Deployment of high-fidelity vehicle models for accurate real-time simulation

Johan Andreasson[1] Naoya Machida[2] Masashi Tsushima[2] John Griffin[3] Peter Sundström[4]
[1] Modelon KK, Japan
[2] Nissan Motor Co., Japan
[3]Modelon Inc., USA
[4]Modelon AB, Sweden

johan.andreasson@modelon.com nao-machida@mail.nissan.co.jp
masashi-tsushima@mail.nissan.co.jp john.griffin@modelon.com peter.sundstrom@modelon.com

Abstract

In the effort to shorten development cycles and with the reduced ability to test in real life, driver-in-the-loop simulators are increasingly used by automotive OEMs and in Motorsports to enable engineers and drivers to experience a new vehicle design in a realistic environment before it is built. With the right level of accuracy, the same model can be applied in other real-time vehicle dynamics applications to allow for testing and verification in the development of new vehicle functions.

This paper gives and overview of the requirements for automotive real-time application and the solution chosen. Emphasis is given on the model definition and real-time configuration as well as parameterization from existing data sources and integration of third party subsystem models.

Keywords: vehicle simulators, vehicle dynamics, real-time, hardware-in-the-loop, driver-in-the-loop

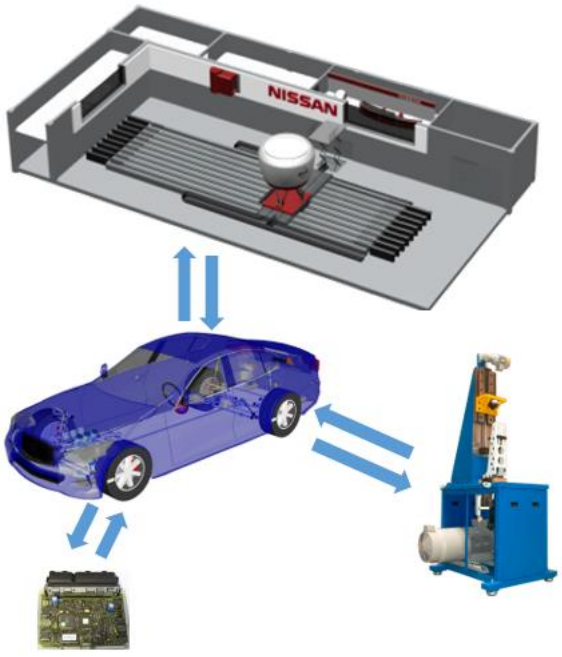

Figure 1. Applications of real-time capable models: Driver-in-the-loop simulator, Yasuno (2014) (top), ECUs (bottom left) and component hardware testing (bottom right).

1 Introduction

Virtual representations that can predict a vehicle's real life behavior have become more and more important in the development process. There are some well-known reasons for this, such as overcoming the cost, time, safety and repeatability issues with physical prototypes (Rauh, 2003).

Recently, deployment of real-time vehicle dynamics simulation for hardware-in-the-loop testing and driving simulators has gained significant attention (Yasuno, 2014). This trend is largely driven by the demand for shorter development cycles that also should result in better products.

Figure 1 illustrates three typical applications that drive the use of real-time models. The ability to perform early assessment of drivers' perception of the vehicle being designed is one of the key driving factors (top).

Another common use is for the integration of Electronic Control Units (ECUs) from suppliers where the actual implementation is often hidden or partly hidden as a so called black box (bottom left). A third case is to include parts of the vehicle as hardware to provide realistic boundary conditions for component testing (bottom right).

Ultimately, all these applications can be combined, with the virtual vehicle representation as the glue to be able to assess the complete driver-vehicle system at any point during the development process. Here, we try to give a brief overview of the requirements a modern engineering tool chain should meet and pointers to where in the paper the topics are addressed.

1.1 Accuracy and real-time capability

The late testing, tuning work and potential design iterations associated with physical prototypes are known to not just be expensive, but also to extend the development time with associated risks to delay market introduction. To be able to reduce physical testing further, the required simulation accuracy increases, which in turn typically increases the computational effort which is in conflict with real-time requirements.

For vehicle dynamics, multi-body representations of the vehicle mechanics are a standard approach in the automotive industry as they are able to capture the relevant phenomena for vehicle dynamics, while providing a straight-forward parameterization.

For motorsports applications, the vehicle chassis is traditionally close to an ideal kinematic behavior. As such, the number of degrees-of-freedom (DOF) can be kept to around 30-50. Such multi-body representations have been used successfully in real-time applications for a long time. Applications include driver training, driver perception evaluation of new designs and complete driver-vehicle systems integration, see e.g. (Toso, 2014).

