in figure 2.6.

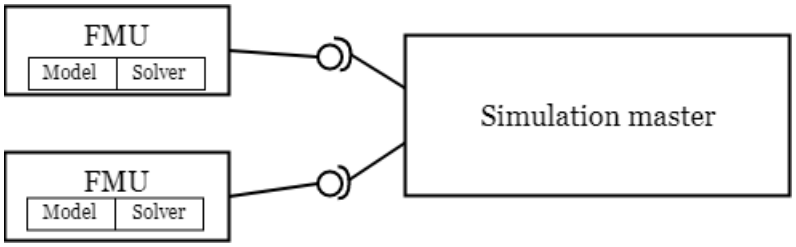

Figure 2.6: FMI Co-simulation

Functional Mock-up Unit (FMU) in the figure 2.6 is used to build a connection between different tools. It is generated and exported by one tool then imported into the simulation master so that several tools cooperate at runtime. Instead of the simulation master using its own solver to interpret models from

other tools in the model exchange of FMI, the co-simulation mechanism allows different tools to use their own solver.

2.8 CANoe

CANoe is a comprehensive software tool developed by Vector Infomatik GmbH for development, test and analysis of individual ECUs and entire ECU networks [38]. Both developers and testers are supported throughout the project development process by the tool. It provides the possibility to simulate the behavior of ECUs in the entire network and output the analysis of bus statistics, oscilloscope view of network data variables, as well as message tracing status.

CANoe uses databases to store all signals that will transfer through the CAN network. With the usage of database, all engineers involved in the different project phases are able to reach a high level of data consistency and the compatibility of the developed systems increases dramatically.

ECU models in CANoe are behavior models emulating the transmission behavior of each ECU. The communication model of CANoe is shown in figure 2.7.

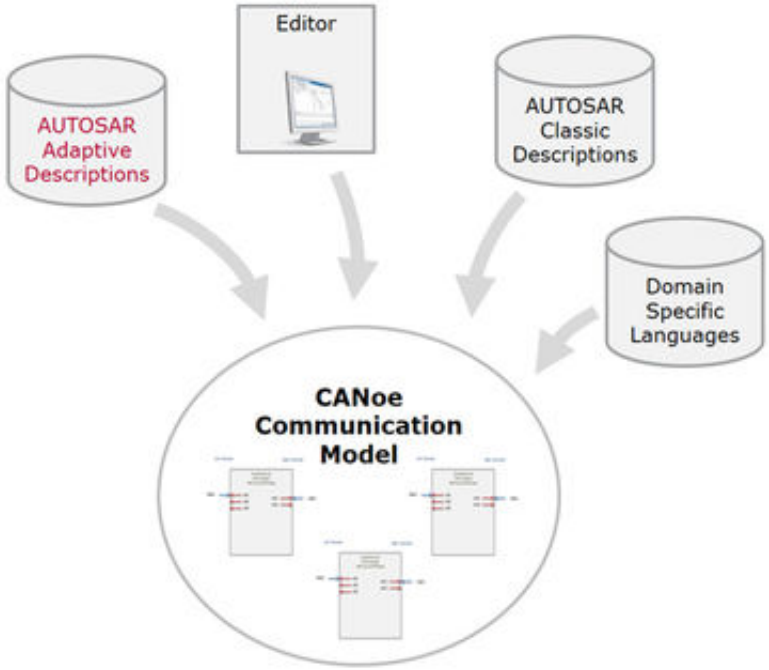

Figure 2.7: Communication model of CANoe [38]

2.8.1 CANdb++ database

At least one database is required in a CANoe project and the following implementation will be based on the database the project connects to. Based on the mapped CAN bus database, ECU nodes are inserted into the network while ECU nodes that are not included in the database are not allowed to be added into the CANoe project. The CAN database lists all the signals and their layouts that are consistent with the actual machine at work, as well as maps those signals to their corresponding message frames then maps the message frames to their ECU transmitters and receivers. One signal can only have one transmitter but multiple receivers. The structure of the database is shown in figure 2.8.

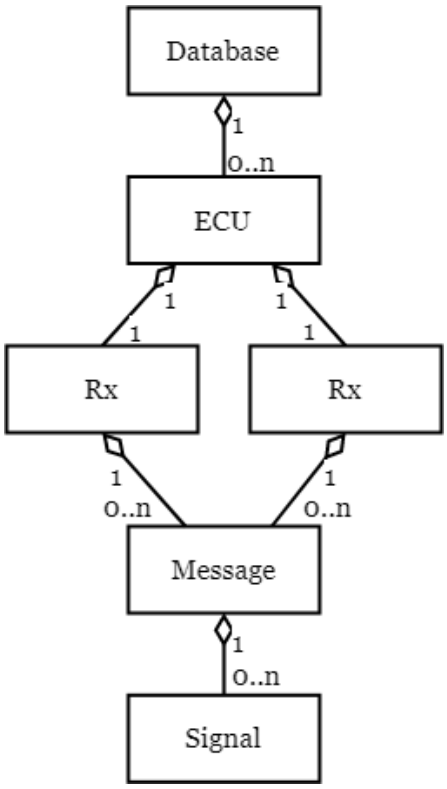

Figure 2.8: Structure of the CAN database in CANoe

2.8.2 CAPL

The Communication Access Programming Language (CAPL) is the programming language used exclusively within CANoe, the original design purposes

of CAPL are to control all test and requirement operations, control system or module simulation, record event and messages and control the playback of the whole communication process [39, Chapter 1].

CAPL can be recognized as an event-driven programming language, but also can be thought of a programming environment. The CAPL program provides the capabilities of modifying a wide variety of inputs, outputs, creating start-stop events, keyboard entry events, timer events, as well as sending and receiving CAN messages [40]. In the real CANoe application, each ECU has its own CAPL script to manipulate the behavior of the node.

Additionally, CAPL is based on the C programming language, users, who have previous experience with C/C++ programming languages, will not spend too much time on programming orientation, mechanics but the difference on the syntax, operators and statements [39].

2.9 DSpace toolchain

DSpace GmbH, located in Germany, is one of the world leading providers of tools for development and testing ECUs. It provides a complete toolchain for validation covering every aspect from generating a virtual ECU, integrating the virtual ECU with other virtual ECUs, building an overall system including environment models and simulating it [41].

Here is an example of ECU validation using dSpace tools. VEOS is a PC-based simulation platform which supports multiple automotive standards and model import. ControlDesk can connect to the application in VEOS and make experiments on it without model modifications. If physical ECU components need to be connected, ConfigurationDesk replaces the role of VEOS to configure the hardware interface and the connections with other models. Functions of tools are also listed in table 2.2.

The toolchain from dSpace provides a high complexity and clear logic to engineers with great software and hardware extensions. On the other side, the example only contains tools for ECU validation and there is a set of tools for ECU development. Too many tools are included in the project life cycle so that the toolchain is complicated and hard to learn.

2.10 ETAS ISOLAR EVE

ETAS ISOLAR-EVE, which is also called ETAS virtual ECU, is a tool environment that serves for testing production ECU software in a virtual environ-

Table 2.2: Functions of tools in dSpace toolchain

Tools	**Function**
VEOS	A platform for pc-based offline simulation
ConfigurationDesk	Configuration and implementation for dSpace hardware interface
ControlDesk	universal experiment software
TargetLink	Production code generation for ECUs, automatically from Simulink
SystemDesk	Generation of virtual ECUs for validation and verification

ment on the host PC [42]. It has similar functions as SystemDesk which is a member of the dSpace toolchain.

The goal of the tool is to help users test their AUTOSAR software, for example, ECU production code on host PCs. The generation of the virtual ECU requires those AuTOSAR configuration files that are mentioned in section 2.1.2 and the production C code imported into the tool itself, where those files are configured for being able to execute on the host PC. The tool will replace the operating system and microcontroller abstraction layer which are hardware-related components in the AUTOSAR software architecture.

The tool also provides an interface for connecting the implemented virtual ECUs to the bus simulation tools, such as CANoe and BUSMASTER, which is one of the most significant advantages compared to SystemDesk in dSpace toolchain. Additionally, the entire ISOLAR EVE environment is built on the basis of AUTOSAR 4 and it does not support any earlier version.

2.11 Related Work

There are several references using simulation environments for ECU testing. Lanigan et al. [43] used the CANoe simulation environment to develop a fault-injection framework with example fault-injection scenarios. The fault-injection framework is totally in the CANoe environment and was proved to be used to exercise AUTOSAR error-handling mechanisms. However, its prerequisite requirement is software code that follows the AUTOSAR standard and it does not consider working with other simulation tools. In addition, in the current development process, it is hard to reuse the embedded software which is already developed for an ECU [44]. Franco et al. [45] developed a virtual validation based on the AUTOSAR architecture with a virtual ECU using the

dSpace toolchain and Simulink. They achieved the communication inside the same virtual ECU and between two different virtual ECUs. The connection between the Simulink model and the virtual ECU was built by the function module inside the dSpace toolchain instead of a common scalable protocol. Based on this, Shan et al. [46] designed an AUTOSAR system solution by using Simulink, ETAS ISOLAR-A and RTA-OS which proved to be reliable. However, the new solution still heavily relied on the customized interfaces between tools.

Simulation and models that are available at different stages of the project development life cycle have been adopted at different levels of abstractions and multiple areas in the automotive design to optimize it from different perspectives. But it would primarily cover the functional design and testing of an ECU with use of simulation tools and be typical focused on a single ECU [47][48]. Phatak et al. [49] built a virtual multi-ECU high fidelity Cyber-Physical multi-rate co-simulation that resembles a realistic hardware based automotive embedded system. Their work paid attention on the integrated testing and issues related to the hardware.

The above work used one or two simulation environments at the same time for ECU testing, but they lacked flexibility and scalability because the functionality of the work relied on the selected simulation environment and the hardware access is not considered. Meanwhile, Simulating many ECUs on a single PC highly influences the bus communication thus biasing the results for performance and latency tests [50], this is also a point to consider. Tools presented in previous sections, for example, CANoe, dSpace toolchain, provide strong bus simulation, ECU emulation, Hardware-in-the-Loop simulation and other functions, and each tool has its own advantages and limitations. So this thesis attempts to combine multiple simulation environments together in order to improve the flexibility and efficiency of the testing process.

As discussed above, modern vehicles are very distributed systems in order to tackle both complexity and heterogeneity. The testing process can be done modularly by co-simulation as well. Al-Hammouri [51] built a comprehensive co-simulation platform for Cyber-Physical systems where the interface between simulation tools is programmed manually. Suzuki et al. [52] implemented a co-simulation framework realized by coordinating Simulink and OMNeT++. They noticed the problem of different simulators operating under different time managements, but the interface between the two simulators still needs to be manually programmed. Awad et al. [53] designed a co-simulation framework based on two main simulators, OMNeT++ and OpenDSS and verified the framework with a case study.

