presented at: 8th International Modelica Conference, 20-22.03.2011, Dresden, Germany

Simulation-based development of automotive control software with Modelica

Emmanuel Chrisofakis[1], Andreas Junghanns[2], Christian Kehrer[3], Anton Rink[1]
[1]Daimler AG, 70546 Stuttgart
[2]QTronic GmbH, Alt-Moabit 91a, 10559 Berlin
[3]ITI GmbH, Webergasse 1, 01067 Dresden
{emmanuel.chrisofakis, anton.rink}@daimler.com, andreas.junghanns@qtronic.de, kehrer@iti.de

Abstract

We present and discuss the Modelica-based development environment currently used by Daimler to develop powertrain control software for passenger cars. Besides well calibrated vehicle models, the environment supports automotive standards such as A2L, MDF, CAN, and XCP to integrate control software and simulated vehicles on Windows PCs.
Keywords: automotive software development, software in the loop

1 Introduction

More and more automotive functions are implemented using software. Hence, there is an increasing demand to support the corresponding development process using virtual, i. e. simulation-based development environments.

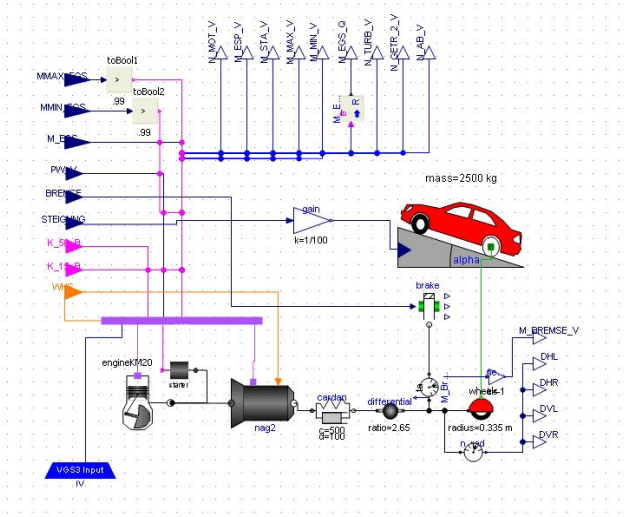

Figure 1: Vehicle model as used for SiL

Virtually coupling control strategies with plant models is standard technology today, mostly using common-place tools such as Matlab/Simulink for pre-development of control algorithms. This paper presents technology targeted toward the late stages in the development process, like tuning, validating and debugging the entire controller software in closed loop with simulated plant models. Virtualizing these later engineering tasks requires plant models with increasingly higher quality (physical effects modeled and quality of calibration) and near-production controller software (percentage of the controller software included, parameterization using production parameter sets and adaptation of the software to the plant) to be coupled.

A tool-chain supporting such coupling should

- be easy to set up and use by automotive developers who are usually not computer scientists
- support many of the engineering tasks usually performed with physical prototypes to allow for front-loading
- support short turn-around times, i. e. minimize the time between editing of control software and validation of the resulting behavior on system level to help find problems early
- provide built-in support for standards and de-facto standards used in automotive software development to allow cost-effective use of existing information sources
- support distributed development and exchange of work products between OEMs, suppliers, and engineering service providers. This requires e. g. measures to protect intellectual property.
- support reconfiguration of the development tool chain, since automotive development tools are frequently updated or replaced, e. g. due to emerging standards, new bus protocols or tool policy considerations.

In this paper, we present the simulation-based development environment used by Daimler to develop the powertrain control software for Mercedes passenger cars. The tool chain presented here addresses the

above demands. It is based on vehicle models implemented using Modelica and processed using SimulationX as a tool for the design and analysis of complex systems, the FMU standard for model exchange, MATLAB/Simulink and TargetLink as a tool for model based development of automotive controllers and Silver as a tool for virtual integration of control software, application data and the simulated vehicle.

The paper is structured as follows: In the next section, we describe why and how Modelica is used here to create vehicle models. Section 3 describes how such a vehicle model is then coupled with control software and what else is needed to get automotive control software running in closed loop on a PC. Section 4 describes how such a SiL setup is used to support automotive software development, and section 5 describes costs and benefits of setting up a SiL.