For passenger vehicles the vehicle design is generally different, especially due to the many elastic elements that are used to enhance comfort and tune vehicle attributes. This leads to more complex representations with typically 150-300 DOF that traditionally were not possible to execute in real-time. Therefore, the current industry practice is to use a vehicle model with significantly reduced complexity that results in a lower computational load. Unfortunately, this means lower accuracy, a limited valid frequency range and additional pre-processing of the model to generate the required parameterization, which slows down iteration time.

With the introduction of new technology for parallelizing vehicle models (Andreasson et al., 2014) and (Elmqvist et al., 2014), the high model fidelity used offline (see Section 2) is made executable in real-time applications, as explained further in Section 5.

1.2 Inter-operability

In the context of shorter lead-times, no tool can be an island, they must connect to form an efficient tool chain. With the amount of legacy tools and methods used by OEMs today, this puts some additional requirements on any new model:

1. The model must be able to share data, meaning reading and writing existing formats.
2. It must also be possible to plug the model into the tool chain in such a way that it can use existing tools for pre- and post-processing.
3. Additionally, subsystem models from different sources need to be included in the complete vehicle model to allow for incremental improvement, multi-fidelity and use of heritage/legacy code.

These topics are further described in Sections 3 and 4.

2 Model overview

The vehicle in this example is a production vehicle featuring a front double wishbone suspension, a rear multi-link suspension, front engine, rear wheel drive, and an automatic transmission, Figure 2. The models are based on the (Vehicle Dynamics Library, 2015) and largely implements the template and interface structure provided, Figure 3.

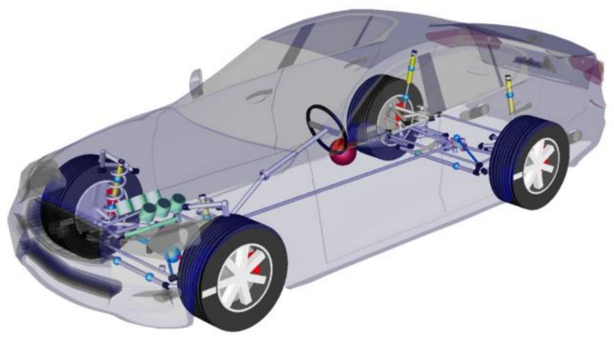

Figure 2 Vehicle model used in the presented work.

For each component or sub-system, the vehicle model can be reconfigured by replacing subsystems with plug-in compatible variants. This allow for changing both configuration (e.g. from automatic to manual transmission) and fidelity (e.g. from multi-body to tabular suspension).

For each subsystem or component, there is a well-defined interface that specifies the boundaries and provides an established framework to connect pieces from existing various simulation tools as described in the next section.