The co-simulation is helpful for the design and testing processes of distributed systems. Then FMI is considered as an important medium to connect and synchronize different simulation environments in the thesis. FMI is more general and supported by most mainstream simulation tools, so no additional interface needs to be made. Some related work using FMI for co-simulation already exists. Thiele and Henriksson [54] used the FMI for the purpose of model exchange in the context of AUTOSAR software component development. They exported a FMU which is implemented from a Modelica controller, converted it to an AUTOSAR software component, and then imported the component to an AUTOSAR tool. Their work used FMI as an intermediate format but the FMI to AUTOSAR mapping process was found to have missing features. Exel et al. [55] illustrated how a FMU can be coupled to widely-used tools and utilized FMUs by coupling them to a simulation framework. They used the SIMIT framework as an example, and the FMI was proven to offer a promising performance that utilizes existing models together with their solving algorithms. There also exist an open source Python package PyFMI [56] for loading and interacting with FMUs. It contains a master algorithm for simulation of coupled FMUs together with connections to other Python packages. Böhme and Réhault [57] used PyFMI to combine models in Modelica and a Python environment together and considered their work can be used for further applications for online and offline fault detection, optimization and predictive control.

Chapter 3

Design

This chapter presents the overall design of the co-simulation framework for in-vehicle ECUs which is one of the major goals of the thesis. The chapter starts by describing different forms of ECUs used in the framework in terms of what they are and why they are required. The next section presents the overall design of the co-simulation framework.

3.1 Forms of ECUs

Multiple forms of ECUs are used in the thesis. Each form of ECU is described in detail in this section. Reasons why they are included in the design of the cos-simulation framework are also discussed.

3.1.1 ECU Model

There are two kinds of models and one is the control model, representing the target ECU. The control model contains the algorithms, functions and logic components of the embedded software using different blocks. The other one is the plant model, representing the physical features of the remaining system. The plant model does not need to have all features of the remaining system but part of functions is enough for the testing purpose, this also reduces the complexity of test. The relationship between the control model and the plant model is presented in figure 3.1.

In the case of the co-simulation framework for in-vehicle ECUs, models can be used for the purpose of either control model or plant model. An ECU model can simulate an ECU node in the vehicle and generate signals the simulated ECU node has, it also can be set as a part of the entire plant model to

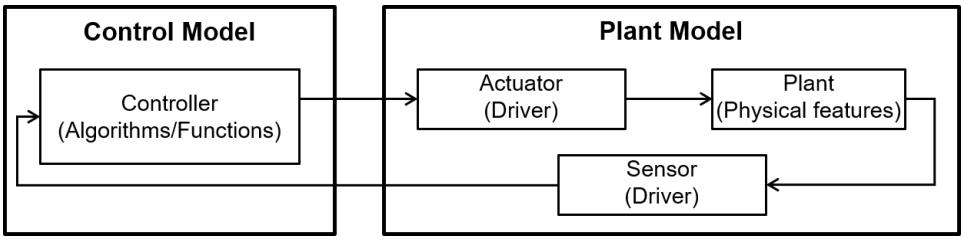

Figure 3.1: Relationship between the control model and the plant model

provide some signals from the rest of the framework.

3.1.2 Virtual ECU

Virtual ECU is the representative of the ECU software, it attempts to simulate the ECU without its hardware implementation and generate ECU behavior based on the production code. The layered structure of the AUTOSAR standard makes the ECU more modular and helps virtual ECU generation, thus most of on-the-shelf virtual ECU generation tools use AUTOSAR as the structure of their virtual ECUs.

The virtualization of ECUs has different levels of abstraction and it depends on their usage. The engineer can only provide application software and the rest of the software architecture will be handled by the virtual ECU generation tool, or import application software together with selected parts of basic software components. The last and the most complete level is that the entire application software and basic software components are given by the user.

The production code belongs to the application layer in figure 2.1 and the rest of software architecture can be customized or use the general software components provided by the tool itself.

3.1.3 Physical ECU

While the previous two forms of ECU allow for testing control algorithms and software implementation of the ECU, they cannot solve the problems arising from the actual implementation of the target ECU, for example, whether the power supply is sufficient to execute the whole system, whether the I/O latency has an influence on the system performance. These problems can be analyzed and tested easily when the physical ECU is included in the testing process.

The physical ECU is treated as a black box in many cases of the testing process when it is selected as the ECU under test and it means that the tester does not know the entire logic behind the ECU instead of its working process and

response to specific input. The framework in the thesis focuses on observing the bus communication behavior of the ECU network and the input and output bus data streams of in-vehicle ECUs can greatly help engineers test functions of the physical ECU.

If there is no physical ECUs in the co-simulation framework, it can run as fast as possible. When the physical ECU is connected to the communication bus through the hardware interface and interacts with the rest of the framework, the framework should be executed in real time.

3.1.4 Behavior model

The ECU behavior model is a script that describes some specific behavior of the ECU. It contains but is not limited to responding to specific messages and sending periodical messages. It is a comprise when none of the three forms of ECUs above is available but is convenient for some simple scenarios. However, behavior models have the lowest cost to build among all four forms of ECUs. For the test scenarios that do not have high requirements on the accuracy of the system, the behavior model is a good choice. The environment of the behavior model is shown in figure 3.2.

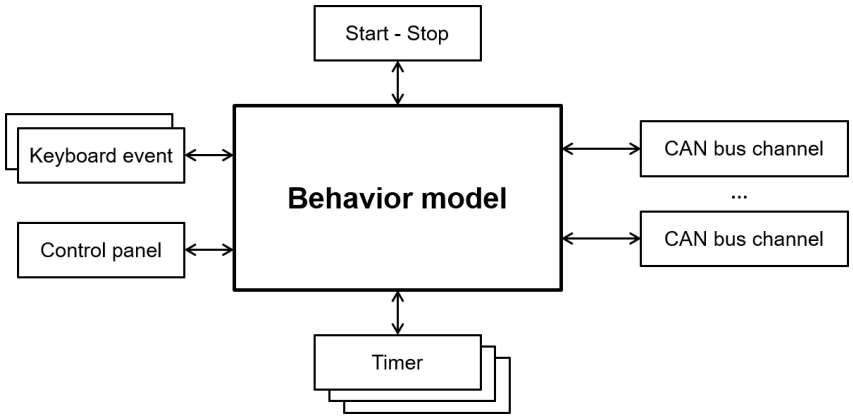

Figure 3.2: Environment of the behavior model

The behavior model can interact with multiple inputs, outputs and functions, such as start-stop events, timers, keyboard events, transmitting and receiving CAN messages. With the help of the behavior model, some events in the co-simulation framework can be manually triggered by the engineer which also brings a lot of convenience.

3.2 Choice of the communication bus

The communication bus chosen in the thesis is the CAN bus and it is introduced in section 2.6.1. CAN bus is one of the most common communication buses used in the automotive industry and all tools that are investigated during the thesis support CAN bus. CAN bus is not the only option for the communication bus of the framework and there are other alternatives of the communication bus, for example, Local Interconnect Network (LIN), FlexRay, which are popular communication buses in the automotive industry as well.

However, the framework prototype is not intended to be a final product that is going to be put into the real industrial usage but a proof-of-concept to integrate different forms of ECUs from different simulation tools into one environment. The bus communication behavior of the ECU network and ECU testing are focused in the thesis and the type of the bus has no impact on the final results.

The advantage of the CAN bus is that the cost to build is fairly low and it provides the ability to work in the different electrical environment. Also, the connection to the CAN bus is not hard to set up so that the difficulty of configuring the prototype can be decreased.

Other alternatives, to use FlexRay or LIN, have some drawbacks. FlexRay is faster and more reliable than the CAN bus but the cost to build it is much higher than CAN systems. LIN has lower bandwidth and a less effective bus access scheme with the master-slave configuration. After weighing advantages and limitations of different bus systems, the CAN bus is set as the communication bus in the framework.

3.3 Overall design of the co-simulation framework for in-vehicle ECUs

This section describes the overall design of the co-simulation framework. The design of the framework contains ECU models, virtual ECUs, physical ECUs and ECU behavior models and it also presents how different forms of ECUs are integrated into the framework for in-vehicle ECUs. ECU nodes in the framework are connected through CAN bus. The overall design of the co-simulation framework is shown in figure 3.3.

Different forms of ECUs are implemented by different tools so that they access the CAN bus in different ways. The virtual CAN bus and behavior models are provided by Vector CANoe, and there is no additional component

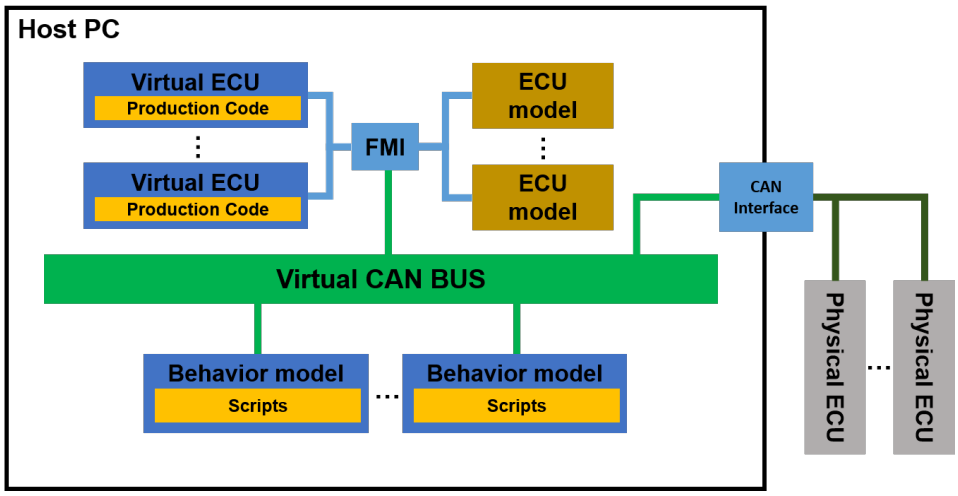

Figure 3.3: Overall design of the co-simulation framework

that is needed for the connection between the virtual CAN bus and behavior models. ECU models are made by Mathworks Simulink. Virtual ECUs are generated through SystemDesk inside the dSpace toolchain and then imported into VEOS which is another member of the dSpace toolchain and responsible for the connection with other tools. Simulink and the dSpace toolchain are two totally different environments for CANoe, and models or components in the three different environments above cannot be directly connected to each other and simulated at the same time.

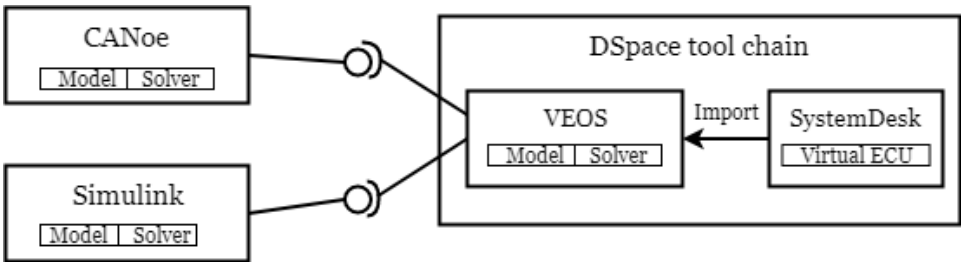

Figure 3.4: Relationship between various tools

In order to achieve the target of co-simulation, FMI is introduced to couple those environments together. Figure 3.4 shows the relationship between the three tools. Simulink and CANoe implement their own FMUs and the two FMUs are imported into the dSpace toolchain. The reason why the dSpace toolchain acts as the co-simulation master is that the dSpace toolchain has no ability to implement a FMU but is able to import FMUs implemented by

other tools. Additionally, both CANoe and Simulink support FMU export and import. During the process of co-simulation, different environments exchange data at discrete communication points and they independently use their own solvers in the time between two data exchange points. The dSpace toolchain, which is the co-simulation master, controls the data exchange between the two FMUs and synchronizes the solvers of CANoe and Simulink.

