Hyundai Framework for Vehicle Dynamics Engineering based on Modelica and FMI

Kwang Chan Ko[1] Erik Durling[2] Jong Chan Park[1] Wonyul Kang[3] Johan Andreasson[2]

[1]Commercial Vehicle CAE research lab., Hyundai Motor Group, Korea, {kcko,impactpark}@hyundai.com
[2]Modelon, Sweden, {erik.durling,johan.andreasson}@modelon.com
[3]Institute of Vehicle Engineering, Korea, wykang@ivh.co.kr

Abstract

This paper describes a framework for systems engineering with primary application in the field of vehicle dynamics, addressing the need to be able to broadly deploy models to accelerate innovation and design. The framework is based on open standards Modelica, FMI and SSP.

Keywords: Vehicle Dynamics, Modelica, FMI, SSP

1 Introduction

This paper describes a framework for vehicle dynamics engineering with focus on evaluation of ride and handling. Especially the following key requirements are addressed:

1. Scalability: The framework should be able to handle both subsystems such as suspensions, and complete vehicle configurations
2. Multi-fidelity: The framework should allow for models of different fidelity to be combined and executed together
3. Deployment: The framework should support broad usage among engineers, including non-simulation experts
4. Future-proof: The framework should be designed with future scope extensions in mind; control design, system integration and applications to other vehicle types such as passenger cars.

Similar requirements are found in several other applications, e.g. (Andreasson, 2016), (Henningsson, 2014), (Sundström, 2016a), (Sundström, 2016b).

2 Framework approach

The approach was to define a framework to allow for separation of different types of engineering work, and engineers skills according to but not limited to the following categories:

1. Model execution: Most engineers are using the models to support engineering decisions. Their need is to configure the vehicle/subsystem, chose/edit data sets, execute analysis and post-process the results. All with a high degree of automatization. Their background in simulation is typically limited and there is limited need to see details in the models. They prefer to work in a streamlined environment that they are familiar to. If they run into problems with simulations, they want to get help rather than solving it themselves.
2. Model authoring: A limited number of engineers need to make new models and change existing ones, they also execute very specialized analysis. They have stronger background in simulation, and typically have experience in using simulation tools. They typically want to have great flexibility and have less need for streamlined workflow. These people are expected to help engineers working with Model execution.
3. System integration: Engineers working with system integration may need to edit how models from different suppliers are connected, but rarely need to see inside each model for details. They need some more flexibility than those working with Model execution, but less compared to those working with Model authoring.

The implementation of the framework is carried out in a sequence of steps with gradually expanded capabilities:

1. Execution of fixed configurations, Figure 1.
2. Execution of reconfigurable models, Figure 2.
3. System integration, Figure 3.

2.1 Execution of fixed configurations

Figure 1 illustrates a common framework to manage the separation of engineering task assuming that model execution does not involve system reconfiguration. System reconfiguration typically means replacing a suspension, change the number of suspensions, or any other change that change the topology of the model.

In this case, the Model author must prepare all combinations needed by the Model executor and

compile them to executable form. If a combination is missing, the Model executor has to order it from the Model author. The output from the Model author is a set of FMUs (executable models according to the FMI standard). These FMUs can then be used by both the Model executor and the System integrator as seen in Figure 1.

With the separation, the Model executor can work in any environment that supports the FMI standard (FMI, 2018), such as MS Excel (FMIE, 2018), MATLAB/Simulink (FMIT, 2018), or python, (pyFMI, 2018). Here, Excel sheets are used for configuration and data management, and MATLAB for Execution and post processing. This is consistent with the setup chosen in e.g. (Sundström, 2016a).

Using the FMI standard, the System integrator can manage models from any tool that supports the standard, see (FMI, 2018) for a list of available tools.

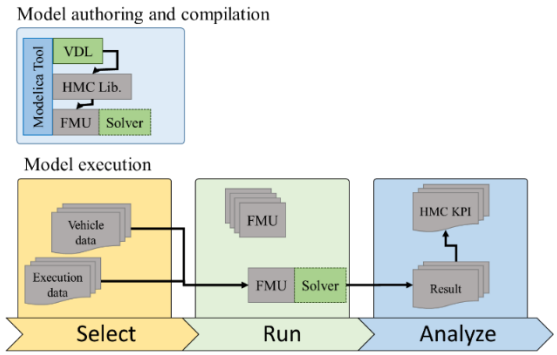

Figure 1 Execution of fixed configurations.