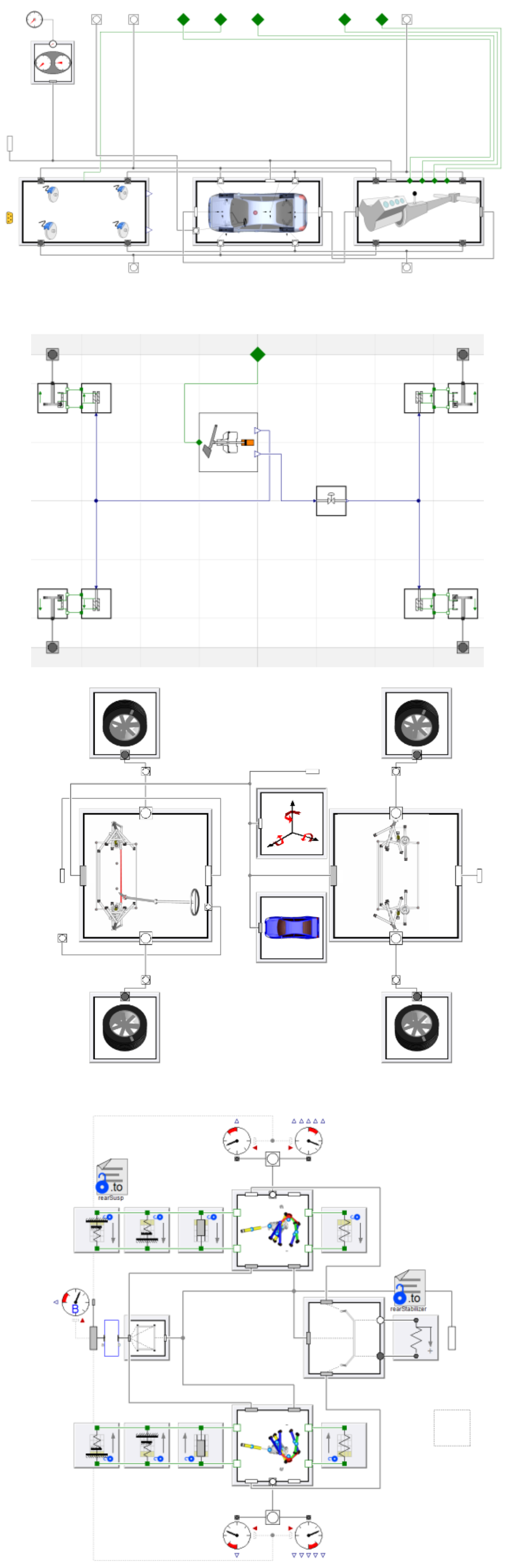

Figure 3. Architecture of the complete vehicle model. Vehicle (top), brake system (second), chassis (third) and rear suspension (bottom).

3 Deployment

To ensure convenient interoperability with subsystem models from different sources, these models connects either from within the Modelica framework or from the tools originally used. This section illustrates the two cases as well as the final integration of the complete real-time model.

3.1 Multi-physics brake model

To include higher detail in the brake simulation and get more realistic brake pedal feel, the pneumatic vacuum booster and the hydraulic master cylinder are modelled as physical components (Hydraulics Library, 2015) and (Pneumatics Library, 2015). Figure 4 shows the layout of the brake system model.

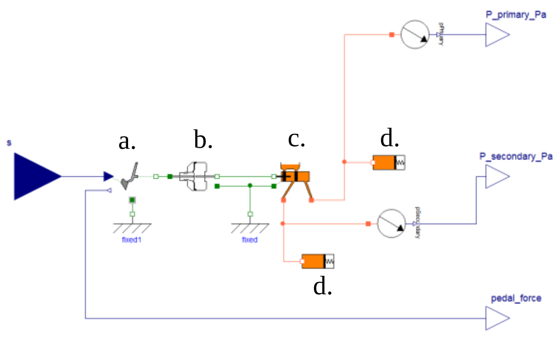

Figure 4. Diagram layer of the brake system model. Components are; pedal actuator (a), vacuum booster (b), master cylinder (c) and brake lines and calipers (d).

The core of the booster model is a double acting pneumatic cylinder corresponding to the booster diaphragm. Based on the pedal actuating the poppet valves, vacuum and atmosphere pressure is applied to the different sides of the diaphragm to boost the pedal force acting on the master cylinder piston.

The force characteristics of the vacuum booster are affected by elasticies and gaps in the mechanisms that open and close the valves to the diaphragm. High stiffness in combination with low mass in these components can result in fast dynamics. This is typically not relevant for the application, and may also be too fast for the desired integrator time step. To handle this, the spring-mass combinations are replaced with elements where the bandwidth can be explicitly defined.

The master cylinder is a two circuit variant with a mechanical gap connecting the two cylinders. As long as the first circuit is pressurized, the gap is open and the hydraulic pressure also activates the second cylinder. If the pressure in the first circuit is lost, the gap will close so the pedal force is still transferred to the second cylinder though with changed pedal travel.

To show the more detailed function of the booster in combination with the hydraulic system can give a more refined accurate prediction of the driver's perception, Figure 5 illustrates how the brake system responds to a press-and-release cycle of the pedal force. The upper plot shows the brake line pressure which also will affect the actual and experienced retardation generated. The lower plot shows the pedal stroke which provides the driver with feedback through the foot.

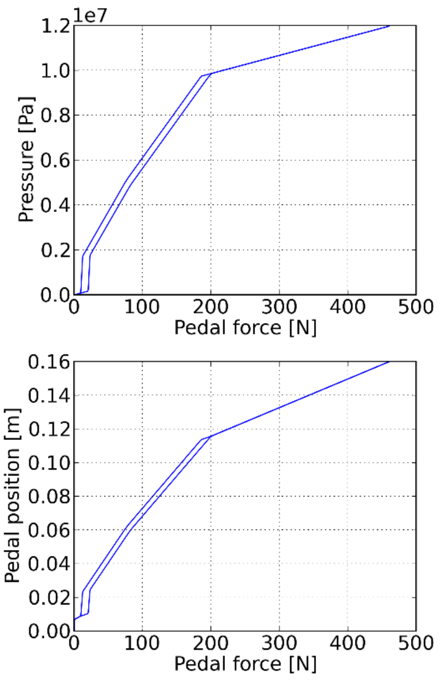

Figure 5. Top plot: Brake line pressure (vertical axis) as a function of brake pedal force (horizontal axis). This shows the knee points where the booster engages and disengages. Bottom plot: Corresponding pedal travel (vertical axis) for the same pedal force excitation. For confidentiality reasons, the numerical values have been removed.

