A FMI-based Approach for CAN Bus Simulation and Simulink Model Integration in Vehicle Simulation Environment

Hyeongrae Kim
School of Electronics and Electrical Engineering
Kyungpook National University
Daegu, South Korea
hrsin95@knu.ac.kr

Harim Lee
School of Electronics Engineering
Kyungpook National University
Daegu, South Korea
hw05165@knu.ac.kr

Jeonghun Cho
School of Electronics Engineering
Kyungpook National University
Daegu, South Korea
jcho@knu.ac.kr

***Abstract*— Automotive software is becoming more complex and diverse, making the development and verification process more important. However, the existing integrated simulation environment does not reflect the characteristics of CAN communication that can occur in the real vehicle environment. In this paper, we propose a Functional Mock-up Interface (FMI) 2.0-based approach to build an integrated simulation environment that includes a CAN bus. We design and implement a CAN Bus Simulation (CBS) FMU model that can simulate CAN communication among various models of vehicle systems, and an FMI CAN Function Module that can make Simulink models compatible with CAN communication. We also integrate the developed models into the MasterSim simulation environment and verify that the data exchange between the models is performed correctly. Our approach has several advantages over the existing methods. First, it supports various models and tools that comply with the FMI 2.0 standard. Second, it can simulate the actual operation of the CAN bus such as message transmission time and priority, and evaluate the impact of these factors on system performance. Lastly, it preserves the original structure and function of Simulink models and requires only minimal modifications to support CAN communication.**

Keywords*— *virtual ECU, Functional Mock-up Interface, Simulink, CAN simulation, automotive embedded software

I. Introduction

Recently, automotive software has become more complex and diverse due to technological innovations and consumer demand in the automotive industry, such as ADAS, autonomous driving, connectivity, etc. [1]. This software is embedded in various ECUs in the vehicle and controls and senses the vehicle. In addition, automotive software requires safety and reliability, so the software development and verification process take a lot of cost and time. Especially in the vehicle system development process, the hardware development speed is slower than the software development speed, so even if the software is developed in advance, the environment for testing it is limited. That makes it difficult to verify the software for various situations that may occur in the actual vehicle environment. To solve this problem, the introduction of virtual ECUs for vehicle environment simulation is increasing [2]. A virtual ECU is an ECU that simulates the hardware of a real ECU in software and runs on a PC. Using virtual ECUs can reduce hardware dependency and perform software development and verification efficiently.

Software operation verification in a single virtual ECU is important, but software verification in the entire system is also important. To verify the software in the entire system, data exchange between virtual ECUs and models developed externally must be considered. For this purpose, we have developed an integrated vehicle simulation environment based on FMI 2.0 that can interconnect virtual ECUs and exchange data with various models. However, the existing FMI simulation research is data exchange that does not reflect the characteristics of the actual CAN communication system. Fig. 1 shows the difference between the existing data exchange and the data exchange that reflects the characteristics of CAN communication. In the former case, direct connections are made between models that require data exchange, and data is exchanged directly when data exchange time occurs during simulation. In contrast, in the latter case, all models are connected to the CAN Bus model and exchange data through the CAN Bus. CAN Bus is a network technology that allows various ECUs inside a vehicle to exchange data and has advantages such as real-time, reliability, and low cost. However, also there are problems such as transmission delay, message collision, bit error, etc. On the CAN Bus, these problems can affect the operation of the software. Therefore, it is very important to build an integrated simulation environment that includes CAN Bus and verify the software under similar conditions to real vehicles.