2.2 Execution of reconfigurable models

When there is a need for the Model executor to also reconfigure the model, add compile-on-demand functionality can be added as illustrated in Figure 2.

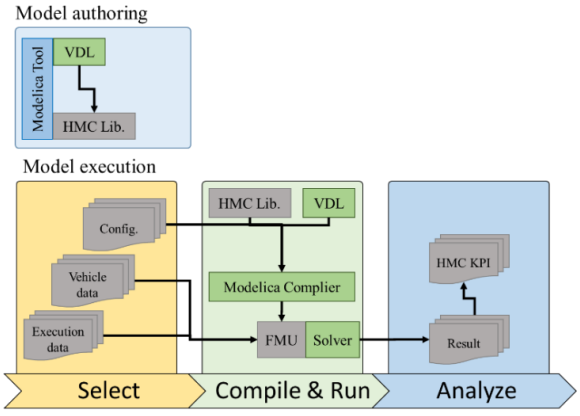

Figure 2 Execution of reconfigurable models.

Here, the output from the Model author is the Modelica library. In addition, the execution environment is equipped with a Modelica compiler. This allow for the execution environment to reconfigure Modelica model from the HMC (Hyundai Motor Group) Library according to the commands of the Model executor. Configuration options can be set up in the HMC Library so that the Model author can control what options are available for the Model executor as illustrated in Figure 3.

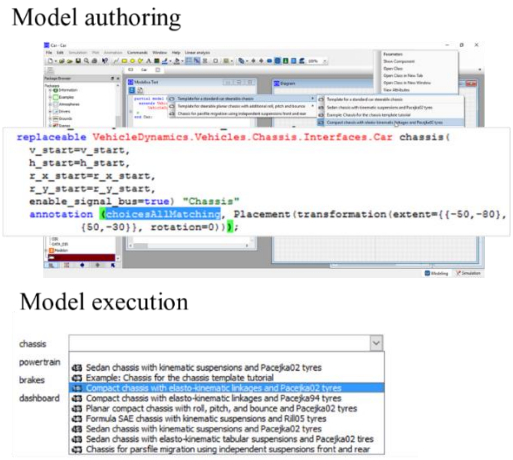

Figure 3 Reconfiguration in Model authoring environment (top) and in Model execution environment (bottom).

2.3 System integration

To allow to combine the model from HMC Library with 3rd party models, the suggestion is to also do this using FMI standard. Here an intermediate step is introduced to bundle the FMUs together prior to executing the simulation. The 3rd party models are maintained in a data base.

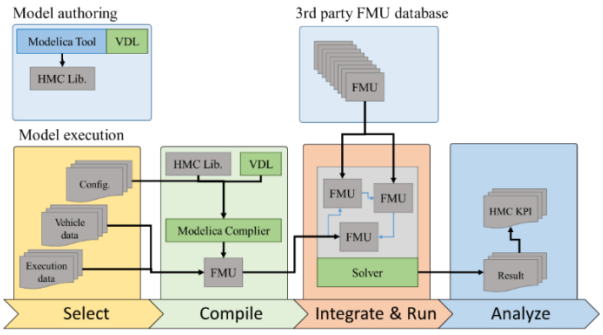

Figure 4 Toolchain with ability to include 3rd party FMUs.

3 Model creation

The HMC_Commercial_Vehicle library is seen in Figure 5. It is based on the Vehicle Dynamics Library (VDL, 2018) and makes use of the Interface – Template structure introduced in (Andreasson, 2006) now broadly used to build system libraries, see e.g. (Sielemann, 2017a), (Sielemann, 2017b).

DOI
10.3384/ecp18148109

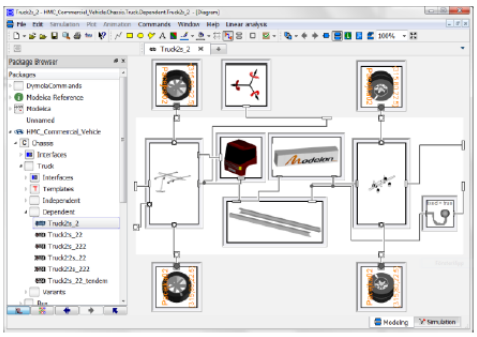

Figure 5 HMC_Commercial_Vehicle library.

In brief, the structure promotes separation of interfaces, topology and configuration as illustrated in Figure 6. Here, a partial model is defined containing all common interface attributes such as connectors and parameters (left) that in turn is inherited by all implementations. An implementation is then separated into the topology definition (middle) and the configuration (right). The topology is called a template and is defined using subsystem interfaces as placeholders, and the connections between these. Based on the template, variants can easily be defined by just stating the contents of each placeholder and thus eliminating the need to duplicate information for layout and connections.