3.2 External steering model

External subsystem models not implemented in Modelica can either be imported into the Modelica framework, or the vehicle model is exported without the corresponding subsystems for integration on an external platform.

External models are brought into the Modelica model using external functions or objects, or to include so called Functional Mock-up Units (FMUs) adhering to the Functional Mock-up Interface (FMI, 2015). In either case, these are wrapped into the subsystem interfaces to ensure plug-in compatibility. A variety of external models, including (OpenCRG, 2015), (DelftTyre, 2015) and (FTire, 2015) are predefined and ready to use.

Correspondingly, when exporting the vehicle model, the systems that should be external are replaced with models that provide no contents. Figure 6 illustrates a Modelica steering system model (top), an external steering model included in the vehicle model (middle) and an empty steering model with external inputs and outputs (bottom).

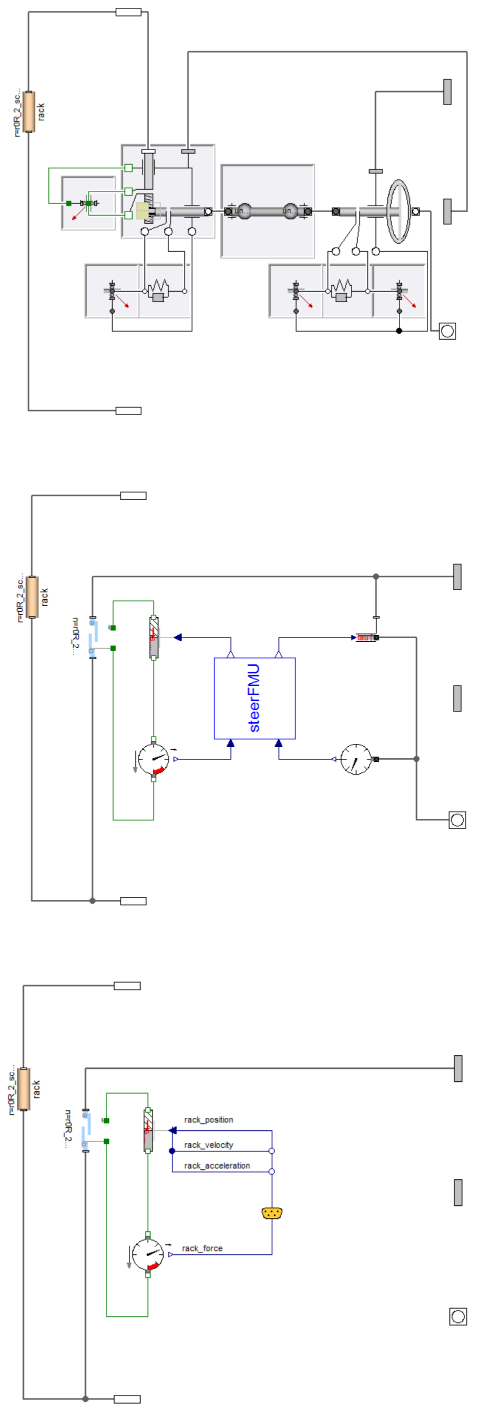

Figure 6. Modelica steering model (top), steering model imported as and FMU (middle), empty steering model requiring rack motion and providing rack force.

3.3 Complete real-time model integration

After configuring the vehicle model to allow for external subsystems, it can be exported in different formats (such as FMI, C, etc.) for system integration elsewhere. For deployment, the system integration is currently done in SIMulationWorkbench (Concurrent, 2015).

The system architecture is shown in Figure 7. The shaded area indicates what is handled by the real-time host. It contains the models (2) of chassis (2a), brake system (2b), steering (2c), power train (2d), controllers (2e) and I/O devices (3) for interaction with the motion platform (3a) and driver interface (3b).

The Communications between the models (2) and the I/O devices (3) is handled through shared memory interconnection (1) called Real Time Data Base (RTDB), Concurrent (2015). RTDB allows access to any variables stored in RTDB for any of the included models and is achieved to wrap the inputs and outputs of each of the model with dedicated read/write functionality.