2 Vehicle models

Daimler started around 2004 to use Modelica for building vehicle models used for test and development of powertrain control software via software in the loop (SiL). For example, the members of the 7G-Tronic transmission family have been developed this way [1]. Ongoing projects developed within this Modelica-based framework include dual-clutch transmissions by Mercedes [2] and AMG [3], and hybrid drivetrains. Basic requirement of a plant model in a SiL-environment for automatic gearboxes is the accurate calculation of the gear shifting. In order to achieve this goal, detailed model representation of gearbox kinematics, clutch mechanics and hydraulic control is essential. Therefore special Modelica libraries have been developed over the years to support transmission development.
For the development of customer specific libraries SimulationX offers a wealth of options such as the dedicated TypeDesigner that simplifies graphical and textual modeling compared to traditional forms.
Based on these libraries, a well calibrated vehicle model for a new transmission project can be setup within just a few weeks. This short development is partly credited to good properties of the Modelica language, which provides outstanding support for the reuse of component models, mainly by providing powerful means to parametrize models and built-in support for acausal modeling. Latter feature offers the model developer great possibilities to calibrate and validate his model by using measurements either from car or from test rig since no model modifications are necessary if the measured signal is a flow or potential quantity (e. g. torque as opposed to speed).

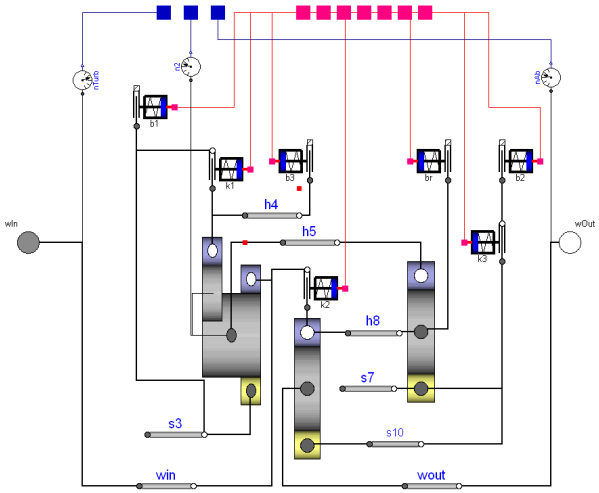

Figure 2: Gearset of a 7G-Tronic Transmission

Different capabilities for implementing measured data in SimulationX and validating the Modelica models against these data without the necessity of using another tool in combination with further options like the VariantsWizard help to increase the effiency of model calibration. With special regard to the needs of powertrain modeling ITI provides different analyzing methods, e. g. the linear system analysis or the steady state simulation.

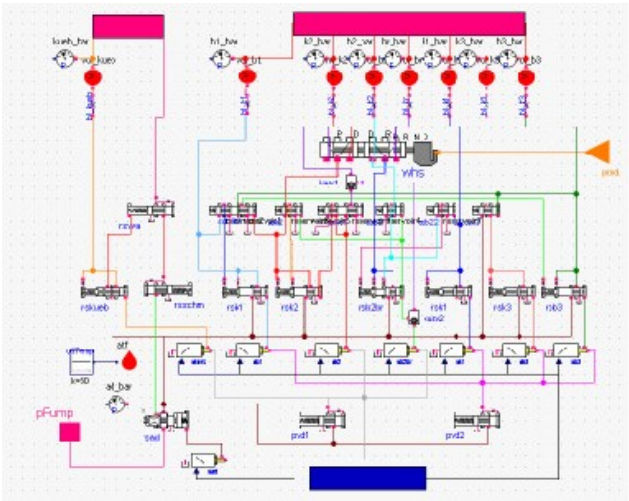

Figure 3: Transmission hydraulics

Figures 1, 2 and 3 show typical Modelica models used in series development projects.