As the ECU under test is treated as a black box, the properties the framework can test are summarized below.

- Timing of messages
- Content of messages
- Packet loss
- Bus load

The timing and content of messages are visualized directly on the track windows of different tools. Additionally, there are two transmission modes, one based on time and one based on user request [58]. The two transmission modes are mapped to periodically messages and event-triggered messages. Packet loss occurs when one or more packets of data travelling across the network fail to reach their destination. This can be observed when the desired messages do not appear on the bus. Bus load is the total message transfer time in per time unit which can be obtained by calculation after the simulation is completed. Additionally, it would be better if the framework can have the “real-time” property which means that it synchronizes and communicates with the physical ECUs included in the framework in a more accurate way, but the co-simulation framework does not have this property because of multiple simulation tools working at the same time.

The overall design attempts to include different forms of ECU implementations in one co-simulation framework and provide a comparable realistic environment for the ECU under test. Although different forms of ECUs are implemented in different environments, they can work and cooperate with each other at the same time with the help of FMI, which also allows more future possible environments to access the co-simulation framework. The design does not have a limit to the number of ECUs from the perspective of the framework if the host PC has enough computing performance. A detailed case study of the framework will be presented in the next chapter.

Chapter 4

Prototype design

This chapter shows a case study of the design of the co-simulation framework. A framework prototype which originates from a product wheel loader at VCE will be implemented. The chapter presents the design of the prototype on the basis of the framework design in section 3.3.

4.1 ECU under test

Telematic Control Unit (TCU) is selected as the ECU under test in the co-simulation framework prototype. Testing the TCU is one of the major duties of the department to which the thesis belongs at VCE. The physical TCU is purchased from a third-party supplier outside the company and the department has no authority to get access to its source code, thus the TCU is treated as a black box and the goal is to see whether the prototype can be applied to test the TCU. If the prototype works, the efficiency of the department can benefit a lot from this work. Following is a brief introduction to the TCU.

The term telematics originally refers to any system by which an electronic or mechanical device can communicate with other devices or with users over a network [59]. As time goes on, the term telematics comes to mean the specific usage of on-vehicle communication and monitoring capabilities in automobiles.

The TCU in the automotive industry is responsible for controlling the tracking of the vehicle and monitoring the vehicle status. The TCU listens to communication between other subsystems in the vehicle, as well as provides an interface between the vehicle and external services through which it can be considered as a diagnostic tool. One regular TCU consists of [60]:

- Global Navigation Satellite System (GNSS) unit: Track the latitude and longitude of the vehicle
- CAN bus module: Transceiver and manage the CAN data communication
- External interface for mobile communication: Send the tracked values to the remote server
- Central processing unit: Process the information and act on the interface between Global Position System (GPS)
- Memory: Store GPS values in case of entering mobile-free zones and part of sensor data

The workflow of the TCU is presented in figure 4.1. The TCU extracts the required CAN messages from the vehicle CAN bus and communicates with the remote server in which case the TCU updates the current vehicle data enabling the remote vehicle status monitoring.

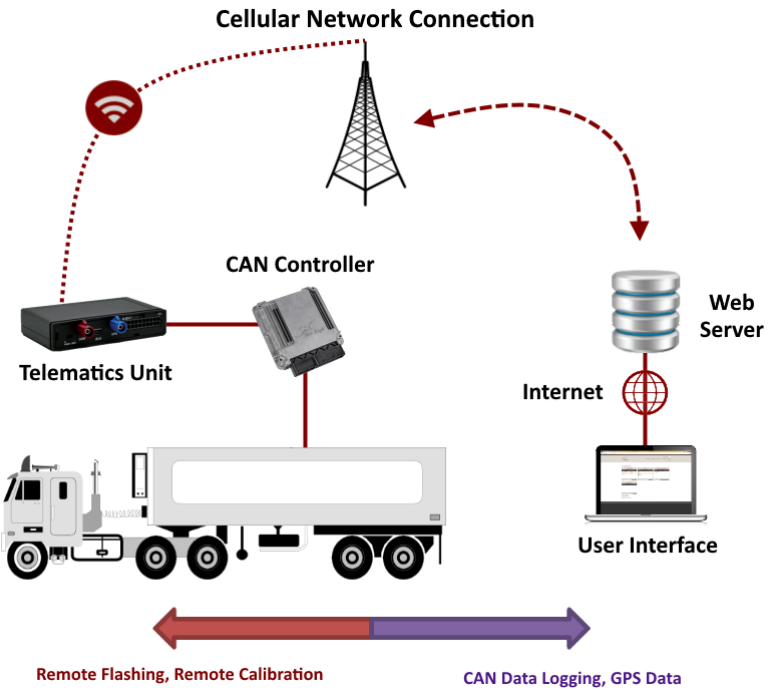

Figure 4.1: Workflow of TCU [61]

In the case of the prototype implemented in this thesis, the TCU sends data through 3G network and the data is able to be checked at the Volvo careTrack

portal which is corresponding to the user interface in figure 4.1. The platform can display available ECU nodes in the vehicle, vehicle working status and its position.

4.2 Selection of other ECUs

The prototype topology is based on that of an actual product wheel loader at VCE containing double 500 kHz CAN channels and more than 10 ECU nodes with up to 400 CAN signals. The CAN message database used for the prototype implementation is born out of the same one used in both the actual product wheel loader and the existing HiL rig for the wheel loader. As an initial prototype of the framework, it would be too complex to implement the entire in-vehicle CAN network and some parts of the original ECU network is unnecessary for the ECU under test.

Given that the ECU under test has already been set as the TCU, ECUs that have no direct data communication with the TCU can be removed, then 6 nodes including the TCU are reserved in the prototype in order to achieve goals of the thesis. Additionally, the TCU is connected to only one CAN bus channel in the actual wheel loader, so the other channel can also be removed which reduces the complexity of implementation work.

The principle of choosing implementation forms of ECUs is based on the maturity of the ECU and its importance to the ECU under test. The maturity of behavior models, Simulink models, virtual ECUs and physical ECUs increases in turn and the cost also rises in turn. The prototype chooses the most mature ECU forms available within the budget, and nodes that have more frequent bus communication with the ECU under test have a higher priority in selecting implementation forms.

The other 5 non-real ECU nodes are chosen to be implemented in three different forms, including a virtual ECU, a Simulink model and 3 behavior models, in order to see the availability of the co-simulation framework. The Human Machine Interface Control Unit (HMICU), which contains the largest number of input and output signals in the system, is implemented as a virtual ECU. There are three another General Purpose Machine Control Units (GPMCUs) implemented by behavior models because they only have limited communication with the TCU. The last node is only responsible for one signal so that it is implemented by a Simulink model.

4.3 Topology of the prototype

The topology of the co-simulation framework prototype is presented in figure 4.2. The framework prototype provides a simplified architecture of the electronic system inside a real vehicle specially for the TCU, which is the ECU under test in the case of the prototype. With the implementation and evaluation of the prototype, the proposed co-simulation framework can be evaluated.

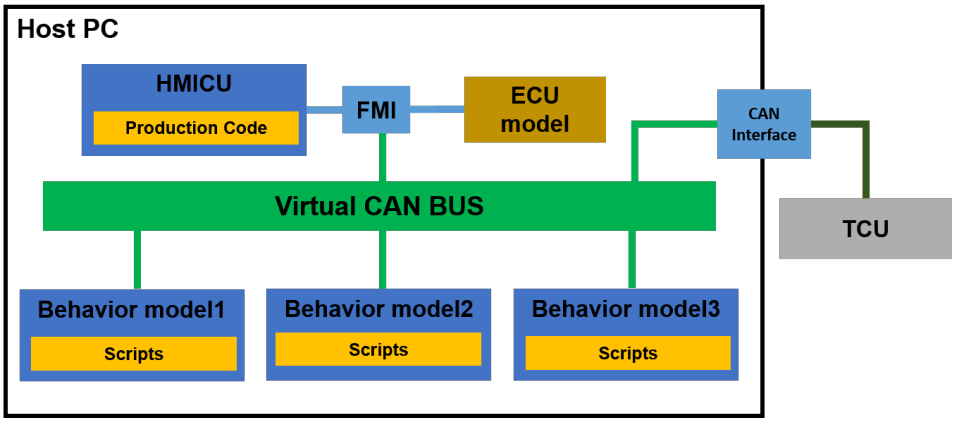

Figure 4.2: Overview of the architecture of the co-simulation framework

The prototype contains all mentioned forms of ECUs in the thesis and they are connected to the same framework and works simultaneously with the help of FMI. The implementation tools and connection methods described in section 3.3 are used in the prototype implementation.

4.4 Functions of the prototype

Limited functions are implemented in the prototype because of limited time and knowledge during the thesis, so not all functions of the TCU but a set of selected functions are supported. The selection of functions for the prototype is based on the functionality of the ECU under test and its bus communication behavior. The following prototype implementation follows prototype functions defined below.

ECU nodes send machine data to the TCU periodically, for example, machine time, fuel level and engine status. Each node is responsible for sending the machine data it should send in the real machine.

At each start up, the TCU sends a request to the whole CAN network for checking available ECU nodes, then the rest of non-real nodes respond to it

with pre-defined format and parameters. Once the TCU realizes the status of the in-vehicle ECUs, it generates a data set which is called machine network map.

According to the functions above, nodes other than the TCU require two event-triggered CAN messages for receiving the machine network request and sending the response. The machine data update function requires three more periodic messages which are shown in table 4.1. The senders of the messages are HMICU, HMICU and GPMECU1 and their receivers are all the TCU.

The length of all CAN messages in the thesis are 8 bytes and the period of the three periodic messages are 100 milliseconds.

Table 4.1: Messages for machine data update

messages	**Signals**	**functions**
HMICU_01	totalVehicleTime	Report current vehicle time
HMICU_02	FuelLvl	Report current fuel level in the tank
	EngineStatus	Report current engine status
BM1_01	EngineSpeed	Report current engine speed

All in all, the prototype functions can be divided into two major parts, machine network map and machine data update, and their requirements are shown in table 4.2. The next prototype implementation should achieve the two functions and follow their requirements.