The real-time host then communicates with the operators through a set of clients that provides a Graphical User Interface (GUI) for configuration of I/O (5a), data recording (5b), playback (5c) and configuration of RTDB (5d).

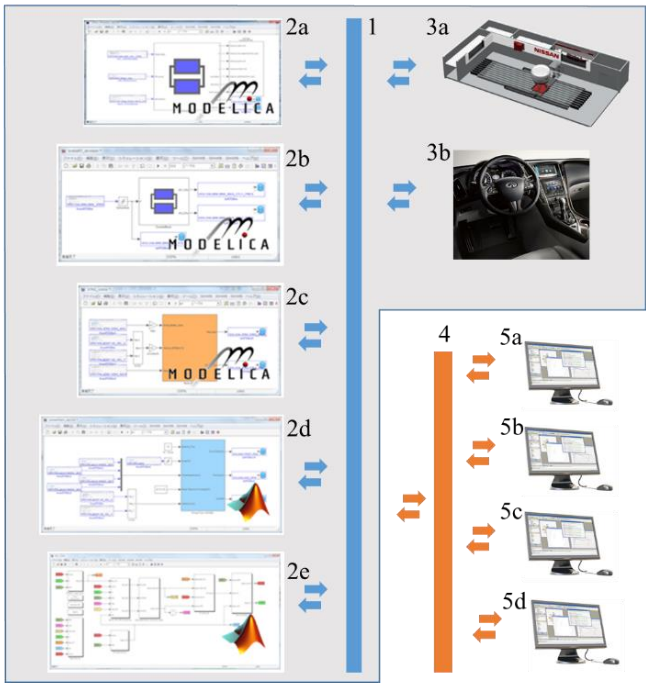

Figure7. Software architecture of the real-time simulation host.

To allow for communication through the RTDB, each model must contain its own numerical integrator and can only expect to share information at each communication point. So, effectively, the real-time host works like a co-simulation platform. A benefit with this approach is the ability to let each model integrate at multiples of the communication interval, so called multi-rate integration.

Multi-rate integration is suitable as it allows for models from tools that only supports explicit integration schemes to be executed at small enough time step to ensure numerical stability. Correspondingly, the method allows for plant models to be integrated at shorter time steps than the controllers.

With co-simulation, there is also inherent support to distribute the integration of each model to its own set of cores. Additionally, the chassis model is parallelized on multiple cores as described further in Section 5.

4 Parameterization

For the real-time model to be an efficient tool, it is crucial that it can access the latest state of development. Therefore, the model is designed to read the same data set that is used in existing offline tools, in this case the TeimOrbit format.

4.1 Accessing data

The data management is accomplished using a general-purpose data management method to read and write external data called DataAccess. This method can handle a variety of different file formats such as .xml, .json and .mat. Since DataAccess is compiled into the simulation code, it is well suited for model export as it allows users to conveniently change model parameters in a consistent way regardless of how and to what format the model is exported, Figure 8.

For this work, the handling of the TeimOrbit format was added to allow data sharing with the offline tools. Initially, the real-time model was parameterized by manually accessing a data value required for a particular attribute, such as the mass of a part. This resulted in significant effort and duplication of code to read the necessary data. As the project progressed, it became evident that duplicate coding could be eliminated by creating data-aware components, Figure 9.

A data-aware component is responsible for reading all the data that it requires from the data file. For example, a data-aware part is responsible for reading all of the mass and inertia data associated with it; and a data-aware bushing is responsible for reading all the force and torque characteristics that describe it. Data-aware components are also easier to validate because correlation occurs at the component-level and the only configuration work is to point the component to the right data source.

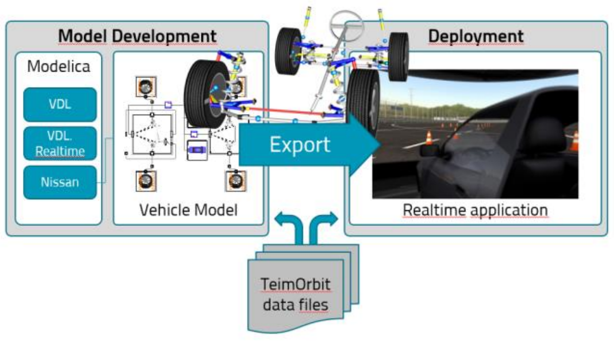

Figure 8. The data management (DataAccess) is compiled into the executable model. This ensures that the parameters are read from the same source regardless of the deployment scenario.

