[Industrial Paper] Performance Measurement and Finding Challenges in Using FMUs to Perform Scenario-Based Testing in a Cloud Environment

Katsuya Tsuzuki[1] Takashi Yamada[1] Kensuke Araki[1]

[1] dSPACE Japan K.K., Japan, {ktsuzuki, tyamada, karaki}@dspace.jp

Abstract

This paper describes the implementation of the scenario-based testing, a test method for autonomous driving software, by coupling a plant model described using MATLAB/Simulink with another plant model provided as functional mock-up unit (FMU) on a cloud platform. During the implementation of plant models into the cloud environment using the functional mock-up interface (FMI), there are problems and countermeasure challenges were identified. In addition, the impacts of integrating multiple models on simulation time by parallelizing test cases are measured.

Keywords: Model-based Development, Cloud computing, Scenario- Based Testing

1 Introduction

In recent years, the automotive industry has been at a turning point that is characterized by two trends. One is the trend toward electrification, and the other is intellectualization (Yamada, 2020; dSPACE Japan, 2013). The trend for intellectualization is a shift of technical development from advanced driving assistance systems (ADAS), such as automatic emergency brake (AEB) systems, to autonomous driving (AD) systems.

As vehicles become more electrified and intelligent, an in-vehicle system for controlling components in the vehicle gets more complicated. Developing components across multiple domains, including electrical, mechanical, fluid, and control systems, becomes essential, and there was an issue with interfacing components from multiple domains on a simulation platform. FMI, a standard proposed by the Modelica Association, solves this problem.

There is a solid demand for applying model-based development (MBD) to AD system development for efficient development and validation. The PEGASUS project, a public-private project funded by the German federal government, discussed what validation environment and methods for AD systems should be. As a result, a testing method using MBD tools was proposed in the project, what was called “scenario-based testing” (PEGASUS project, 2019).

Scenario-based testing is a test method characterized by automatic new test case generation based on past test results. It is helpful to efficiently perform AD system validation with a limited number of test workloads by automatically finding test parameters that should be intensively tested. Scenario-based testing was also proposed in ISO21448:2022 Road vehicle Safety of the intended functionality (SOTIF) for AD software safety functionality, which was published in June 2022 (ISO, 2022).

Compared to validating a conventional real-time control system such as an internal combustion engine control system, the number of test cases for AD system validation is still large, even when the scenario-based testing method is applied. An AD system validation environment needs to manage such a large number of tests.

People expect the adaption of a cloud computing environment to be one of the solutions to this challenge. In cloud computing, users access computation resources via the Internet. On a cloud computing platform, tests can be performed in parallel while flexibly increasing or decreasing required computing resources depending on the number / scale of the tests.

However, there are few examples using cloud computing for system validation using MBD tools, although Yamada, Araki, and Tsuzuki reported some cases at JSAE 2022 Spring and Autumn Congress (Yamada *et al.*, 2022; Araki *et al.*, 2022).

Based on this background, this paper summarizes the challenges faced in integrating multiple plant models using a FMU to realize a prototype system for AD system verification by scenario-based testing in a cloud environment. In addition, we carry out several scenario-based tests in the environment, and the simulation times are measured to compare execution time with and without the FMU. Furthermore, we parallelize the execution of several test cases, compare the impact of parallelization on the simulation time and discuss how FMU affects execution time when test cases are parallelized.

This paper is organized as follows. Section 2 describes the experimental test system and the models to be simulated. Section 3 and Section 4 describes the experiment results of the test system execution described in Section 2, and the results obtained from the experiments, respectively. Section 5 lists challenges identified during the experiments performed in Section 3 and discusses how we can solve challenges. Section 6 concludes the paper with limitations and future works.

2 Experimental Environment

An experimental environment for validating automated driving algorithms using FMU in a cloud environment was built and tested in the following three steps.

Our goal in this section is to integrate the plant model with a FMU to a cloud environment, which is suitable for scenario-based testing. Section 2.1 describes a plant model used in simulation and how it is integrated to an on-premise simulation environment on Microsoft Windows, which was commonly used to validate a conventional system so far. In section 2.2 we explain how the plant model, designed to be used on Windows, is ported to a Linux environment, which is an operating system commonly used in a cloud platform. Finally, in section 2.3 we describe how we integrate the ported plant model into a cloud simulation environment.