Table 4.2: Prototype functions and their requirements

Functions	**Requirements**
Machine network map	The TCU can send requests to nodes in the ECU network with the right format and content
	Nodes other than the TCU can respond to the request with the right format and content
	The machine network map can be correctly updated to the Volvo careTrack platform
Machine data update	Periodic messages can be monitored on the CAN bus
	Values of periodic messages are simulated and always changed

The TCU has the capability of communicating with the remote Volvo careTrack portal through cellular network and data it sends can be accessed inside

the department. So if the fist function works, the machine network map should be seen at the remote portal. Regarding the second function, bus communication behavior can be collected both in the CANoe environment and the dSpace toolchain, same message values should be seen in the two tracking windows of the two different environments.

Chapter 5

Prototype implementation

This chapter presents the implementation details of the co-simulation framework prototype. The chapter starts with the implementation of the CANoe part, followed by the configuration of dSpace toolchain. Furthermore, the process of setting up co-simulation is described.

5.1 CANoe environment implementation

This section shows the process of implementing the CANoe part of the co-simulation framework prototype. The CAN message database configuration from the production wheel loader is firstly presented, followed by descriptions of CAPL scripts programming for two evaluation scenarios, control panel design and hardware connection setup.

In order to keep the prototype and its content as clear as possible, the initial database which comes from the actual production wheel loader is edited according to prototype functions defined in section 4.4. Signals that are not relevant to the functions are directly removed from the database. The final database shows that all the remaining CAN messages use extended CAN frame format with 29 identifier bits.

5.1.1 CAPL scripts programming

CAPL scripts are deployed on each behavior model and emulate their behavior as accurately as possible. The scripts can be called the behavior model itself. At the beginning of the entire implementation of the prototype, all non-real ECU nodes are purely simulated in the CANoe environment, and part of the HMICU is replaced by the virtual ECU and the Simulink model in the later

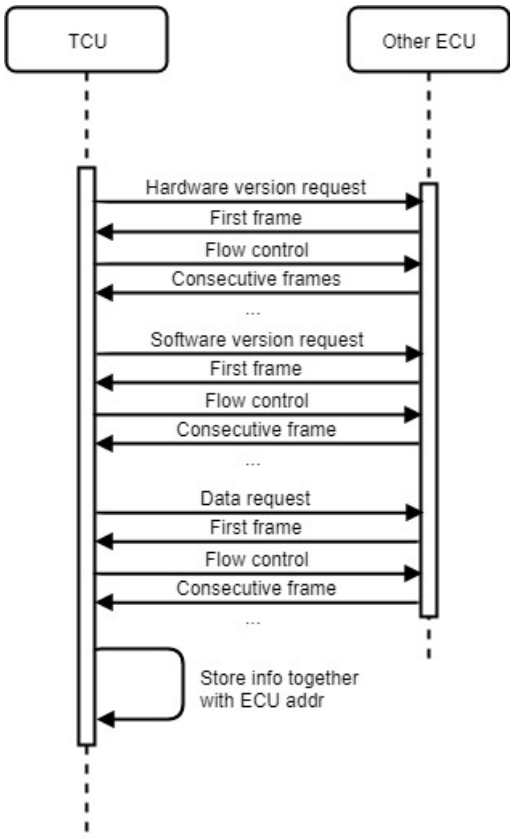

Figure 5.1: Complete workflow for one ECU node

implementation, then the prototype becomes corresponding to its topology in figure 4.2.

With respect to the prototype functions, message-triggered procedures are mapped to the arrival of specific receiving machine network map request messages and values of machine data that can be changed by timers whose period is determined by the period of cyclic messages in the database. This section shows how functions of the framework prototype are implemented through the CAPL scripts for multiple ECUs.

Machine network map

The information of the machine network map transfers through UDS above CAN frames in the product wheel loader. In this case, the *on message* event procedure is invoked after the ECU node receives the corresponding request message then the response containing the machine network map information of the current node is sent back to the TCU. Additionally, the length of the entire

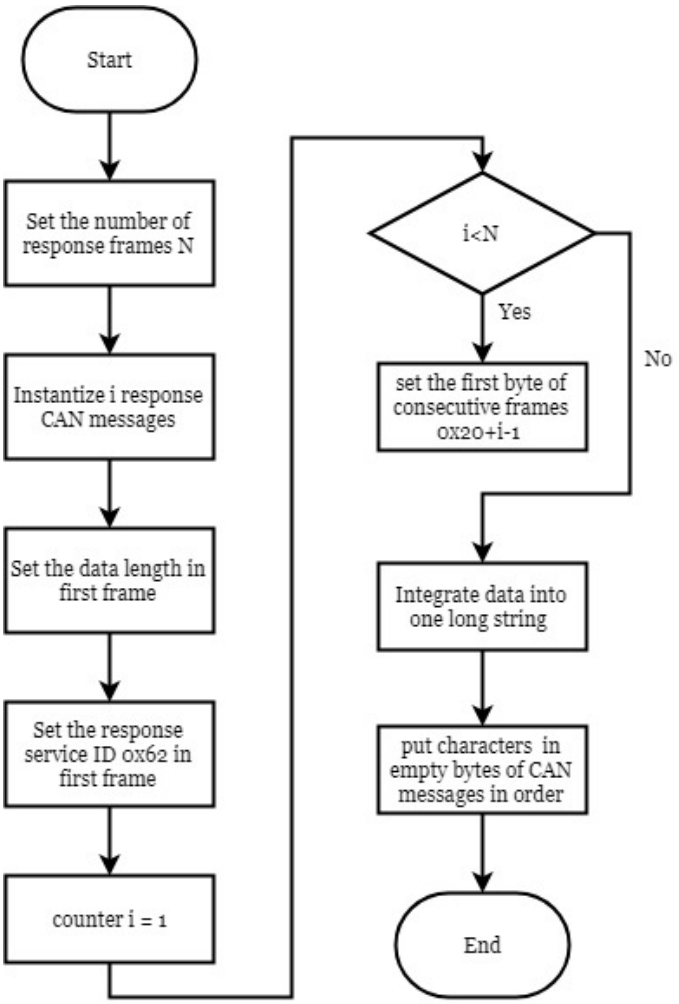

Figure 5.2: The flow of machine network map response generation

machine network map information has exceeded the maximum length of the 8-byte data section in a CAN data frame, so the information should be segmented into multiple CAN frames. The complete workflow of machine network map request and response for one single ECU node is shown in figure 5.1.

The service ID the prototype use is 0x22 which means *"Read data by identifier"*. The first response frame in figure 5.1 informs the receiver of the length of the entire machine network map information, which is the number of response frames, as well as the response service ID. After the TCU sends back the flow control, the following consecutive frames can be sent to the TCU sequentially. There are three types of machine network map information required by the TCU as seen in figure 5.1, software version, hardware version

and data, and the information is identified by the TCU through the component ID section in the first frame.

The machine network map information is a set of data strings, functions are implemented to encapsulate them into multiple frames in order. The sequence of generating the machine network map is presented in figure 5.2. As the context of the machine network map is already known, the number of frames and the entire data length are directly determined. The response service ID and the data length are put into the first frame initially. Response data strings are integrated into one long string and then the set of message frames takes characters one by one.

Machine data update

In this function, messages are mainly cyclic. For those cyclic messages, signal values are able to be edited directly in different events but no *send* commands are required as they will be sent out every period time which is already defined in the CAN database for them. At the same time, it is not necessary to instantiate the cyclic message in the variables part of a CAPL script. Below is a part of the CAPL script for updating fuel level.

Listing 5.1: CAPL sample script for updating fuel level

```
variables
{
  timer tFuelLvl;
  double gFuelLvl:
}

on start
{
  // Period of the timer: 1s
  setTimer(tFuelLvl, 1);
  gFuelLvl = 100;
}


on timer tFuelLevel
{
  $HMICU_MachineData::FuelLvl=@MachineStatus::FuelLvl;
  setTimer(tFuelLvl, 1);
}
```

The CAPL script above for updating fuel level includes one timer whose period is set as 1 second. The value of system variable *FuelLvl* which belongs to the *MachineData* namespace is assigned to the corresponding signal whenever the timer expires. During the execution of the script, CANoe will transmit the message *HMICU_MachineData* with this value. Additionally, simulating the change in the value of *FuelLvl* is programmed in another place of the script.

The one another thing worth mentioning is that the local variables in the script are always declared statically, so that variables will only be initialized once at the first time the event procedure or the function is executed, and then the value will be kept until the next manipulation.

5.1.2 Control panel

Control panel is a user-defined graphical interface which is implemented in order to improve the immersiveness of the simulation and simplify usage of the tool. Different ECUs in the framework have the responsibility to transfer different machine data and some of machine data is registered as system variables that can be edited or displayed on the control panel. The system variables are global data objects in the entire CANoe environment and the connection between the control panel and CAPL scripts are built through them. Changes in the value of the object on the panel can be reflected on its registered system variable whose change can be detected by the script, then corresponding procedures are invoked.

Each node having its own panels may be better for clarity in most applications. However, the implemented control panel in the thesis integrates components for different ECU nodes into one panel window in order to have a more directive view of environment variables and decrease the complexity of the framework maintenance, as shown in figure 5.3.

The two control components, the main switch and the ignition key, are used to activate the system while simulating the real working condition of the vehicle. Only when both of the control components are turned to be 1 can the other components start working. The light for engine status in the panel is turned to be green at the same time. At the current stage of the framework implementation, signal values of fuel level, engine speed and machine time are simulated and updated by CAPL scripts in the corresponding ECU nodes.

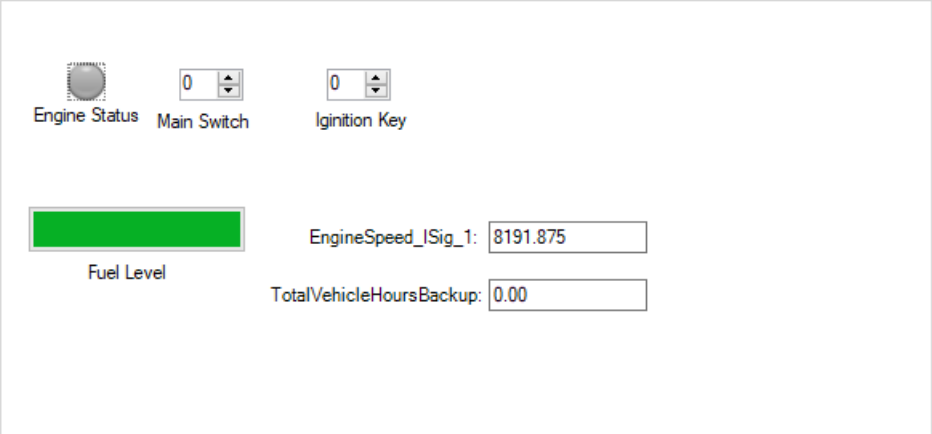

Figure 5.3: Control panel

5.1.3 Physical ECU access

In the previous sections, ECU behavior models are implemented and the TCU is replaced by an interaction layer where machine network map requests are able to be manually sent out by users. This section has the physical TCU included in the prototype then the request part of the machine network map function happens automatically at each time the TCU starts up.

The most important reason why the TCU is selected as the actual ECU in the figure 4.2 for the implementation of the framework prototype is that the TCU can connect to the remote portal where the machine status and machine network data the TCU actually receives are able to be checked conveniently. For the TCU, previous work means that the rest of the bus which provides all essential signals it needs in the real production application scenario. Meanwhile, the simulation mode of CANoe should be set as real bus where the prototype project runs in real time instead of running as fast as possible.