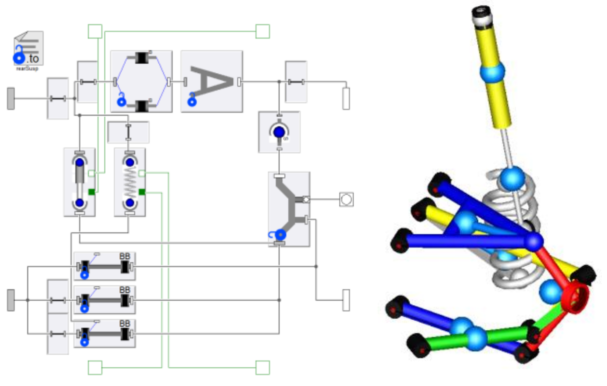

Figure 9. Linkage topology view (left) configured with data aware components that read TeimOrbit data and resulting 3D view (right).

4.2 Validation against offline tool

To ensure consistent behavior between offline tool used in development and the new real-time model, a validation procedure was carried out. This procedure included comparison on different levels, from components to chassis on road. For the suspension level, the standard offline procedure was replicated in Modelica, and configured to read the test specifications produced by the offline tools. Figure 10 shows the resulting test model that contains three main components; the test rig, the suspension, and the signal source.

The test rig provides a constraint between vehicle body and ground. The wheel centers are excited through moving the wheel pads which induces forces through the tires. Additionally, forces and torques can be applied either at the wheel center, or at the tire contact patch. For the front suspension, either steering wheel position or steering wheel torque is given.

The suspensions are the same suspension that is used in the complete vehicle (Figure 3). The source block is configured using DataAccess to read the configuration information used by the off-line tool eliminate manual reconfiguration of test scenarios. Figure 10 shows some example correlation plots from the front suspension for parallel wheel travel.

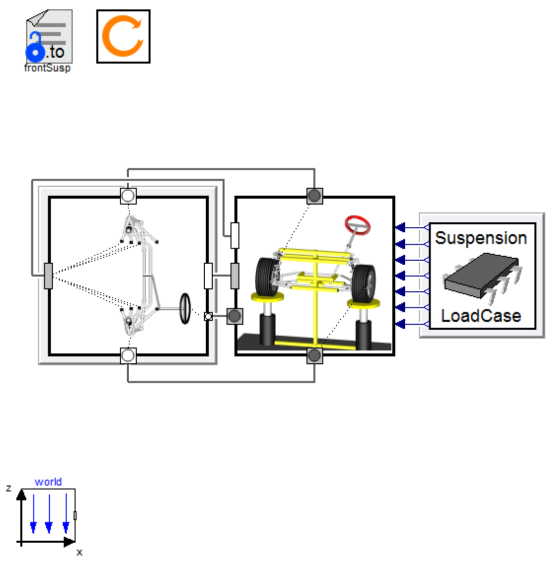

Figure 10. Suspension test model implementation showing the tested suspension, the boundary conditions, and the source block that reads the test specification.

5 Realtime configuration

The Vehicle Dynamics Library have been used to model real-time capable multi-body vehicle models for more than a decade, (Elmqvist et al., 2004). These models are heavily adopted in the Motorsports industry for various applications, see e.g. (Toso, 2014).

With recent development (Andreasson et al., 2014), it is now possible to execute high fidelity vehicle models with more than 150 DOF (300 states). This allows the models used for vehicle development at CAE departments to be executed directly in real-time applications. Key methods to achieve the performance is the inlining of the real-time solver and parallelization of the executable code.