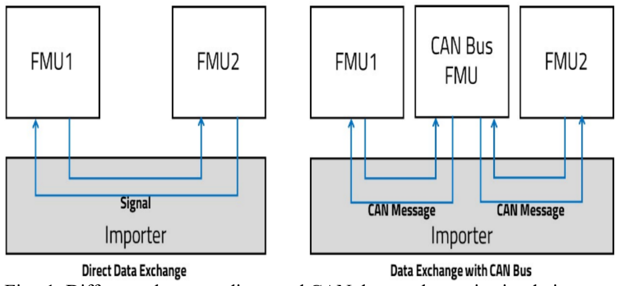


Fig. 1. Difference between direct and CAN data exchange in simulation

There have been various studies on introducing CAN Bus to simulation. [3] proposed a method to develop a vehicle CAN Bus simulation and test system using a tool called CANoe from Vector Co., Ltd. However, this method incurs a high cost for acquiring hardware and software. [4] performed a CAN Bus simulation based on queuing model to analyze problems such as transmission delay, message collision, bit error, etc. of CAN Bus. However, this simulation does not include FMI standard, so it is difficult to build an integrated simulation environment. [5] describes how to support CAN communication simulation in FMI 3.0. However, FMI 3.0 standard was announced relatively recently and there are limited open sources available for research.

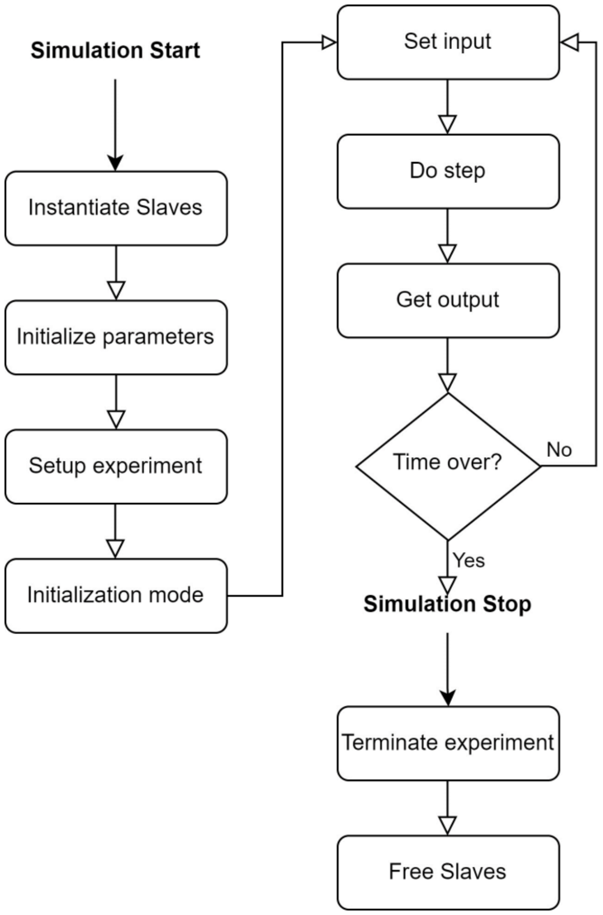


Fig. 2. Flow diagram of the FMI Master Algorithm

In this paper, we propose a module for CAN communication between various models in an integrated vehicle simulation environment called CAN Bus Simulation (CBS) FMU module and a method to make Simulink [6] model compatible with an FMI-based integrated simulation environment with CBS FMU. CBS module is an FMU based on FMI 2.0 standard that is compatible with various simulation tools and models and has high scalability and reusability. This module calculates transmission time according to the CAN message and performs arbitration to determine the priority of CAN messages to reproduce the actual operation of CAN communication. This enables simulation based on timing and actual operation of CAN messages. The Simulink model is a modeling and simulation tool provided by MATLAB that can develop and test various vehicle SWs and control algorithms. We designed FMI CAN Function Module for the Simulink model to send and receive CAN messages through FMI CAN communication. This module is located at the data input/output end of the existing Simulink module and converts data from the Simulink model into FMI CAN message to transmit through FMI CAN communication, extracts data from the message received through FMI CAN communication, and delivers it as input data to the model. This method enables smooth data exchange between CBS FMU and Simulink model.

This paper covers the following chapters. First, in Section 2, we describe the background knowledge related to this research, and in Section 3, we introduce the concept, structure, and function of the CBS FMU that we propose. In Section 4, we propose a method to make the Simulink model compatible with CAN communication-based FMI integrated simulation environment. In Section 5, we explain how to integrate the developed models into the simulation environment and the settings. In Section 6, we conclude with future works.