2.1 Plant Model on a Windows PC

The plant model used for the experiments was prepared in an on-premise Windows PC environment, which is commonly used today. The simulation model was prepared based on ASM Traffic, one of the packages in the dSPACE Automotive Simulation Models (ASM). ASM Traffic is a model designed to simulate roads, traffic, buildings, and in-vehicle sensors around vehicles, and is used for validation of AD and ADAS electric control units (ECUs).

This time, ASM in dSPACE Release 2021-A was modified in MATLAB / Simulink R2020a to remove blocks related to aerodynamics. Functionality corresponding to the removed aerodynamic block was implemented in OpenModelica 1.19.2 and output as an FMU. The output FMU conforms to FMI 2.0 co-simulation.

The ASM-modified model and the model created in OpenModelica 1.19.2 were integrated using dSPACE VEOS, a simulation platform that enables software-in-the-loop (SIL) simulation on a Windows PC.

As a controller model, which is a model that should be validated in the SIL simulation system, we used Soft ECU in this experiment, a simple controller model that comes with ASM.

Figure 1 shows a schematic of the plant model built on a Windows PC. Figure 2 shows a screenshot of dSPACE VEOS Player when the test environment is being built. It can be seen that the I/O from the controller model and the plant model are inter-connected.

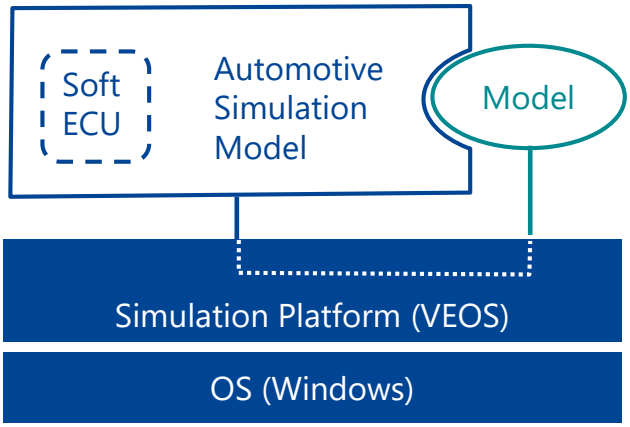

Figure 1 An overview of Windows simulation environment.

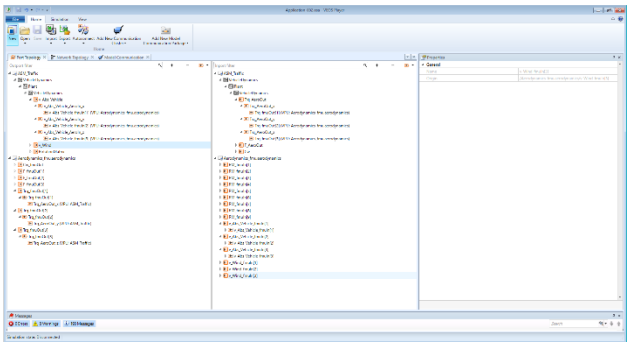

Figure 2 An example of simulation configuration in an on-premise environment with dSPACE VEOS Player.

2.2 Cross Compiling Models for Linux Environment

Linux is the most common operating system for cloud computing environments. In order to integrate the plant model in a cloud environment, it was necessary to cross-compile the model to a Linux environment.

Simulink, ASM, and VEOS support execution on Linux, and a FMU generated from OpenModelica also claims to support Linux. Cross-compilation of the plant model to a Linux environment was performed without major problems. However, since operating system is different, it was necessary to check whether there is any difference in simulation results between the Windows environment and the Linux environment.

In this paper, Ubuntu 18.04LTS was used as Linux. Figure 3 shows the plant model that was built on a Linux environment.

DOI
10.3384/ecp193141

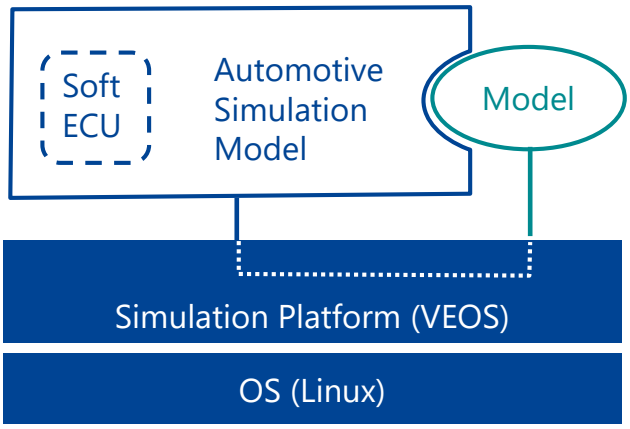

Figure 3 An overview of Linux simulation environment.