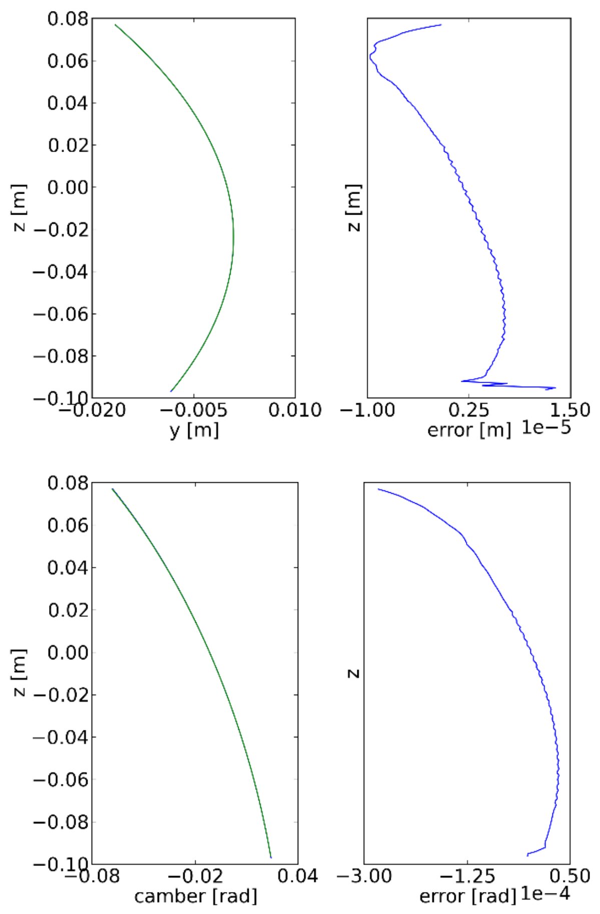

Figure 11. Comparison of results generated from offline tool (blue) and real-time model (green) for a parallel wheel travel. Plots show lateral (top) and camber (bottom). Each plot shows the curves overlaid and error, respectively.

5.1 Inlining

So called inlining (Elmqvist et al., 1995), has proven to be a successful way of achieving real-time performance of stiff systems, (Elmqvist et al., 2004). Inlining essentially means that the discretization formulas of the integration methods are substituted into the differential equations of the model, and then structural analysis and computer algebra are applied on the augmented system of equations. The method can be combined with both explicit and implicit discretization schemes. The explicit scheme is straight-forward but requires small enough steps to ensure stability. The implicit scheme has better stability properties but typically results in non-linear system of equations that need to be solved iteratively.

5.2 Parallelization

Solving large systems of non-linear equations is O(n^3), meaning cost to solve the problem grows with the third power of the number of equations. For the model in this work, the size of the manipulated inlined implicit integration system is 178 (n_1=178), which would be difficult to solve robustly in any real-time application without further manipulation.

As described in (Elmqvist et al., 2014), parallelized code can now be generated from Modelica models according to the (OpenMP, 2015) standard. The real-time model takes advantage of this functionality to distribute the workload of solving the systems of equations across multiple cores according to the following principle:

After the implicit integration scheme has been inlined with the model, the resulting system of equations is then divided into several smaller systems corresponding to the dynamics of the rear and front left and right suspension linkages, the powertrain, the steering and the wheels. The resulting impact on the structural side is that one large system of equations is now reduced to several smaller systems after manipulation, here n_2={40, 40, 32, 32, 12, 11, 1, 1, 1, 1, 1, 1, 1, 1}.

This split gives two advantages, first several smaller systems solve significantly faster than one big system due to the cubic growth described above. In this particular case the reduction corresponds roughly to n_1^3/Σn_2^3, which is about 30 times.

The second advantage is that the parallelization of the code allows the execution to be distributed on multiple cores. This also means that as long as there are cores available to distribute the calculations onto, any added model complexity will have a limited effect on the overall turn-around time as long as it does not add to any of the largest systems of equations.

5.3 Simulation accuracy and performance

The real-time configuration has been validated both with respect to accuracy and performance. To ensure robustness to high amplitude and high frequency inputs, the test suite contains a broad range of excitations such as jump and police turn in addition to the more traditional simulation set-ups. Figure 12 shows the vertical acceleration of the body while the car accelerates over an uneven ground surface.

The performance of the real-time model is defined by the time it takes to solve for each time step, so called turn-around time. Indications of the performance can be done directly on a desktop or laptop using timers, and on a Windows laptop (i7-3520M @ 2.90 GHz) the performance is roughly real-time. In Figure 13, the timing plot on the hardware-in-the-loop platform (Concurrent Xeon E5-2687w v2) is shown.