II. Background

A. Functional Mock-up Interface (FMI)

FMI [7] is a standardized interface that provides compatibility between models developed in different tools and supports integrated simulation. FMI provides a container in the form of a ZIP file consisting of XML files, binary files, C code, and an interface defined in C language. The container of the ZIP file is called a Functional Mock-up Unit (FMU). In the integrated simulation environment, FMUs are imported and the initial values of the variables defined in each FMU are set and connected with other FMUs before performing the integrated simulation. Recently, most modeling tools, including Simulink, provide the function to export models as FMUs. FMUs consist of models in the form of differential equations and solvers for numerical analysis. During simulation, they receive inputs at agreed times and use solvers to analyze the models and output the results.

FMI supports two modes: Model Exchange (ME) and Co-Simulation (CS). In the former case, the FMU does not have a solver, but instead provides information such as variables, models, parameters, etc. to the importer or master. Then, the importer uses its solver to analyze the model of the FMU. Therefore, all FMUs operate according to the time step of the import. In the latter case, the FMU has a solver and receives input from the importer and analyzes the model with its solver, and sends output at regular intervals. Which mode to use depends on the situation. Generally, ME mode is suitable for cases where accuracy is important, and CS mode is suitable for cases where speed and security are important. In our project, since we ultimately construct an integrated simulation environment for virtual ECUs as targets, we use CS mode. Virtual ECUs operate with their own timers and exchange data according to the time set by the importer, so CS mode is suitable.

The operation of FMI usually proceeds according to the master algorithm shown in Fig. 2. First, all imported FMUs are instantiated and set initial values embedded in each FMU before starting a simulation. The communication point is determined at the boundary of the time step set by the importer when data exchange occurs during simulation. When the communication point occurs, the importer calls fmi2SetData to update the variables of each FMU with data stored from the previous time step. Then it calls fmi2doStep of each FMU to proceed with simulation and then calls fmi2GetData to store the output. At this time, Data is called by the importer according to variables defined in a modelDescription.xml of each FMU. This operation is repeated until the simulation time is set before the simulation ends.

In this paper, CBS FMU is implemented using FMU SDK [8]. FMU SDK is a free software development kit provided by Qtronic that provides functionality to create FMUs based on C language. FMU SDK supports both FMI 1.0 and FMI 2.0 standards. To create an FMU using FMU SDK, you need to write C language source code and model description XML file that defines variables. C language is mostly provided as a

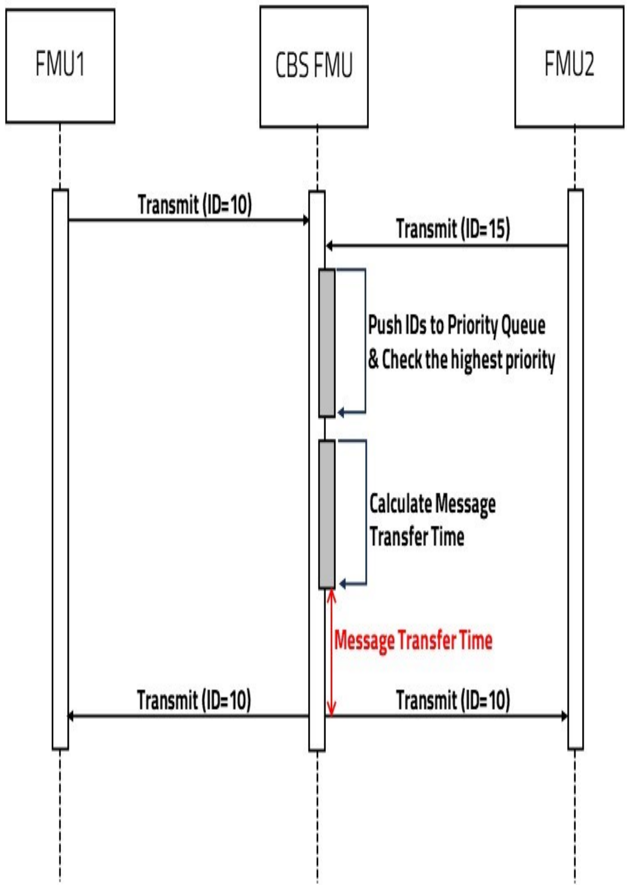


Fig. 3. Data flow in CBS FMU-based CAN communication simulation

template by FMI, and users can add their behavior and write it.