2.3 Plant Model Integration in Cloud Environment

In the cloud environment, we used dSPACE SIMPHERA version 22.4, which is an environment for SIL simulations running on various cloud platforms such as Microsoft Azure. Taking advantage of the features of cloud computing, which can secure computing resources in a scalable and flexible manner, the tests are distributed and executed in parallel on multiple computation nodes, enabling the execution of a huge number of tests, such as those for autonomous driving applications. Figure 4 illustrates an overview of simulation environment on a cloud.

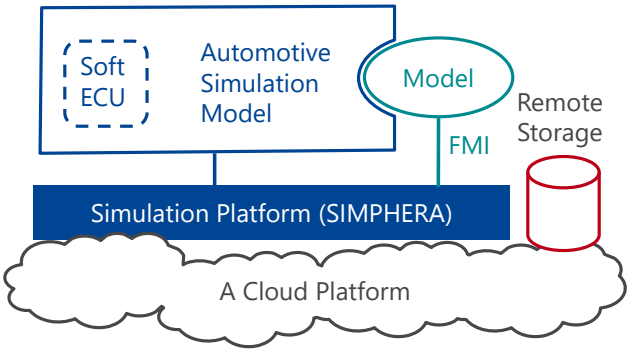

Figure 4 An overview of the simulation environment in the cloud.

SIMPHERA is developed as a Kubernetes cluster in a cloud environment. Kubernetes is an open source container orchestration software that is widely used. Container orchestration is automation of container deployment, management, scaling, and networking.

The SIMPHERA system is composed of open source software (OSS) built on a Kubernetes cluster and software from dSPACE. As OSS, MinIO, an object storage server compatible with Amazon S3 cloud storage service, is used for storage management, PosgreSQL, a relational database management system, is used as a database for input and result data, and Keycloak, personal authentication and access management software, is used for WebUI login management.

On the other hand, dSPACE software includes SIMPHERA execution agent, which is equivalent to an application execution unit that executes jobs for each parameter to compute scenario-based tests in parallel.

dSPACE VEOS runs as simulation software for scenario-based testing in SIMPHERA execution agent. The SIMPHERA system configuration diagram is shown in Figure 5.

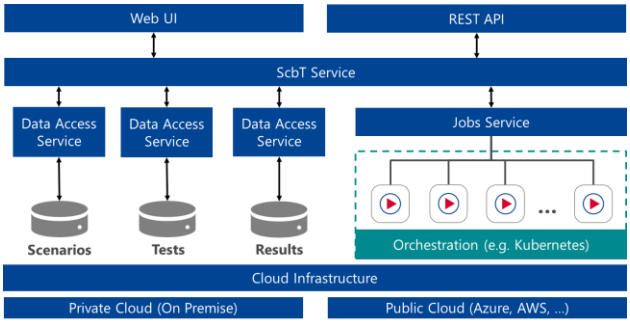

Figure 5 System overview of dSPACE SIMPHERA

In this study, the private cloud environment shown in Table 1 was prepared and SIMPHERA was built on this private cloud environment.

Table 1 Specifications of Private Cloud.

	loud Specification
CPU	28 logical cores
Memory	48GB
Storage	512GB
OS	Ubuntu 20.04LTS

3 Scenario Definition

Using the experimental environment in which the plant model was implemented in SIMPERA on the cloud environment as described in Section 2.3, simulations in the scenario-based test were performed with different parallelism levels for models with and without FMU, and the relationship between parallelism and execution time was measured.

The scenario used in the scenario-based test was the United Nations Economic Commission for Europe (UNECE) Automatic Lane Keeping System (ALKS) Cut-In Driver. This is a validation scenario for the

ALKS when another vehicle, traveling in adjacent lane, changes to the own lane.

In Figure 6, an illustration of the UNECE ALKS Cut-In Driver scenario and the definition of each parameter are shown.

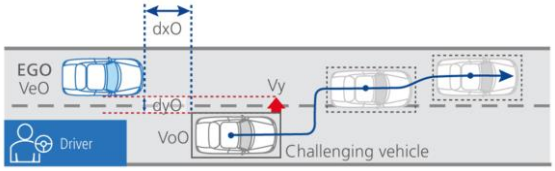

Figure 6 UNECE ALKS Cut-In Driver Scenario and Parameters.