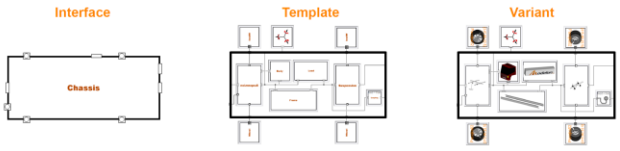

Figure 6 Interface - template structure according to (Andreasson, 2006).

3.1 Modelica library export of FMUs

To enable execution without recompilation, a structure that supports automatic generation of a large number of preconfigured variants was set up. In brief this is set up based on a strict package structure and naming convention with the corresponding representation on the file system as illustrated in Figure 7.

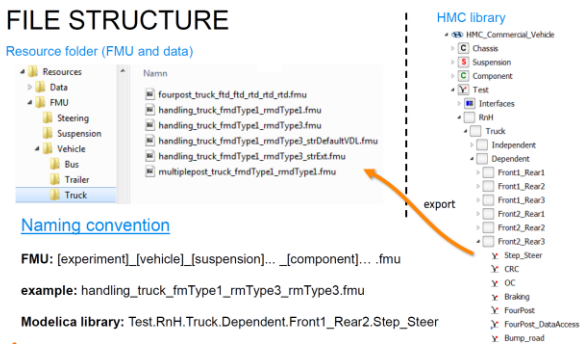

Figure 7 Library and FMU file structure

3.2 Model parameterization

Parameterization is separated from the model definition using DataAccess, see e.g. (Andreasson, 2016) for more elaborate examples. DataAccess allow parameter to be pulled and pushed by the model itself, and thus makes models natively able to interact with data repositories regardless of implementation and execution tool. In this case it means that the FMUs that are exported from the Modelica environment can read and write data to the HMC data-base also when executed in MATLAB and Excel.

The directory and filenames are string parameters that can be changed after export, before initializing the FMU.

3.3 Multi-FMU support

For some applications, for example when there are subsystems from a 3rd party tool. The framework needs to support multiple FMUs. From the Modelica-perspective, the HMC_Commercial_Vehicle library is prepared to support this by the introduction of causal adaptors. The causal adaptors converts the Modelica connectors to a directional signal flow that fits with other causal model implementations. To make this compatible with the interface-template structure, the methodology presented in (Andreasson, 2016) was reused with the addition that compound signal connectors is defined to handle the common cases. Figure 8 shows an example where the model is prepared for an external 3rd party steering FMU.

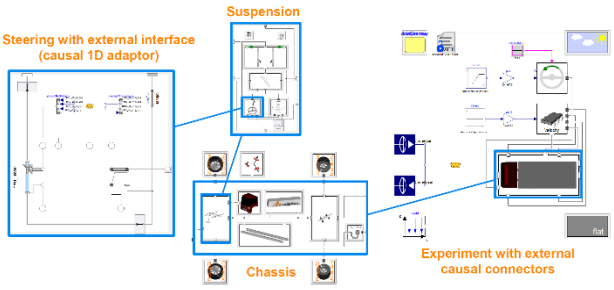

Figure 8 Model prepared for external steering FMU.

4 Model deployment

Figure 9 illustrates the model execution of single FMUs, without recompilation. The implementation is based on FMI functionality in MATLAB, (FMIT, 2018) and provides an API for high level operations as well as a graphical user interface. The workflow is based on three steps; configure, parameterize and simulate.

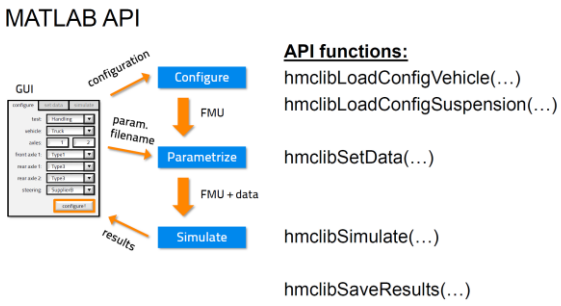

Figure 9 Model execution without recompilation.