B. *Simulink Model Export*

Simulink is a modeling and simulation tool that works in MATLAB environment and is widely used as a tool for visualizing and modeling complex system behavior. It provides a graphic diagram-based interface that allows intuitive modeling and simulation of system dynamics. Simulink allows you to represent system behavior by combining blocks and lines, and through this you can simulate system behavior from the initial design stage to predict and analyze actual behavior. And you can extract this system as an FMU, making it easier to create an FMU.

Simulink includes libraries and tools that support system modeling in various fields, and you can also bring in additional libraries and tools. In addition, you can create blocks that define behavior in C language as well as MATLAB language, so you can create blocks that are not supported or optimize existing blocks to suit your system.

C. *MasterSim*

MasterSim [9] is an FMI integrated simulation environment that supports FMI Co-Simulation 1.0 and 2.0. This tool sets initial values for simulation, imports FMUs, connects them and performs integrated simulation. MasterSim consists of GUI settings and command line simulations. In GUI settings, you can import FMU slaves and connect input/output variables. At this time, you can control the data flow required for simulation by connecting input/output variables graphically. In addition, you can set simulation options such as simulation start time, end time, master algorithm, logging, etc. The options set through GUI settings are saved as scripts in project files. Command line simulation loads project files created by GUI and executes simulation as defined in project files. Simulation results are saved as CSV files.

III. CAN Bus Simulation FMU

The data transmission flow between models, when CBS is present, is shown in Fig. 3. FMU1 sends a CAN message to CBS FMU for CAN transmission. The CBS FMU that received the CAN message performs two operations. First, it compares the ID of the received CAN message and the previous messages in the buffer to compare the priority. If the priority of the received message is the highest, it decides to send the message. After deciding which message to send, it calculates the transmission time of the message and sends it to all connected FMUs. The FMU2 that received the message confirms that it is a message to be received through CAN ID filtering and receives it.

A. *CAN Message Transfer Time*

The calculation of the transmission time of the CAN message (CMTT) is as follows in equation (1). 1 Time Quanta (TQ) is obtained by taking the reciprocal of Clock Frequency. By multiplying this value with Bit Time, which is the transmission time of one bit of CAN data, and the entire size (bit) of the CAN message, one can calculate the transmission time of the CAN message. At this time, Clock Frequency and Bit Time are defined as variables of CBS FMU so that simulation users can set initial values. The variable parts of the CAN message are the ID field with 11 bits in version 2.0A and 29 bits in version 2.0B, and the Data field with up to 64 bits depending on the value of the Data Length Code (DLC) field. Since there are two versions, if you define version information as Boolean-type data, only data length depending on the DLC bit remains as a variable parameter. Therefore, when data comes in, read DLC and add the bit size of the fixed area and bit size of the data area to get the total CAN message size and multiply it by 1TQ and Bit Time to calculate transmission time. CBS FMU stores data from Source FMU internally and sends data to the Destination at the communication point closest to the calculated transmission time.

$$CMTT = 1TQ * Bit\ Time * Size\ of\ CAN\ Message \quad (1)$$

B. *CAN Message Arbitration*

CAN Arbitration is a method to determine the priority of messages and avoid collisions on the CAN bus. Actual CAN performs bit-by-bit arbitration. When multiple nodes try to send messages on the CAN bus at the same time, each node compares the ID of the message the are sending with a signal on the bus bit by bit. The ID of the message has higher as the