The Vector VN8910, which is designed to use with CANoe, is used as the hardware interface in the implementation and the physical CAN bus channels are assigned to the bus channels in the CANoe project. In the case of this thesis, only one CAN bus channel is used and assigned to channel 1 of the hardware interface while the hardware interface supports up to 4 CAN bus channels.

The connection between the TCU and the hardware interface needs to be configured manually. The pin map of the VN8910 D-subminiature 9-pin port is shown in figure 5.4 and only pins for CAN high and CAN low are used for the connection.

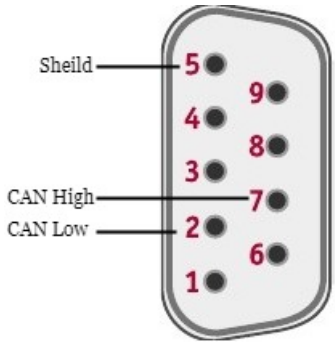

Figure 5.4: Pin map of the port in Vector hardware interface

Twisted cables are made to connect to pin 7 and pin 2, and a 120 Ω resistor is attached at one end of the connection. The handmade implementation of the connection is shown in figure 5.5. If there are going to be multiple ECUs accessed to the co-simulation framework, the hardware interface can be considered as one ECU node then the connection of CAN bus is able to be built in the regular way.

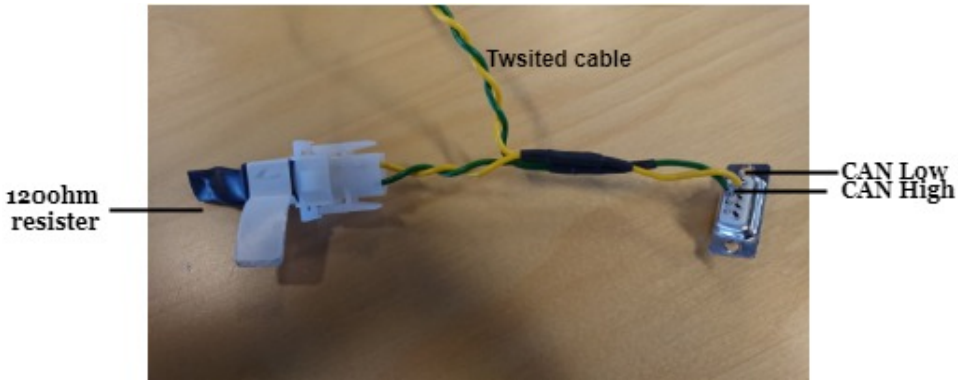

Figure 5.5: Physical CAN bus implementation for the hardware interface

5.2 Virtual ECU generation

This section will describe the virtual ECU generation. As the company VCE is currently at the evaluation stage of the next generation testing technology, the complete dSpace toolchain is not available. Partial technical details of virtual ECU generation are discussed below.

The virtual ECU is currently delivered from the dSpace side as the production tool SystemDesk, which has the ability to generate virtual ECUs in

their toolchain, is not provided at this evaluation stage. Meanwhile, the software architecture at VCE is a highly customized version of AUTOSAR, so a lot of engineering efforts are required if the production C code can be directly used to generate a virtual ECU. Thus a simplified virtual ECU is decided to be integrated into'5 the prototype and the tool SystemDesk has not been directly evaluated but did participate in the implementation process of the framework prototype.

As a proof-of-concept work, the virtual ECU, which is included in the thesis prototype, is generated based on the database that is used for the CANoe project. In order to decrease the complexity of virtual ECU generation and view possibilities clearly, the database used in SystemDesk is simplified with only one CAN message containing the engine speed signal which belongs to the HMICU. The resulting virtual ECU is able to transmit the message according to its requirements so that it can be said that part of the HMICU is virtualized.

After the database file is imported into the SystemDesk, the three .arxml description files are created automatically. Then an ECU Simulink model is made based on the new .arxml description files that show what the basic software components are required for the ECU. The production C code is generated from the Simulink model through TargetLink and added into the previous SystemDesk project where the complete virtual ECU will be generated.

Above is an entire process of virtual ECU generation. If the production code of the ECU already exists, then the process of building and transferring Simulink model is not required anymore, the production C code and the .arxml files are enough for SystemDesk to generate a workable virtual ECU. Tasks of the ECU are shown and scheduled based on the defined order.

As a result, the virtual HMICU is able to send out the engine speed message every 100ms with dummy data.

5.3 Co-simulation

This section shows the process of combining the previous CANoe project with the VEOS application and a Simulink model through FMU.

5.3.1 FMU export

When the CANoe project implements its FMU, system variables and signals in the database should be selected before its generation, and values of selected

variables and signals are going to be exchanged between different environments during co-simulation. Types of those selected objects are categorized into input and output at the time of selection. The input objects will read values from the mapped object while the output objects write to the mapped objects. Figure 5.6 shows the selection of signals and variables of FMU that is used during co-simulation.

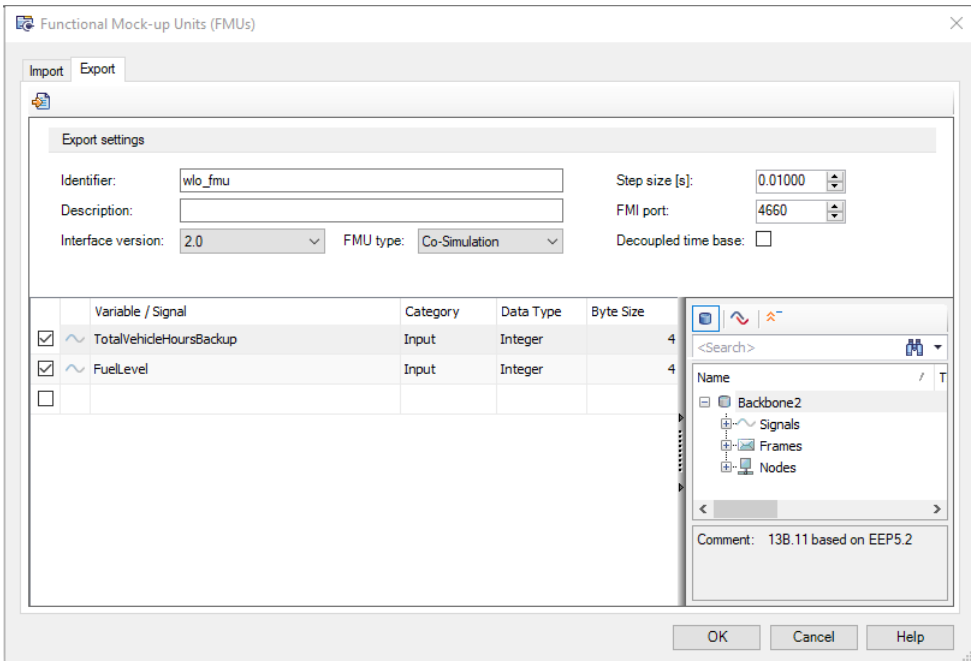

Figure 5.6: FMU configuration before export

The framework prototype selects the machine time as one input port of the FMU from the CANoe project. Value of the signal will be transmitted from another integrated model in the VEOS application at discrete time points during co-simulation.

5.3.2 Simulink model

A Simulink model is implemented so that the extensibility of the framework is presented. The Simulink model is responsible for providing the value of the machine time through a counter block. Figure 5.7 provides details of the Simulink model.

A pulse signal whose period is 1 second connects to the counter block. The unit of machine time is hour, so the output value of the counter should be divided by 3600 because the value that the counter block outputs is in seconds.

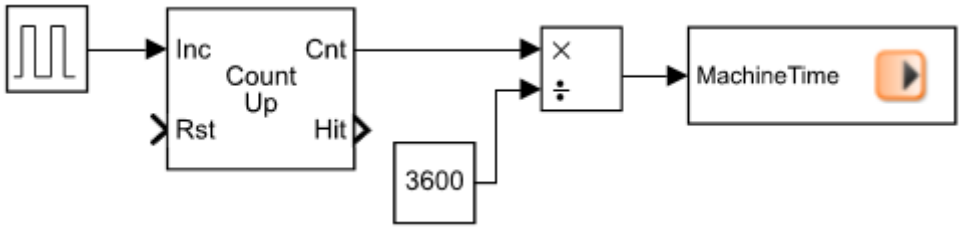

Figure 5.7: Simulink model for value of machine speed

The rightmost block is the data output block of the dSpace model interface blockset and the block is able to provide one or more data output ports to send signals to another behavior model or function port. Also value of blocks in the Simulink model can be changed which will be mentioned in the following section about co-simulation configuration.

5.3.3 Co-simulation configuration

The virtual ECU is able to be included in either virtual or HiL bus environment through different tools inside the dSpace toolchain. As the framework prototype in the thesis puts the physical ECU in the CANoe part, the rest of the framework should be purely virtual, then VEOS is used.

After the FMU from the CANoe project and the Simulink model have been implemented, the VEOS application imports them into itself. It builds the existing virtual ECU and the purpose for the building step is to add services for different models which are used for measurement and calibration in the later implementation.

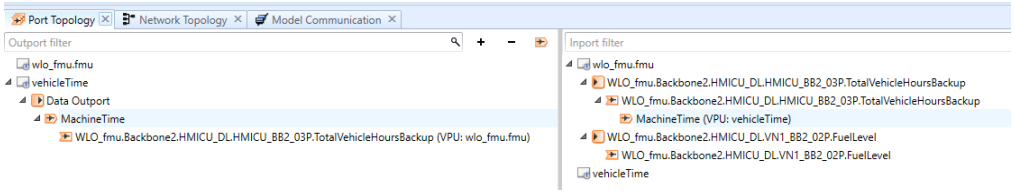

Figure 5.8: Co-simulation connection

The input and output ports of integrated virtual ECUs and models are listed in two different queues on the VEOS panel. Ports of different objects are able to be connected directly. Figure 5.8 shows that output port of the Simulink model connects to the machine time port of the FMU so that data stream from

the Simulink environment can be transmitted to the CANoe environment with the help of FMI.

Figure 5.9 shows how different components are mapped to different simulation environments in the framework prototype. It is obvious that the CANoe project provides the most functions of the framework and the dSpace toolchain help bond different environment together.