4.1 MATLAB API

Figure 10 shows an example of how the API can be used to set up and execute simulations of single FMUs.

use case 1

handling

```
%% configure use case 1
%
% A handling experiment for a truck with one front and one rear axle.
% The front axle has a Type1 dependent MBD suspension.
% The rear axle has a Type3 dependent MBD suspension.
% The resource directory contains FMUs and data files.

config = hmclibLoadConfigVehicle(struct(...
    'experimentType','handling',...
    'vehicleType', 'truck',...
    'suspensionTypesFront',{{'mdType1'}},...
    'suspensionTypesRear',{{'mdType3'}},...
    'resourceDirectory','C:\SVN\P264-HMC_Framework\FromIVH\HMC_Commercial_Vehicle\Resources'))

%% parametrize
%
% Set the FMU data files for
%   a default step steer handling experiment.
%   a default truck with Type1 dependent MBD front suspension
%     and Type3 dependent MBD rear suspension.

experiment = hmclibSetData(config,...
    'handling_stepsteer_default.xml',... % experiment
    'fmd1_rmd3_default.xml')             % test object

%% simulate
results = hmclibSimulate(experiment)

%% save
hmclibSaveResults(results,'results1.mat')

%% load
clear
load('results1.mat')
results
```

Figure 10 Example of API usage.

When adding third party FMUs, an intermediate aggregation step is included, that calls the kernel of FMI Composer (FMIC, 2018). In brief this kernel combines multiple FMUs into one FMU such that all subsequent steps can remain intact. From the API user perspective, this appears as two optional inputs in the function call, Figure 11.

external steering

```
%% configure use case 1
%
% A handling experiment for a truck with one front and one rear axle.
% The front axle has a Type1 dependent MBD suspension.
% The rear axle has a Type3 dependent MBD suspension.
% The resource directory contains FMUs and data files.

config = hmclibLoadConfigVehicle(struct(...
    'experimentType','handling',...
    'vehicleType', 'truck',...
    'suspensionTypesFront',{{'mdType1'}},...
    'suspensionTypesRear',{{'mdType3'}},...
    'steeringType','DefaultVDL.fmu',...
    'resourceDirectory','C:\SVN\P264-HMC_Framework\FromIVH\HMC_Commercial_Vehicle\Resources',...
    'fmuStepsize',0.01))
```

Figure 11 Configuring experiment with 3rd party FMU.

4.2 GUI for pre-processing

GUI is built based on MATLAB-Simulink to give engineers convenience of execution. First, pre-processing is performed by inputting the model parameters like Figure 12.

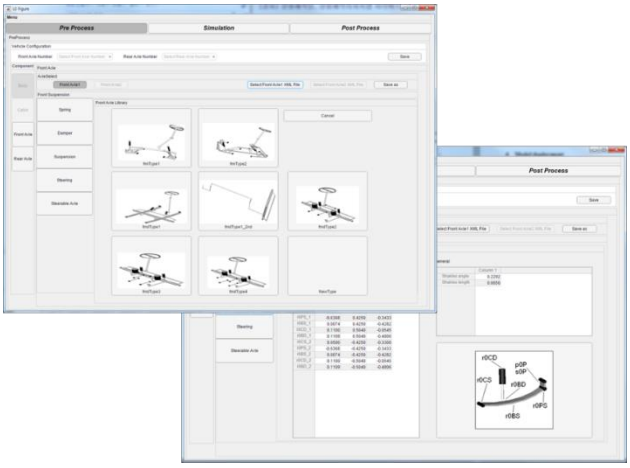

Figure 12 MATLAB GUI for FMUs parameterization

The GUI is employed not just to make connection between model parameters and FMUs, but also to give simulation condition. Figure 13 illustrates the GUI of simulation mode definition. By technical assistance of MATLAB FMI toolbox by Modelon, FMIT (2018), directly MATLAB start simulation of FMUs after this parameterization and simulation condition definition.

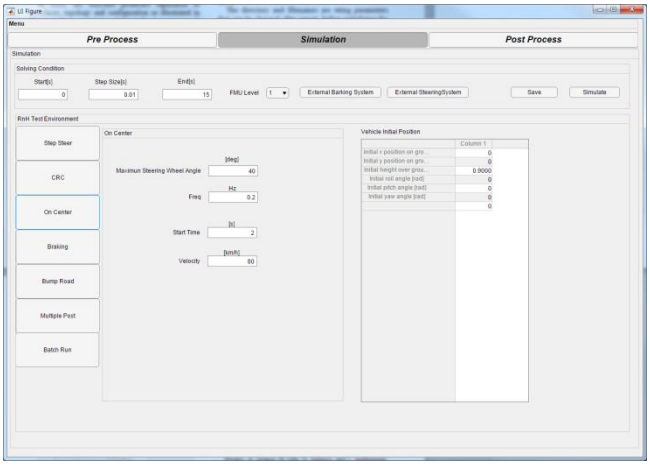

Figure 13 MATLAB GUI for definition of Ride and Handling simulation condition

4.3 Results

Finally, the simulation results are reproduced by plots and Key Performance Indexes that HMC has already defined. Once again, post-processor was developed on the MATLAB for the purpose of establishing consistent work flow from pre to post process. Figure 14 illustrates this automated post process results.