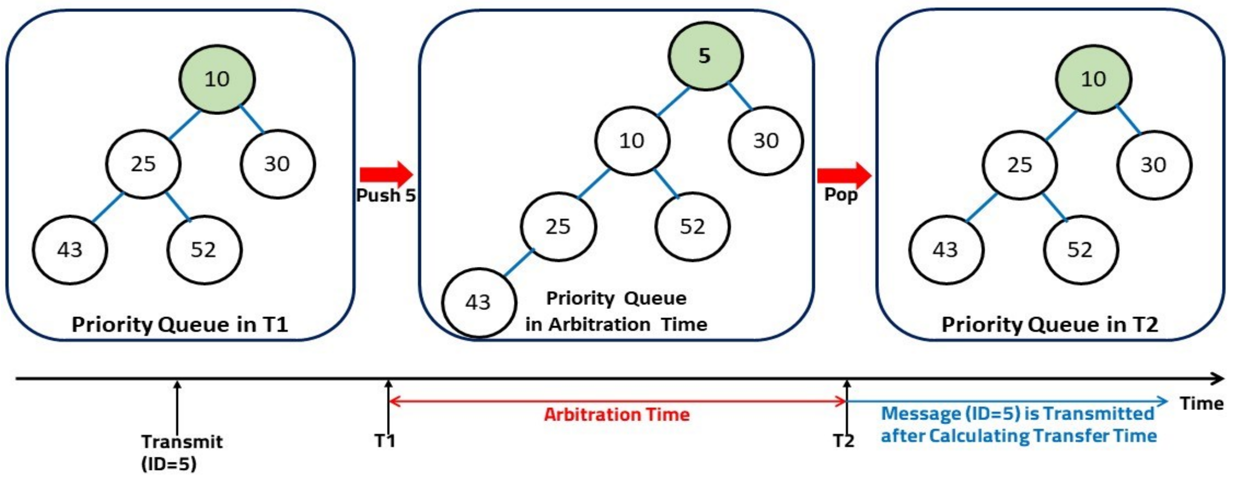


Fig. 4. Arbitration using the priority queue

value is smaller. If the node is transmitting recessive bit (1) and detects dominant bit (0) on the bus, it determines that its priority is low and stops or postpones transmission. On the contrary, if the node is transmitting the dominant bit and detects the dominant bit on the bus, it determines that its priority is high or equal and continues transmission. In this way, only messages with the highest priority can occupy the bus and be fully transmitted.

In the integrated simulation, data exchange between CAN communication is done by message unit, so it is difficult to implement bit-by-bit arbitration. Therefore, we propose a method to perform arbitration by message unit. Fig. 4 shows an arbitration we propose. This method puts a buffer in CBS FMU and stores all CAN messages from FMUs that want to transmit at the communication step in the buffer. Currently, the buffer is implemented as a priority queue with Min Heap. Since a smaller ID means higher priority for the CAN message, if pushed into the priority queue with Min Heap, higher priority means closer to the head of the queue. A message with the highest priority is popped at that time step and sent after calculating the transmission time. Messages that were pushed back in priority remain in the queue. After transmission, when the next time step comes, compare priority with new messages coming in and determine the order of transmission.

C. CBS FMU using FMU SDK

We implemented CBS FMU using FMU SDK. First, we defined variables in the modelDescription.xml file. Table 1 shows a list of variables defined in modelDescript.xml. Name means the name of the variable, and Type means the data type of the variable. Data types supported by FMI 2.0 include Real, Integer, Boolean, String, and Enum types. Causality is a property that defines how a variable interacts with the outside of the model. There is a Parameter whose value does not change after initialization, Input, which is a variable that receives a value from outside the model, and Output, which is a variable that sends a value to the outside of the model. Variability is a property that defines how the value of a variable changes over time. It is defined in the Continuous-time domain and Discrete in the discrete-time domain. A constant value is determined before model initialization and the others are determined after model initialization. There are Clock Frequency and Bit Time for calculating message transmission time, and they are defined as Real and Integer respectively. A version variable of Boolean type determines whether it is CAN 2.0A or 2.0B. We declared ID indicating the priority of the CAN message, DLC indicating the data length of the CAN message and actual data. Since the Integer variable of FMI 2.0 is 32 bits, we declared two variables Data1 and Data2 to represent all 8 bytes of data. Next, we wrote the behavior that CBS FMU will perform in the fmi2doStep function in the C file. If there is an update of the FMU variable, push the ID variable of the corresponding value into the priority queue, pop data from the queue, and determine the message with the highest priority. If the CBS FMU finds a message with the highest priority, change the state variable to BUSY, read the DLC of corresponding data, calculate message transmission time, and send corresponding data at the communication point closest to this time. After calculating message transmission time, even if higher priority data comes in before sending time, CBS FMU is already in a BUSY state, so the message goes into the priority queue and gets an opportunity to be sent after the transmission is over and state variable of CBS FMU becomes an IDLE state.