In this experiment, the speed of the ego vehicle (Ve0) was varied from 50 km/h to 51 km/h for all two cases of testing. The scenario parameters used in the experiment are shown in Table 2.

Table 2 Scenario, parameters.

Scenario	UNECE ALKS Cut-In Driver
Parameter variation type	Cross Variation
dx0	11 m (Fix)
Ve0	50 ~ 51 km/h (Δ = 1 km/h)
Vy	3 m/s (Fix)

In SIMPHERA, the WebGUI shown in Figure 7 was used to implement visually easy-to-understand settings for the above parameters.

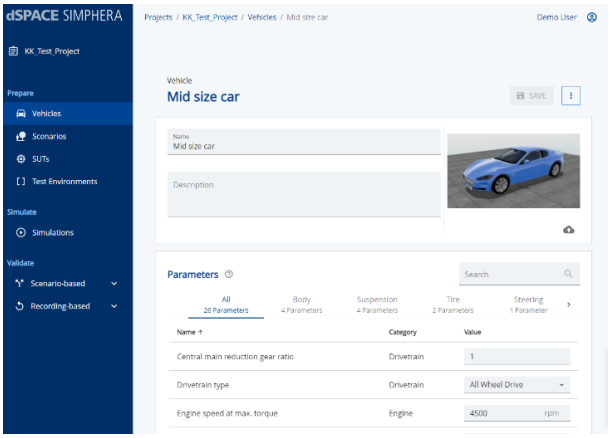

Figure 7 An example of simulation configuration in a cloud environment with dSPACE SIMPHERA.

4 Experimental Results

Simulations were performed on SIMPHERA for a model linking ASM and FMU and a model of ASM alone, changing the parallelism from one to two parallel jobs for all two cases of scenario-based testing, respectively, and the execution times were measured.

Parallelism is the maximum number of jobs that can be executed simultaneously on SIMPHERA and is one of the parameters that can be given to the execution environment by the user.

When the parallelism level was set to 1, the execution time of the ASM stand-alone model was 68.7 [sec], while the execution time of the model linking the ASM and FMU was 70 [sec]. When the parallelism was set to 2, the execution times were 35.7[sec] and 36.3[sec], respectively. Figure 8 shows the experimental results.

The results show that the execution times of the ASM stand-alone model and the model linking the ASM and FMU were almost the same.

In addition, when the parallelism level was changed from 1 to 2, the computation time was inversely proportional to the parallelism level and almost halved. Experimental results are shown in Figure 8.

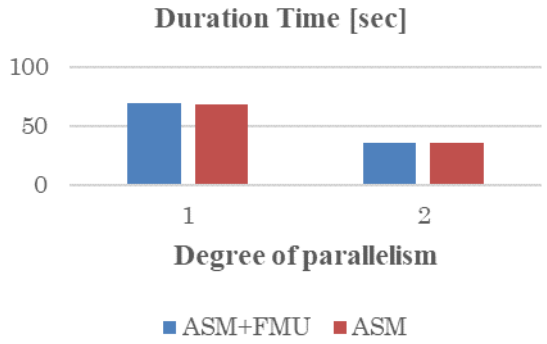

Figure 8 Duration time depending on the degree of parallelism

In the case of the model in which ASM and FMU are linked, the computation time may increase compared to the ASM stand-alone model due to overhead caused by communication between cores and overhead caused by communication between ASM and FMU, etc.

However, since the computation time for the overhead has not increased even if the parallelism level is increased The overhead due to the parallelism and communication between the ASM and the FMU are

DOI
10.3384/ecp193141

confirmed as not increased as the overhead due to the communication between the ASM and the FMU.

The reasons for the almost no difference in computation time between the model linking ASM and FMU and ASM can be attributed to the fact that the FMU model created and used for this study is stand-alone and lightweight, and that the overhead in linking the ASM and FMU models on dSPACE VEOS used for the simulation was small. The reasons for the difference are considered to be the following.

5 Challenges Identified during the Experiment

During the experimental environments and its execution, we identified some challenges. In this section we attempt to list up the challenges faced to the trial.

5.1 Challenges on Operating System Migration

Prior to the plant model construction on a Windows PC described in section 2.1, we had created an FMU using OpenModelica 1.18.1. However, when the FMU output from OpenModelica 1.18.1 was to be run in a Linux environment, additional software libraries specific to the Linux environment had to be installed.

The software libraries that needed to be installed additionally are, for example, liblapack.so. Other software libraries that were additionally required are shown in Talbe3.