DOI
10.3384/ecp18148109

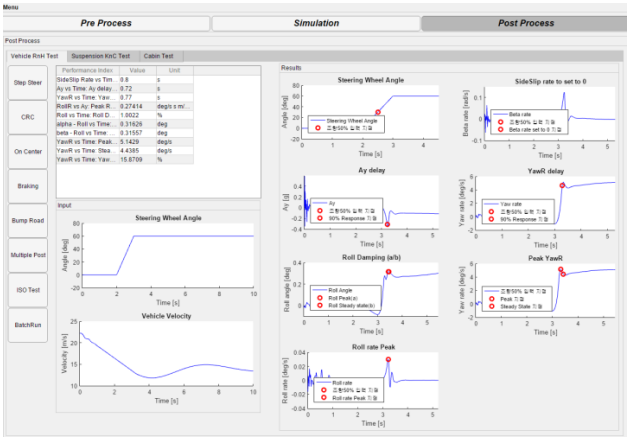

Figure 14 Automatically generated Simulation Results

5 Summary and future work

From this collaborating project, HMC is able to set up stable and consistent simulation tool for Ride and Handling performance using FMI technology and it will be utilized during our vehicle development process. Also HMC plan to set up similar workflow in many other research and development section like CFD, fuel consumption.

On the other hand, there is now the possibility to have a client-server approach to Modelica-based engineering (Elmqvist, 2018), which allows such functionality as FMU generation and execution of simulation and optimization to be remotely from the user and without the need for installation of software on the local machine. Such workflows as described here can therefore be carried out from a web browser using the same models as previously described, Figure 12.

References

J. Andreasson, M. Gäfvert: The Vehicle Dynamics Library – Overview and Applications, In proceedings of the 5th International Modelica Conference, Wien, September 2006. https://modelica.org/events/modelica2006/Proceedings/sessions/Session1b3.pdf

J. Andreasson, N. Machida, M. Tsushima, J. Griffin, P. Sundström: Deployment of high-fidelity vehicle models for accurate real-time simulation, In proceedings of 1st Japanese Modelica Conference, May 23-24, Tokyo, Japan, 2016. http://www.ep.liu.se/ecp/article.asp?issue=124&article=0

H. Elmqvist, M. Malmheden, J. Andreasson. A Web Architecture for Modeling and Simulation, In proceedings of 2nd Japanese Modelica Conference, May 17-18, Tokyo, Japan, 2018.

FMI, Functional Mockup Interface, 2018 http://www.fmi-standard.org

FMIC, FMI Composer, 2018. http://www.modelon.com/products/modelon-deployment-suite/fmi-composer

FMIE, FMI Add-in for Microsoft Excel, 2018. http://www.modelon.com/products/modelon-deployment-suite/fmi-add-in-for-excel

FMIT, FMI Toolbox for MATLAB/Simulink, 2018. http://www.modelon.com/products/modelon-deployment-suite/fmi-toolbox-for-MATLABsimulink

M. Henningsson, J. Åkesson, H. Tummescheit: An FMI-Based Tool for Robust Design of Dynamical Systems, In proceedings of 10th International Modelica Conference, March, Lund, Sweden, 2014. https://www.modelica.org/events/modelica2014/proceedings/html/submissions/ECP1409635_HenningssonAkessonTummescheit.pdf

pyFMI, FMI support for python, 2018. https://pypi.python.org/pypi/PyFMI

M. Sielemann, A. Pitchaikani, N. Selvan, M. Sammak: The Jet Propulsion Library: Modeling and simulation of aircraft engines, 2017. In proceedings of 12th International Modelica Conference, May, Prague, 2017.

M. Sielemann, J. Andreasson: Towards Model-Based Design of Aircraft Systems, 2017. NAFEMS World Congress 2017.

P. Sundström and J. Andreasson: Model-based design and control of long heavy vehicle combinations, In proceedings of 2016 IEEE Intelligent Vehicles Symposium, June 19-22, Gothenburg, Sweden, 2016.