TABLE I. CBS FMU VARIABLE LIST

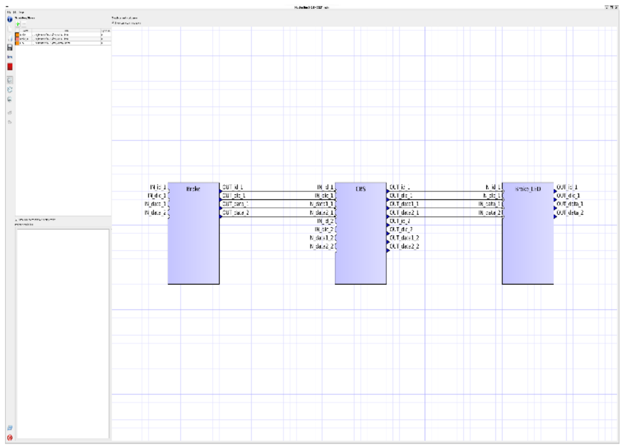
Variable List in modelDescription.xml			
Name	***Type***	***Causality***	***Variability***
Clock Frequency	Real	Parameter	Constant
Bit Time	Integer	Parameter	Constant
Version	Boolean	Parameter	Constant
IN_ID	Integer	Input	Discrete
IN_DLC	Integer	Input	Discrete
IN_Data1	Integer	Input	Discrete
IN_Data2	Integer	Input	Discrete
OUT_ID	Integer	Output	Discrete
OUT_DLC	Integer	Output	Discrete
OUT_Data1	Integer	Output	Discrete
OUT_Data2	Integer	output	Discrete

```
/* CAN ID filtering function */
function IdFilter(id, array, arraySize)
  r = 1
/* array contains ID value to be received */
  for index = 0 to arraySize - 1
    if id == array[index] then
      r = 0
      break
    end if
  end for

  return r
end function

/* Merge Data from FMI */
function MergeData(data1, data2, mergedData)
  mergedData = 0
  mergedData |= (data1 << 32)
  mergedData |= data2
end function

/* Split Data for FMI */
function SplitData(outputData, data1, data2)
  data1 = (outputData >> 32)
  data2 = (outputData & 0xFFFF)
end function
```

Fig. 5. Pseudo-code of FMI CAN Function Module

CAN Communication Simulink Model

The existing Simulink model exchanges data itself, and the data received through CBS FMU is delivered to all nodes, so it cannot communicate properly with CAN if it uses the received message as it is. We designed the FMI CAN Function Module to perform FMI CAN communication simulation with CBS while maintaining the original Simulink model. We will add a custom block using the pseudocode shown in Fig. 5 to make the existing Simulink model compatible. The data that comes in as input first passes through the CAN ID Filter function. In this function, it compares the ID of the incoming message with the predefined ID list, and if it matches, it recognizes that it is its own data and moves on to the next function, otherwise, it stops processing the message. If it confirms that it is a message to be processed, it goes through the process of extracting data from the message. Since FMI 2.0 does not support array variables, the data of the message received from CBS FMU is stored in two 32-bit integer data variables, and the Merge Data block connects these two data variables to make one 64-bit integer data. The model uses this



6. CBS FMU and Simulink models with FMI CAN Function Module integrated into MasterSim

processed data as an input variable. Conversely, when the Simulink model outputs data, it goes through the Split Data block and makes two 32-bit integer data. After passing through Generate Message block, the generated CAN message is sent to CBS FMU.