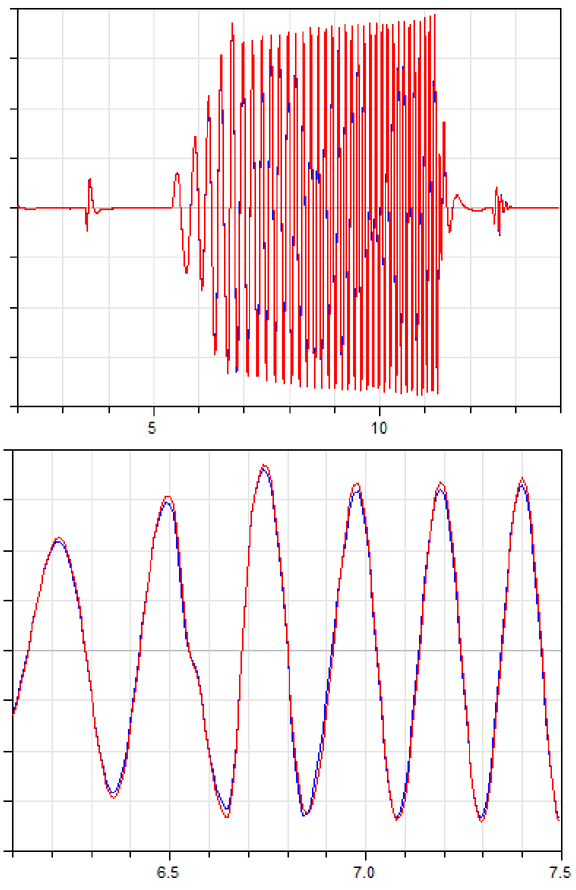

Figure 12. Time trace of vertical acceleration while accelerating over an uneven road. Reference trajectory generated with Dassl solver using relative tolerance 1e-6 (blue), and real-time solver (red). For confidentiality reasons, the numerical values of vertical axis have been removed.

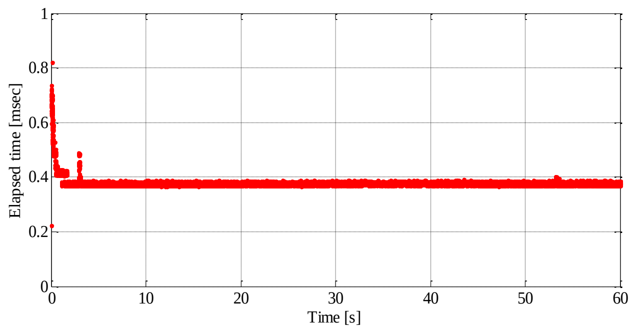

Figure 13. Execution time on the hardware platform. Each time step is 1ms.

6 Conclusions

This paper presents a real-time capable, high-fidelity model of a production vehicle. It is shown how this model can add real-time capability to the existing toolchain without having to replace or re-implement existing functionality. This is achieved by combining an open architecture with the ability to read and write legacy data formats. It is also shown how to enable real-time simulation of high-fidelity vehicle models using inlining and parallelization.

All-in-all, the model presented in this paper can respond accurately to inputs and realistically predict the vehicle behavior as of the latest state of the development process. The deployment in real-time environments such as driver-in-the-loop and hardware-in-the-loop simulators enables both subjective and objective evaluation. This in turn allow for early assessment of the human-vehicle interaction and the integration of vehicle safety systems.

7 References

Andreasson, J. et al. (2014). Realtime simulation of detailed vehicle models using multiple cores, in proceedings of *International symposium of Advanced Vehicle Control, AVEC*, Tokyo.

Concurrent (2015). www.ccur.com

DelftTyre (2015). https://www.tassinternational.com/delft-tyre

Elmqvist, H. et. al. (1995). Inline Integration: A new mixed symbolic/numeric approach for solving differential-algebraic equation systems, Proceedings of *European Simulation Multiconference*, June, Prague, pages XXIII-XXXIV.

Elmqvist, H. et al. (2004). Realtime Simulation of Detailed Vehicle and Powertrain Dynamics. In Proceedings of the *SAE World Congress 2004*, Paper no 2004-01-0768, Detroit, Michigan.

Elmqvist, H. et. al. (2014). Parallel Model Execution on Many Cores, Proceedings of *10th International Modelica Conference*, Lund, Sweden.

FMI (2015). www.fmi-standard.org

FTire (2015). www.cosin.eu

Hydraulics Library (2015). www.modelon.com/products/modelica-libraries/hydraulics-library/

Modelica (2015). www.modelica.org

OpenCRG (2015). www.opencrg.org

OpenMP (2015). openmp.org

Pneumatics Library (2015). www.modelon.com/products/modelica-libraries/pneumatics-library/

Rauh, J. (2003). Virtual development of ride and handling characteristics for advanced passenger cars. *Vehicle System Dynamics*, 40(1-3), 135-155.

Toso, A. and Moroni, A. (2014). Professional Driving Simulator to Design First-Time-Right Race Cars. *SAE Technical Paper 2014-01-0099*.

Vehicle Dynamics Library (2015). www.modelon.com/products/modelica-libraries/vehicle-dynamics-library/

Yasuno, Y. et al. (2014). Nissan's New High Performance Driving Simulator For Vehicle Dynamics Performance & Man-Machine Interface Studies. In proceedings of *Driving Simulation Conference 2014* Paris, France, September 4-5, 2014