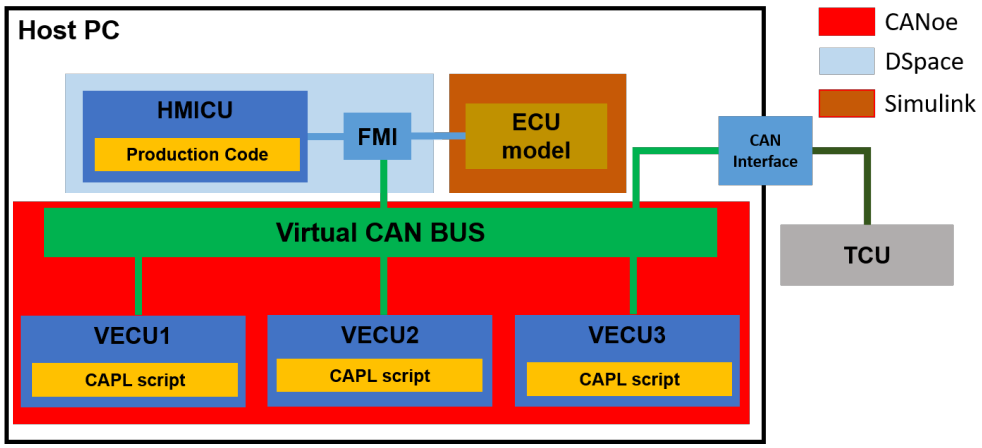

Figure 5.9: Map of tools for different components

5.3.4 Co-simulation monitoring

Values of signals and variables and bus communication status need to be tracked during the run time of the prototype. As the prototype uses the co-simulation mode of FMI, solvers of both tools are working at the same time. So trace windows on the CANoe project side can be directly used.

On the side of dSpace toolchain, ControlDesk is used for the purpose of data tracking. The description files of models in the VEOS application are firstly registered in ControlDesk then variable values and bus messages can be selected and monitored. Meanwhile, variable and signal values are able to be changed by ControlDesk during the run time.

Chapter 6

Evaluation

This chapter starts by designing two evaluation scenarios for the implemented co-simulation framework prototype based on its functions. Then evaluation results of the two scenarios are presented in sequence. The last part of this chapter discusses the evaluation results and properties of the prototype.

6.1 Design of evaluation scenarios

The only physical ECU in the framework prototype is the TCU. The environment within the host PC provides the residual bus environment for it. Functions of the prototype and the co-simulation among multiple tools should be proven through evaluation scenarios. Two evaluation scenarios are designed as follows.

6.1.1 Machine network map request and response

The machine network map is an important set of data which contains the list of ECU nodes that are connected with the TCU, as well as versions of software and firmware of those connected nodes. The machine network map request and response are encapsulated in the CAN message using UDS format. It is very useful for machine diagnostic and maintenance. The TCU collects the machine network map at each start up and then sends the information to the remote portal if any parameters are changed from the previous boot.

Every time the TCU is powered up, it will send the machine network map request to ECU nodes in a fixed order in its node list. Each request repeats three times with different component ID which represents for different kinds of target machine network map data. The other ECUs will respond the spe-

cific kind of machine network map data to the TCU and the data are changed manually so that the data displayed in the remote portal should change to same data as well, then the function of the prototype can be proved.

The purpose of this evaluation scenario is to test whether event-triggered events that are implemented in the prototype can be invoked and accepted by the physical ECU.

6.1.2 Machine data update

The machine data includes fuel level, engine speed, machine time and engine status. Each of these four kinds of machine data corresponds to a CAN signal and all of them will be sent to the TCU. Additionally, the engine status will be updated in the remote Volvo careTrack portal while the machine is on.

All CAN signals which contain machine data information in the thesis belong to cyclic CAN messages. After the user turns on both the main switch and the ignition key on the control panel in the CANoe project, which means that the engine status becomes on, machine data starts update in the ECU network and their values are simulated by some dummy numbers.

This evaluation scenario is intended to check the availability of co-simulation between different environments.

6.2 Machine network map evaluation scenario

It is known that TCU sends the machine network map data to the remote Volvo careTrack portal when changes of the vehicle machine network map information are noticed. The stored old machine network map data is able to be checked through Volvo's own engineer software, then the data which will be encapsulated into the machine network map response frames is turned into different strings but still keeps the same length. Then it is clear to evaluate whether the machine network map information is accepted by TCU in the right way.

Regarding evaluating the machine network map scenario, one of the most significant features to check is that the set of CAN message frames is sent in the right format and order. Some problems occurred during the evaluation process. The first one is that the license for CANoe at VCE only supports the real bus simulation mode of an older version of CANoe but it is available for the simulated bus mode of the new version the thesis uses. In this case, CAPL scripts for each node are evaluated separately through CANalyzer which is a weakened version of CANoe with monitoring function.

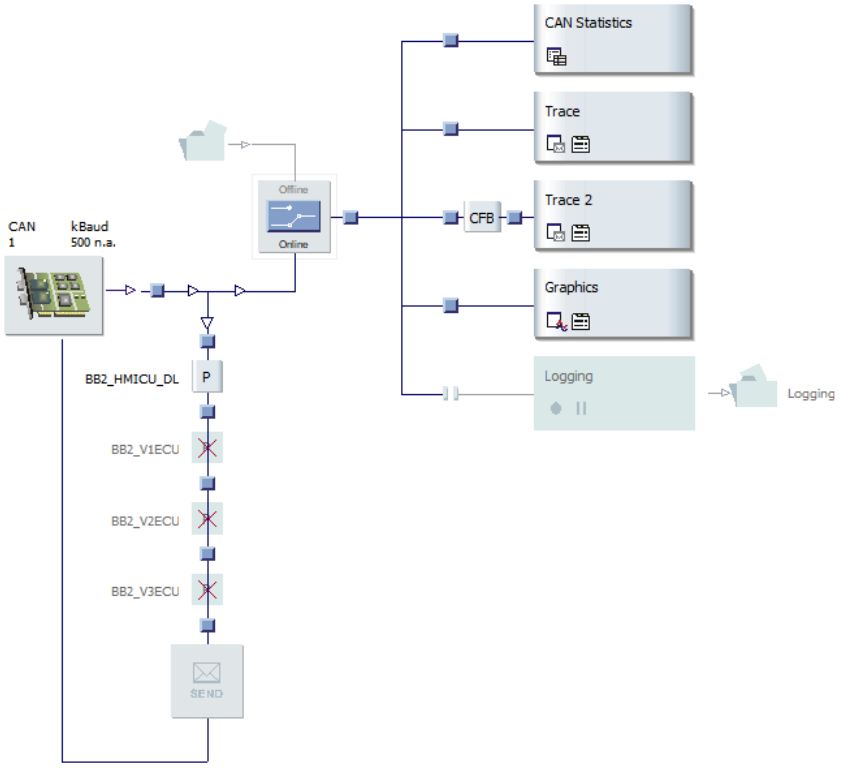

Figure 6.1: CANalyzer project for CAPL scripts evaluation

CAPL scripts are injected into the CANalyzer project sequentially with TCU connected as presented in figure 6.1. Only one script will be evaluated at a time and the other scripts are blocked otherwise the CANalyzer will not work correctly.

The TCU sends the machine network map request in order and waits 5 seconds if there is no response received then turns to the next ECU node in the list under normal circumstance. Thus, evaluating one CAPL script at a time has no influence on evaluation results because it can be treated as the only one ECU node in the CAN network.

Evaluation results show that CAPL scripts for the 4 non-real ECU nodes responded to their own machine network map request with expected reaction. The first response frame is sent after the target ECU receives the request and consecutive frames are sent after the target ECU receives the flow control. The changed information could be seen at the remote portal after about 3 minutes. This evaluation was repeated 3 times with different machine network information, the same results were obtained.

6.3 Machine data update scenario

As mentioned above, this evaluation scenario is mainly used to test the possibility of co-simulation between different environments. CANalyzer that is used as the alternative to CANoe does not support the FMI standard, so it cannot be used in this evaluation scenario anymore. Instead, the CANoe project runs in the simulated bus mode where the TCU is not connected and replaced by a CAPL script.

After starting the VEOS application, other simulation environments start working at the same time automatically. A ControlDesk experiment is registered to the VEOS application. Same data can be seen when comparing data in the ControlDesk experiment and the CANoe project. The actual working condition is shown in figure 6.2.

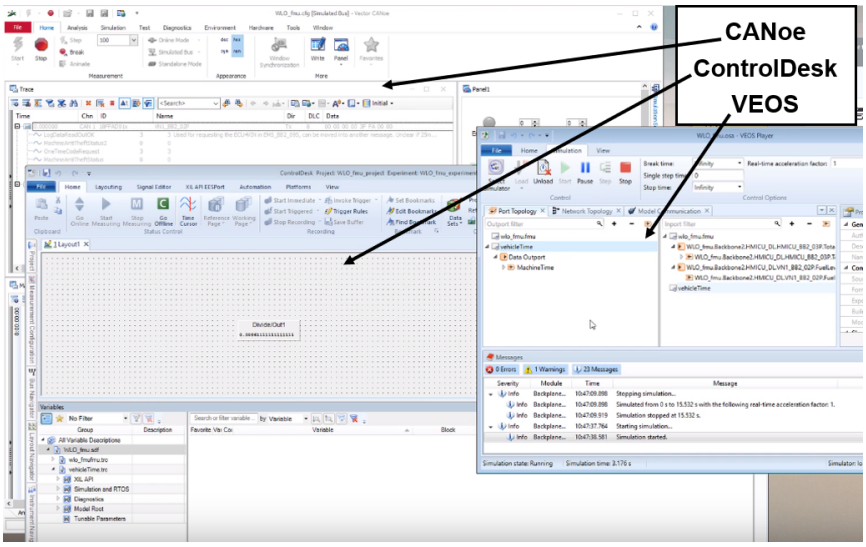

Figure 6.2: Actual working condition of co-simulation

Results of this scenario show that the co-simulation between different environments works. Value of the machine time from Simulink model is updated in the CANoe project through the FMI connection, and the value can be tracked in both ControlDesk experiment and trace window in CANoe during the runtime of the simulation.

Regarding the evaluation of co-simulation with the delivered virtual ECU, it is tested individually with the TCU through dSpace hardware interface Scalexio due to limitations on the evaluation stage of the company. When the virtual ECU is connected to the CAN bus together with the TCU, periodical messages containing a dummy machine speed number is able to be seen on the

CAN bus, and the value of the machine speed proves to be changed in the registered ControlDesk experiment.

6.4 Analysis of the evaluation

Results of the two evaluation scenarios turn out that the implemented prototype meets its requirements in section 4.4. The prototype successfully updated the correct machine network map to the remote platform and the machine data is transmitted periodically in the network. Both scenarios are repeated at least three times in order to improve the feasibility of the results. Requirements in table 4.2 and their testing results are shown in table 6.1.

Table 6.1: Prototype requirements and their testing results

Requirements	**Testing results**
The TCU can send requests to nodes in the ECU network with the right format and content	The TCU sent machine network map requests to nodes in the network sequentially, and the content and format of the request messages were correct
Nodes other than the TCU can respond to the request with the right format and content	Responses from other ECU nodes had correct format and parameters
The machine network map can be correctly up-dated to the Volvo careTrack platform	The machine network map which was set by the tester was seen on the remote portal
Periodic messages can be monitored on the CAN bus	The four machine data signals in table 4.1 were transmitted every 100 milliseconds
Values of periodic messages are simulated and always changed	When the machine data signals were monitored on the CAN bus, their values were changed with the defined dummy data

Virtual ECU, Simulink models, physical ECU, behavior models from multiple simulation tools are integrated into one co-simulation framework in the two evaluation scenarios. Results of the evaluation prove that the framework can accommodate with multiple ECU implementation methods which are from abstract to concrete.