IV. Integration Test of the Developed Models in the Simulation Environment

We integrated CBS FMU and improved the Simulink model into the simulation environment to perform integrated simulations. The integrated simulation environment is based on MasterSim. We confirmed that FMU was imported without error when importing CBS FMU and FMU extracted from the Simulink model in MasterSim and confirmed that the input/output connection between each model was done normally. Fig. 6 shows how FMUs are imported in MasterSim. We have not defined the exact model yet, so we configured the simulation setting with a default value and confirmed that CAN message transmission and reception were done.

V. Conclusion

In this paper, we proposed an FMI 2.0-based approach for CAN bus simulation and Simulink model integration in a vehicle simulation environment. We designed and implemented the CAN Bus Simulation (CBS) FMU model that can simulate CAN communication between various models of vehicle systems and the FMI CAN Function module that can make the Simulink model compatible with CAN communication. In addition, we integrated the developed models into MasterSim simulation environment and confirmed that data exchange between models was performed correctly.

Our approach has several advantages over existing methods. First, it can support various models and tools that comply with FMI 2.0 standard. Second, it can simulate actual operation of the CAN bus such as message transmission time, priority, etc., and evaluate the impact of these factors on system performance. Lastly, it can preserve the original structure and function of the Simulink model and requires minimal modification to support CAN communication.

Our research will solve the following problems in future work. First, limitations of FMI 2.0 for CAN. FMI 2.0 does not support array variables, so data is divided into two 32-bit integer variables for simulation. This increases unnecessary operations. Also, the communication point in FMI 2.0 is static once determined, so the accuracy of message transmission time may be low even if message transmission time is calculated. On the other hand, FMI 3.0 supports the array variable type and it can adjust the length of the communication step during simulation. These problems should be solved first when FMI 3.0 becomes popular. In addition, as we target SW development and verification through virtual ECU, we will integrate virtual ECU that operates similarly to actual ECU into an integrated simulation environment and obtain meaningful CAN simulation results.

ACKNOWLEDGMENT

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No. 1711160343, Development of virtual ECU-based vehicle-level integrated simulation technology for vehicle ECU application software development and verification automation)

REFERENCES

[1] S. K. Anjum and C. Wolff, "Agile Principles in Automotive Software Development: Analysis of Potential Levers," 2021 IEEE European Technology and Engineering Management Summit (E-TEMS), Dortmund, Germany, 2021, pp. 141-147, doi: 10.1109/E-TEMS51171.2021.9524860.B.-G. Moon, C.-W. Lee, and H.-J. Shim, “Global Automobile Recall Trends and Implications for Us,” Auto Journal of The Korean Society Of Automotive Engineers, 2014.

[2] Chrisofakis, Emmanuel & Junghanns, Andreas & Kehrer, Christian & Rink, Anton. (2011). Simulation-based development of automotive control software with Modelica. 1-7. 10.3384/ecp110631.

[3] F. Zhou, S. Li and X. Hou, "Development method of simulation and test system for vehicle body CAN bus based on CANoe," 2008 7th World Congress on Intelligent Control and Automation, Chongqing, China, 2008, pp. 7515-7519, doi: 10.1109/WCICA.2008.4594092.

[4] Zhang, J., Li, T. (2013). Based on the Queuing Model of CAN Bus Simulation and Application. In: Yin, Z., Pan, L., Fang, X. (eds) Proceedings of The Eighth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA), 2013. Advances in Intelligent Systems and Computing, vol 212. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-37502-6_76

[5] Modelica Association. (2023). FMI Library Set for Bus Communication [online]. Available: https://modelica.github.io/fmi-ls-bus/main/

[6] Klee, H., & Allen, R. (2017). Simulation of Dynamic Systems with MATLAB® and Simulink® (3rd ed.). CRC Press. https://doi.org/10.1201/9781315154176

[7] "Functional Mock-up Interface for Model Exchange and Co-Simulation", *2.0 Release Candidate 1*, October 2013, [online] Available: https://www.fmi-standard.org.

[8] *Qtronic*. (2021). FMU SDK free software development kit. [online] Available: https://github.com/qtronic/fmusdk

[9] Bauklimatik Dresden. (2022). MasterSim User Manual. [Online]. Available:https://bauklimatik-dresden.de/mfiguregFastersim/help/MasterSim_manual_en.html