Table 3 External libraries referenced by FMU.

External Libraries
liblapack.so.3
libblas.so.3
libgfortran.so.5
libquadmath.so.0

However, this issue has been resolved in OpenModelica 1.19.2, and additional libraries are pre-installed in the FMU. In this experimental environment, there was no need to install additional software libraries. In an on-premise environment, the installation of libraries may not be so much of a problem because the person who performs the simulation often has administrative privileges, but in a cloud environment, where many users are supposed to use the system, it is rare for the user to have administrative privileges, and it is difficult to install additional software libraries. Additional installation of software libraries is difficult and should be done with caution.

In addition, non-compatibility of measurement and calibration software with Linux may also be an issue. Software for measurement and on-line calibration of simulation results after the simulation environment has been built also requires attention. Although many companies provide measurement and calibration software that can be used on-premise, most of them are Windows versions, and the same software may not be available for Linux and cloud environments. On the other hand, some software, such as dSPACE ControlDesk, is designed to work with simulation environments in the cloud.

5.2 Challenges arising from the use of FMUs

In this study environment construction, we faced some issues specific to the use of FMI/FMU that are likely to occur regardless of whether the system is used in the cloud or not. Many of these are described in the FMI guidelines published by the Society of Automotive Engineers of Japan (JSAE), but are listed again in this section. The main issues faced were that "the FMU end time setting is contained within the FMU" and "a bus cannot be specified as the interface of the FMU.

First, the FMU end time settings are contained in the FMU, and any attempt to modify them requires direct editing of the files contained in the FMU. Otherwise, the operation of the FMU may be stopped unintentionally by the user, independent of the operation on the simulation platform.

Next, it is noted that buses cannot be specified for FMU interfaces; only arrays defined by a scalar value or a set of scalar values of the same type can be used to describe interfaces between FMUs and their externals. The interface between the FMU and the external model can use variables of various types, preferably in the form of structures in high-level programming languages or buses in Simulink.

However, the FMI2.0 specification used in this study does not support the connection between FMU and external models using these types and buses, and it is necessary to separate signal lines that are grouped into buses and connect them one by one. This leads to an increase in man-hours required to connect models, especially when connecting huge models. Regarding this issue, it is highly likely that FMI3.0 solves this problem; future research on experiments using FMI3.0 is expected.

6 Concluding Remarks

In this study, we performed scenario-based test jobs for validating autonomous driving algorithms using FMU in a cloud environment with different degrees of parallelism, evaluated the performance of parallel computation, and compared the computation time with and without model linkage between ASM and FMU.

The results show that the calculation speed increases with the degree of parallelism even when the model linking ASM and FMU is used. In addition, we examined the point where there is no significant difference in computation time between ASM and FMU-linked models and ASM and the reasons for this difference.

6.1 Future Works

In this study, we evaluated the performance of scenario-based tests for validating automated driving algorithms in a private cloud environment by varying the parallelism of the jobs. Since we have not yet conducted parallel computation in a public cloud environment, where computing resources can be flexibly changed, it remains to be seen whether the trend of results differs between the two environments and whether the bottleneck changes.

Also, we evaluated the performance of the scenario-based test for validating the automatic driving algorithm using FMUs when the test was run in parallel.

We compared the computation time with and without FMU collaboration. In this study we used a single FMU and a small-scale model. What will happen when multiple or large-scale FMUs are used, and how we dealing with them will be the subject of future research.

References

K. Araki, K. Tsuzuki, and T. Yamada (2022): Simulation Performance for Scenario-based Testing in a Cloud Environment, *2022 JSAE Annual Congress Autumn* (written in Japanese).

dSPACE Japan (2013): Model-based Development, Nikkei BP (written in Japanese).

ISO (2022): ISO21448:2022 Road vehicles -Safety of the intended functionality, available at https://www.iso.org/standard/77490.html, accessed on 19 Sept 2022.

PEGASUS Project (2019): PEGASUS Method – an Overview, available at https://www.pegasusprojekt.de/en/home, accessed on 19 Sept 2022.

T. Yamada (2020): Model-based Development, *Technology Roadmap 2021-2030 Automotive and Energy*, Nikkei BP, p. p. 236 – 239 (written in Japanese).

T. Yamada, K. Araki, and K. Tsuzuki (2022): Challenges and Countermeasures in Using FMUs to Perform Scenario-based Testing, *2022 JSAE Annual Congress Spring* (written in Japanese).

DOI
10.3384/ecp193141