The evaluation of the virtual ECU is limited. Although AUTOSAR is a popular software architecture standard in the automotive industry, companies like Volvo CE still do not tend to follow it but use internal tools instead. At the same time, most of the virtual ECU generation tools on the market have high requirements on ECUs which wants to be virtual. DSpace supports generating virtual ECU from the non-AUTOSAR ECU code, but it requires a lot of engineering effort and the work is to add a medium to convert the origin code to those fulfilling the AUTOSAR standard. If the migration can be done, it will be quite convenient and efficient to generate a virtual ECU with the production code and test the new code. ETAS ISOALR EVE is a powerful virtual ECU generation tool as well, but only has support to ECUs following AUTOSAR 4.

Apart from the virtual ECU with one-time high costs on time and resources, the other methods to represent ECUs offers a more efficient and affordable way to set up the environment for the ECU under test because they do not require expensive hardware components. The main cost of the framework comes from licenses of the tools included, but those tools are popular in the automotive industry, so most companies have already purchased these software licenses, The Simulink model provides a user-friendly way to build a model for the machine components or nodes at the early design stage, and it would be better if the project team totally follows model based development, otherwise building and maintaining Simulink models only for the testing purpose is not a good choice from the perspective of economy and efficiency. Physical ECU is the final form of an ECU node which provides stable and reliable support but it is available at the very late stage. Additionally, the behavior model is the easiest one to build among the four forms of ECUs and it has a similar limitation to the Simulink model. It needs to be updated together with the project progress.

As a result, if there is no plan or budget for the AUTOSAR code migration, the CANoe part with ECU behavior models and FMI support is enough for testing the bus behaviors of the ECU under test, otherwise, the virtual ECU has attractive features of testing new code efficiently and providing a more realistic environment.

The co-simulation works because same data is tracked in different environment. One significant advantage of the co-simulation based on FMI that can be seen in the thesis is that it can adapt to changing needs and be deployed easily across a wide range of stakeholders. Different environments are connected to the FMI simulation master and they can choose what they want to exchange with others. The FMU used in the thesis is directly implemented by the tool

itself and no extra interfaces need to be programmed. Different tools are used to implement different subsystems and accomplish different goals, it is a struggle to combine them together. The co-simulation combines subsystem models into a full system which brings full flexibility to the work, engineers then can have a larger variety of tools at their disposal to design and implement their products.

With regard to the properties the framework is intended to test, table 6.2 shows whether they can be tested in the co-simulation framework. The content of messages on the bus can be directly seen. As the history of overall bus communication is recorded, the bus load can be obtained. However, the framework does not have the real-time property at this time, and it just runs as fast as possible when running the co-simulation, so the timing of messages cannot be tested currently. At the same time, the framework is assumed to be error-free at this stage, so the packet loss of the network cannot be tested as well.

Table 6.2: Results of the properties intended to test

Properties	**Whether can be tested**
Timing of messages	No
Content of messages	Yes
Packet loss	No
Bus load	Yes

When compared to the HiL rig, the overall cost of the proposed efficient co-simulation framework is lower and the rig has a more realistic environment for bus communication. In the first scenario, the prototype transmits the consecutive frames almost with no time interval while the rig sends them with a time interval of 50 milliseconds. However, the result turns out that the remote portal can receive correct data in both the implemented prototype and the HiL rig. The scenarios were repeated multiple times and got the same result. The majority part of the framework is located in a host PC and its physical ECUs should be a stable version or the ECU under test, thus the number of hardware components and difficulties in upgrading and modifying the system decrease.

The bottleneck of the co-simulation framework is not evaluated in the thesis, but it is considered to exist in the host PC. If more virtual ECUs and Simulink models are added into the framework, solvers of simulation tools will occupy increasing computing resources which may lose the synchronization with the physical ECU. This feature is an important future work of the thesis.

Chapter 7

Summary and Conclusions

This chapter presents summary and conclusions learned throughout the thesis. Conclusions and completion level of goals are given, followed by discussion about limitations of the thesis. The last part of this chapter, which is also the last part of the thesis, describes some future work that can be done.

7.1 Summary

As the complexity and interconnection of the ECUs in the vehicle continue to increase, we still hope that the time to market can be as early as possible. The demand on the testing process increases dramatically. The ECU under test is always considered a black box due to limited knowledge of testers and limited access to source code in the process of testing. The current HiL simulation system provides a realistic environment for the ECU under test, but it is expensive to build and maintain.

The thesis proposes a co-simulation framework for black-box testing of in-vehicle ECUs and implements a prototype which originates from an actual product wheel loader. The co-simulation framework is intended to provide a proof-of-concept for follow-up studies in the automotive testing area. Four different forms of ECUs, ECU models, virtual ECUs, physical ECUs and behavior models, are included in the framework and they become available at different stages of the project development life cycle. FMI is used to connect simulation tools that are responsible for implementing different forms of ECUs and exchange data between them. Once a node is set as the ECU under test, the rest of the framework provides a relatively realistic environment for it. The ECU under test can receive responses to its requests and updates to other required information. Through the co-simulation framework in the thesis, the

communication status of the ECU under test and the load of the communication bus can be acquired easily. However, the framework cannot guarantee the timing of messages.

7.2 Conclusions

With the demand to ECU testing increasing dramatically, the origin testing method HiL rig is no longer as good as before because of its high cost and difficulty to maintain. Additionally, the HiL rig is only used at the very late stage of the project life cycle.

This thesis designs a co-simulation framework and implements a proof-of-concept prototype containing multiple forms of ECUs including virtual ECUs, Simulink models, as well as CAPL scripts. The framework uses black-box testing for in-vehicle ECUs and bus communication behavior of the ECU network is used to test the ECU under test in the thesis. The co-simulation framework is able to be used at multiple stages of projects using V-model and supports MiL simulation, SiL simulation and HiL simulation. Simulink model is useful for nodes at the early design stage and virtual ECU allows the new programmed code to be loaded and tested in a relatively realistic environment. Physical ECU is basically mature which can be treated as a black box. ECU model is a complement of the above three forms of ECUs .

Virtual ECU provides an effective way to test ECU software but also strict requirements on it, for example, totally following AUTOSAR. Meanwhile, current virtual ECU generation tools are quite expensive. So the practical value of its usage in the framework need to be considered based on budget and current software architecture.

The co-simulation framework prototype implemented in the thesis is not as accurate as the HiL rig. Bus messages and variables are monitored to test ECUs, and the prototype is tested through two evaluation scenarios from aspects of scripts behaviors and co-simulation data communication.

7.2.1 Completion level of goals

This section discusses the goals of the thesis and compares results of the thesis with respect to the original goals. The goal of the thesis is to design and implement a co-simulation framework prototype.

Regarding the design goals of the thesis, the framework prototype should contain both real and virtual ECUs and repeat the real bus communication behaviors. The proposed design of the prototype meets goals above as it contains

virtual ECU, physical ECU and ECU behavior models. And the prototype has the correct behavior while responding to machine network map request. Additionally, a Simulink model is integrated into the framework prototype.

When it comes to implementation goals of the thesis, the original idea was to migrate the HiL rig to software with the same performance. However, only functions for two designed scenarios are implemented. And part of functions is not evaluated because of a software license problem and late arrival of virtual ECU delivery.

7.2.2 Limitations

In order to achieve a suitable design of the co-simulation framework, plenty of time was invested in the research of the state-of-art techniques and different industrial solutions.

Also a long period of time was spent on negotiating with different suppliers for their virtual ECU generation tool and making a choice between those different tools which took much more time than expected. As a result, the implementation of the framework prototype was delayed.

Another limitation of the thesis is that there is no standard virtual ECU included in the implemented prototype. The virtual ECU evaluated in the thesis is generated from the database file and requires complex manual operation instead of directly importing the production C code and AUTOSAR configuration description files.

7.3 Future work

Because of the limited time budget and the author's limited knowledge, parts of the prototype are undone and some other work can be improved. This section discusses what future work can be done and how to conduct it.

7.3.1 Virtual ECU

The current virtual ECU used in the thesis is delivered from dSpace and only one is included. As mentioned in section 7.2.2, standard virtual ECU should be included in the prototype and evaluated. Also the current virtual ECU sends messages directly on the CAN bus. The further ECU can try to use data ports which are better for connecting different models and sharing data between models.

Known bottleneck of the co-simulation framework lies in the computing power of the host PC but it is not discussed in the thesis because the prototype only contains a limited number of ECUs which is far away from the limitation of the computing power of the PC. More virtual ECUs should be added into the framework prototype in order to discover the bottleneck of it.

7.3.2 SIMONE

The current on-the-shelf productions for virtual ECU generation only apply to those who use AUTOSAR software architecture. Volvo CE has its own internal software SIMONE for virtual ECU simulation but SIMONE does not support hardware access and can only run as fast as possible.

The hardware interface for CANalyzer and CANoe which is made by Vector has an open source universal programming interface. Third-party applications are able to use the library to access the virtual CAN bus and the real CAN bus in the interface. So there is a potential solution that SIMONE supports hardware access through the Vector hardware interface. The difficulty of the implementation lies in the time synchronization between the hardware interface and SIMONE. The hardware interface records the time stamps of each message that pass it, it is possible to use the time stamp to reach some kind of "soft real-time" for the simulation, but this still needs to be evaluated.

7.3.3 Surrounding environment

Last but not least, surrounding environment signals from cameras, radars can be added as the input of ECUs into the co-simulation framework. And it would be interesting that the ECU processes the received environment signals and then controls connected simulated actuators which are able to have influence on the simulated surrounding environment.

In this case, the in-vehicle communication system can communicate with the environment now and reach a closed loop with the environment.

Bibliography

[1] Peter Marwedel. *Embedded System Design*. 2nd. Springer, 2011. doi: 10.1007/978-94-007-0257-8.

[2] Alberto Sangiovanni-Vincentelli et al. "Benefits and challenges for platform-based design". In: *Proceedings of the 41st annual Design Automation Conference*. 2004. doi: 10.1145/996566.996684.

[3] Tino Schulze and Jan-Eve Stavesand. "Hardware-in-the-Loop Test Process for Modern E/E Systems". In: *Simulation and Testing for Vehicle Technology: 7th Conference*. 2016. doi: 10.1007/978-3-319-32345-$9_2$3.

[4] Andrei Hagiescu et al. "Performance analysis of FlexRay-based ECU networks". In: *Proceedings of the 44th annual Design Automation Conference*. 2007, pp. 284–289.

[5] W Richards Adrion, Martha A Branstad, and John C Cherniavsky. "Validation, verification, and testing of computer software". In: *ACM Computing Surveys (CSUR)* 14.2 (1982), pp. 159–192. DOI: 10.1145/356876.356879.

[6] Hosam Fathy et al. "Review of Hardware-in-the-Loop Simulation and Its Prospects in the Automotive Area". In: *Modeling and Simulation for Military applications* 6228 (2006).

[7] Wayne Goddard and Stuart Melville. *Research methodology: An introduction*. Juta and Company Ltd, 2004.

[8] Wikipedia. *Electronic control unit*. 2019. URL: https://en.wikipedia.org/wiki/Electronic_control_unit (visited on 04/15/2019).