P. Sundström, M. Henningsson, X. Carrera Akutain, Y. Hirano, A. Ocariz, H. Iida, N. Aikawa, and J. Andreasson: Virtual Vehicle Kinematics and Compliance Test Rig, In proceedings of 1st Japanese Modelica Conference, May 23-24, Tokyo, Japan, 2016. http://www.ep.liu.se/ecp/article.asp?issue=124&article=004

VDL, Vehicle Dynamics Library, 2018 http://www.modelon.com/products/modelon-library-suite/vehicle-dynamics-library

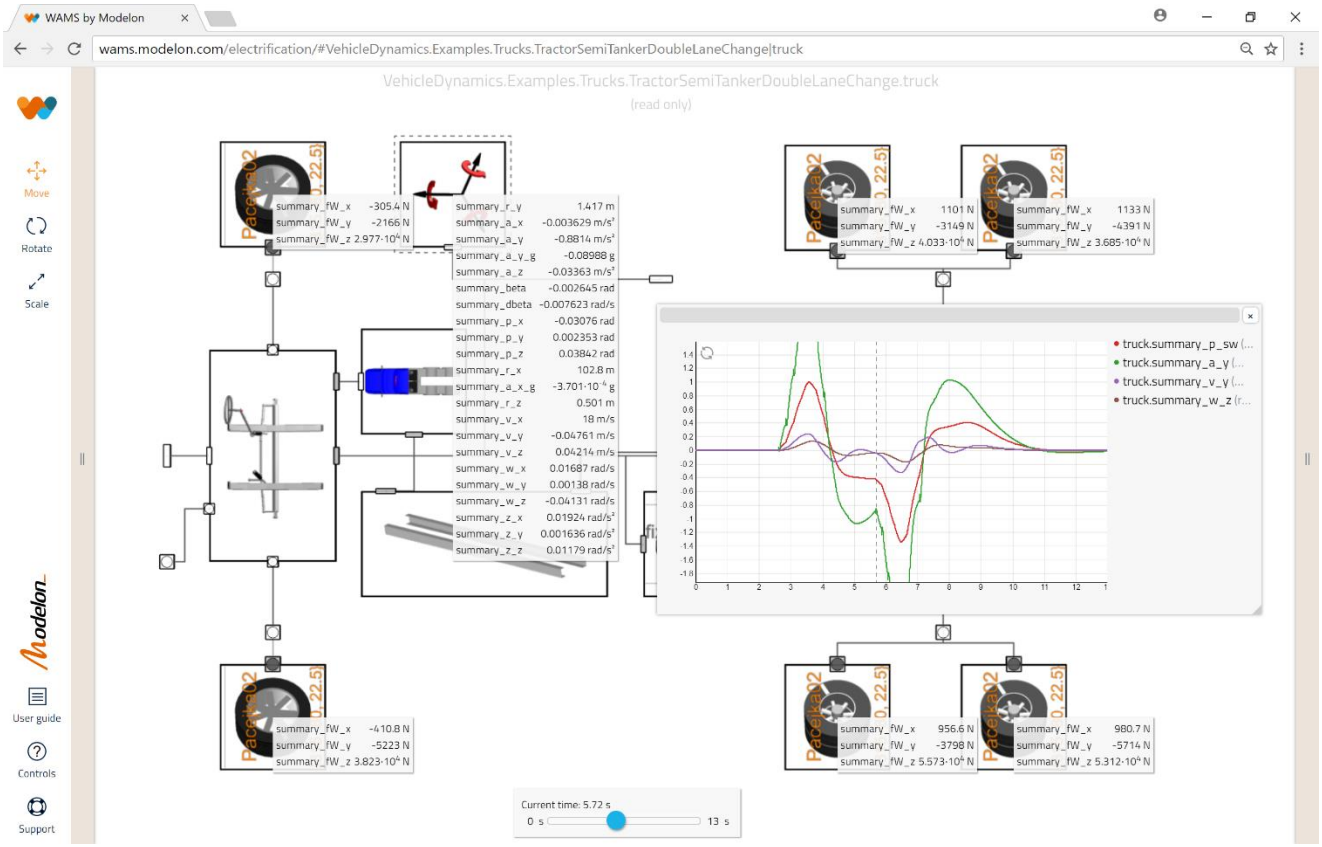

Figure 12 Client-server based truck simulation using web architecture for modeling and simulation. (Note that web link is not available without special access.)

DOI
10.3384/ecp18148109