[9] Embitel. *'ECU' is a Three Letter Answer for all the Innovative Features in Your Car: Know How the Story Unfolded*. 2017. URL: https://www.embitel.com/blog/embedded-blog/automotive-control-units-development-innovations-mechanical-to-electronics (visited on 04/23/2019).

[10] Ugur Keskin. “In-vehicle communication networks: a literature survey”. In: *Computer Science Report* 10 (2009).

[11] Edward A. Lee. “Embedded Software”. In: *Advances in Computers* 56 (2002), pp. 55–95. DOI: 10.1016/S0065-2458(02)80004-3.

[12] Lui Sha et al. “Real Time Scheduling Theory: A Historical Perspective”. In: *Real-Time Systems* 28.2 (2004), pp. 101–155. DOI: 10.1023/B:TIME.0000045315.61234.1e.

[13] Christof Ebert and Capers Jones. “Embedded Software: Facts, Figures, and Future”. In: *Computer* 42(4) (2009), pp. 42–52. DOI: 10.1109/MC.2009.118.

[14] AUTOSAR. 2019. URL: https://www.autosar.org/ (visited on 04/11/2019).

[15] Patrick Shelly. *Operating systems for cars*. 2016. URL: https://evroadmapconference.com/program/presentations/PatShelly.pdf.

[16] Donald Lewine. *POSIX programmers guide*. ” O’Reilly Media, Inc.”, 1991.

[17] Real-Time Systems Laboratory. *An introduction to AUTOSAR*. URL: https://retis.sssup.it/sites/default/files/lesson19_autosar.pdf.

[18] Peter M. Kruse, Joachim Wegener, and Stefan Wappler. “A Cross-platform Test System for Evolutionary Black-box Testing of Embedded Systems”. In: *SIGEVOlution* 5.1 (May 2010), pp. 3–9. ISSN: 1931-8499. DOI: 10.1145/1811155.1811156.

[19] *What is Embedded Testing in Software Testing?* URL: https://www.guru99.com/embedded-software-testing.html#4 (visited on 08/30/2019).

[20] Muhammad Zohaib Iqbal, Andrea Arcuri, and Lionel Briand. “Environment Modeling with UML/MARTE to Support Black-Box System Testing for Real-Time Embedded Systems: Methodology and Industrial Case Studies”. In: *Model Driven Engineering Languages and Systems*. Ed. by Dorina C. Petriu, Nicolas Rouquette, and Øystein Haugen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 286–300. ISBN: 978-3-642-16145-2.

[21] Srinivas Nidhra and Jagruthi Dondeti. “Black box and white box testing techniques-a literature review”. In: *International Journal of Embedded Systems and Applications (IJESA)* 2.2 (2012), pp. 29–50.

[22] Tingting Yu et al. “Using property-based oracles when testing embedded system applications”. In: *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*. IEEE. 2011, pp. 100–109.

[23] Agilent Technologies. *Increase Automotive ECU Test Throughput*. URL: https://doc.xdevs.com/doc/HP_Agilent_Keysight/ (visited on 08/31/2019).

[24] G. Tibba et al. “Testing automotive embedded systems under X-in-the-loop setups”. In: *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2016, pp. 1–8. DOI: 10.1145/2966986.2980076.

[25] Vivek Jaikamal. *Model-based ECU development – An Integrated MiL-SiL-HiL Approach*. Tech. rep. 2009-01-0153. SAE Technical Paper, 2009. DOI: 10.4271/2009-01-0153.

[26] Agonne National Lab. *Model based design definition of terms*. 2012. URL: http://www.autonomie.net/references/model_based_design_defs_24c.html (visited on 04/23/2019).

[27] Marko Bacic, Neild Simon, and Peter Gawthrop. “Introduction to the special issue on hardware-in-the-loop simulation”. In: *Mechatronics* 19.7 (2009), pp. 1041–042. DOI: 10.1016/j.mechatronics.2009.09.005.

[28] L. Heidrich et al. “Hardware-in-the-loop test rig for integrated vehicle control systems”. In: *IFAC Proceedings* 46(21) (2013), pp. 683–688.

[29] Sven Flake. *What Are Virtual ECUs?* 2018. URL: https://www.dspace.com/en/inc/home/news/engineers-insights/blog-virtuals-ecus-1808.cfm (visited on 04/16/2019).

[30] Andreas Junghanns et al. “Building virtual ECUs quickly and economically”. In: vol. 7(3). 2012, pp. 48–51.

[31] Paul Rook. “Controlling software projects”. In: *Software Engineering Journal* 1(1) (1986), pp. 7–16.

[32] Victor Reyes. *Dealing with automotive software complexity with virtual prototyping*. 2014. URL: https://www.embedded.com/design/prototyping-and-development/4430390/1/Dealing-with-automotive-software-complexity-with-virtual-prototyping---Part-1--Virtual-HIL-development-basics (visited on 04/18/2019).

[33] S Balaji and M Sundararajan Murugaiyan. “Waterfall vs. V-Model vs. Agile: A comparative study on SDLC”. In: *International Journal of Information Technology and Business Management* 2(1) (2012), pp. 26–30.

[34] Wolfhard Lawrenz. *CAN System Engineering From Theory to Practical Applications*. Sprinter London, 2013. DOI: 10.1007/978-1-4471-5613-0.

[35] ISO 14229-1. *Road vehicles - Unified diagnostic services (UDS)*. standard. International Organization for Standardization, 2013.

[36] FMI. 2019. URL: https://fmi-standard.org/ (visited on 06/08/2019).

[37] Torsten Blochwitz et al. "The functional mockup interface for tool independent exchange of simulation models". In: *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*. 063. 2011, pp. 105–114. DOI: 10.3384/ecp11063105.

[38] Vector Infomatik GmbH. *Testing ECUs and Networks with CANoe*. 2019. URL: https://www.vector.com/int/en/products/products-a-z/software/canoe/ (visited on 04/12/2019).

[39] Vector CANtech Inc. *Programming with CAPL*. 2004. URL: https://can-newsletter.org/assets/files/media/.../a456e3078f907a0482182ce831912427.pdf (visited on 04/24/2019).

[40] Bruce Emaus. *Quick Introduction To CAPL*. 2003. URL: https://can-newsletter.org/assets/files/media/.../a456e3078f907a0482182ce831912427.pdf (visited on 04/16/2019).

[41] DSPACE GmbH. *dSPACE Tool Chain for Virtual Validation*. URL: https://www.dspace.com/en/inc/home/products/systems/virtual_validation/viva_toolchain.cfm (visited on 06/10/2019).

[42] ETAS. *ISOLAR-EVE details*. 2019. URL: https://www.etas.com/en/products/isolar_eve-details.php (visited on 08/05/2019).

[43] Patrick E Lanigan, Priya Narasimhan, and Thomas E Fuhrman. "Experiences with a CANoe-based fault injection framework for AUTOSAR". In: *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*. 2010, pp. 569–574. DOI: 10.1109/DSN.2010.5544419.

[44] D. Kum et al. "AUTOSAR migration from existing automotive software". In: *Proc. of Int. Conf. on Cont. Aut. and Syst.* 2008, pp. 558–562.

[45] Felipe R Franco et al. “Workflow and toolchain for developing the automotive software according AUTOSAR standard at a Virtual-ECU”. In: *2016 IEEE 25th International Symposium on Industrial Electronics (ISIE)*. 2016, pp. 869–875. DOI: 10.1109/ISIE.2016.7745004.

[46] Z. Shan, K. Song, and T. Zhang. “Development of Energy Management Software Module for Fuel Cell Vehicle Based on AUTOSAR Methodology”. In: *2018 IEEE Vehicle Power and Propulsion Conference (VPPC)*. 2018, pp. 1–6. DOI: 10.1109/VPPC.2018.8605002.

[47] Sujit S. Phatak, D. J. McCune, and George Saikalis. “Cyber Physical System: A Virtual CPU Based Mechantronic Simulation”. In: *IFAC technical paper* (2010). DOI: 10.3182/20100913-3-US-2015.00077.

[48] Y. Ito et al. “VIRTUAL HILS: A Model-Based Control Software Validation Method”. In: *SAE Int. J. Passeng. Cars - Electron. Electr. Syst.* 2011, pp. 142–149. DOI: 10.4271/2011-01-1018.

[49] Sujit S. Phatak et al. *Virtual Multi-ECU High Fidelity Automotive System Simulation*. 2016. URL: https://saemobilus.sae.org/content/2016-01-0013.

[50] Oliver Sander et al. “System concept for an FPGA based real-time capable automotive ECU simulation system”. In: *Proceedings of the 22nd Annual Symposium on integrated circuits and system design*. ACM, 2009, pp. 1–6. ISBN: 9781605587059.

[51] Ahmad T. AI-Hammouri. “A comprehensive co-simulation platform for cyber-physical systems”. In: *Computer Communications* 36 (2012), pp. 8–19.

[52] Atsushi Suzuki et al. “CPS-Sim: Co-Simulation for Cyber-Physical Systems with Accurate Time Synchronization”. In: *IFAC-PapersOnLine* 51.23 (2018). 7th IFAC Workshop on Distributed Estimation and Control in Networked Systems NECSYS 2018, pp. 70–75. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2018.12.013.

[53] Abdalkarim Awad, Peter Bazan, and Reinhard German. “SGsim: Co-simulation Framework for ICT-Enabled Power Distribution Grids”. In: *Measurement, Modelling and Evaluation of Dependable Computer and Communication Systems*. Cham: Springer International Publishing, 2016, pp. 5–8. ISBN: 978-3-319-31559-1.

[54] Bernhard Thiele and Dan Henriksson. “Using the Functional Mockup Interface as an Intermediate Format in AUTOSAR Software Component Development”. In: *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*. 63. 2011, pp. 484–490.

[55] L. Exel et al. “Re-use of existing simulation models for DCS engineering via the Functional Mock-up Interface”. In: *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. Sept. 2014, pp. 1–4. DOI: 10.1109/ETFA.2014.7005261.

[56] *PyFMI 2.5*. URL: https://pypi.org/project/PyFMI/.

[57] G. Böhme and N. Réhault. “Development of a Framework Based on PyFMI for Optimization and Fault Detection”. In: 2014.

[58] AUTOSAR. *Acceptance Test Specification of Communication on CAN bus*. URL: https://www.autosar.org/fileadmin/user_upload/standards/tests/1-0/AUTOSAR_ATS_CommunicationCan.pdf (visited on 08/31/2019).

[59] Grabianowski. *How the Hughes Telematics Device Works*. 2009. URL: https://electronics.howstuffworks.com/gadgets/automotive/hughes-telematics-device.htm (visited on 04/22/2019).

[60] Wikipedia. *Telematic control unit*. 2018. URL: https://en.wikipedia.org/wiki/Telematic_control_unit (visited on 04/22/2019).

[61] New Eagle. *Raptor-Telematics*. 2018. URL: https://www.neweagle.net/support/wiki/index.php?title=Raptor-Telematics (visited on 04/25/2019).

TRITA-EECS-EX-2019:689