Daimler uses Dymola and also SimulationX [4] to edit and process Modelica models. Since Modelica version 3.1 there is full compatibility of the plant models both in Dymola 7.4 as well as in Simula-

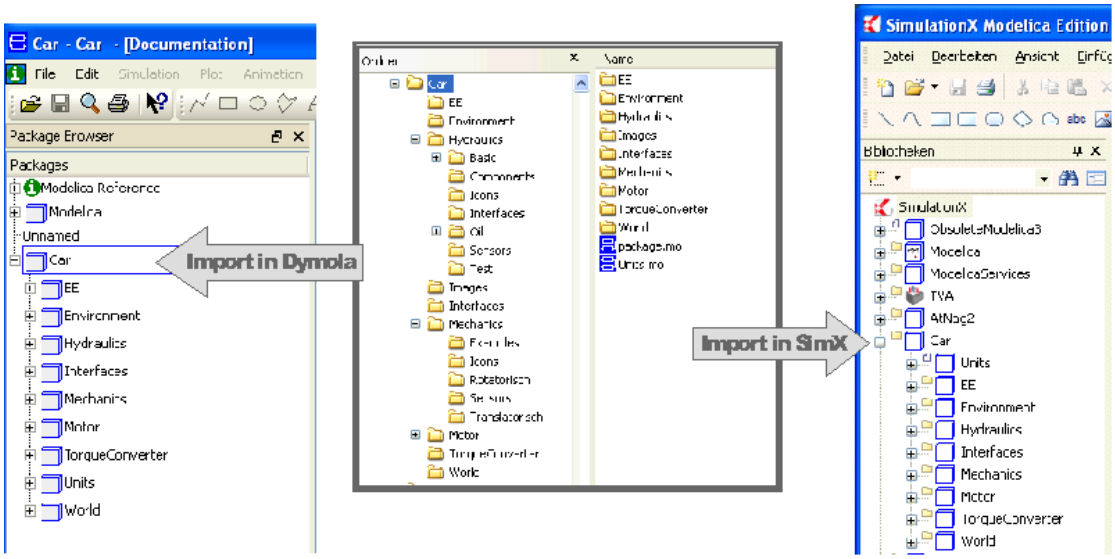

Figure 4: Modelica library Car in SimulationX and Dymola

tionX 3.4. Models and libraries are stored on hard disk as .mo files. Both tools are able to read these files with no specific modification, i. e. they use exactly the same files for displaying exactly the same structure. Figure 4 shows a screenshot of the directory structure and the integration in every tool.
This proves that one design goal of Modelica and the Modelica Standard Library (MSL) has been reached now, namely to provide a tool-vendor independent representation format for simulation models. There are however still a few issues to be solved to fully reach vendor independence of the MSL:

- The definition of tables in Modelica Standard Library is based on external functions. The implementation of these functions is not part of the library itself and has to be done by tool vendors. In consequence of missing specification the different implementations are not completely compatible.
- With the exclusive usage of external functions it is difficult to adapt the implementation on the requirements of the underlying tool. The substitution of external functions by external objects would improve the implementation capabilities.
- For users of a Modelica tool it is difficult to decide whether a used construct is compatible to Modelica language specification or not (e. g. classDirectory function). All tool dependent extensions of Modelica language should be marked as vendor specific similar to existing vendor specific annotations.
- Modelica libraries often use different version of annotations for graphical objects or attributes which are invalid in the particular context (e.g. fillColor for lines). While several tools ignore such annotations other programs generate error messages, which can be a little bit confusing for users and developers. For that reason a stronger validation of annotations would be preferable.

To create a Software in the Loop setup, the Modelica model is then exported. In previous years, the C code generated by either Dymola or SimulationX from a given Modelica model has been wrapped and compiled for execution by one of the SiL tools described in Section 3. For export, special wrapper code had to be developed for each simulation tool, and even for each version of such a tool, which was time consuming and error prone. Daimler started recently to use the FMI [8] developed within the Modelisar project as an export format for Modelica models. This standard is supported by the latest versions of SimulationX, Dymola, and Silver. This removes the need to maintain version and vendor specific wrapper code, which further improves and speeds up the SiL-based development process.

3 Getting automotive control software into the loop

Daimler uses Silver [5] and its in-house predecessor Backbone to virtually integrate vehicle models and control software on Windows PCs. Tools such as Silver or Backbone are mainly needed to support vari

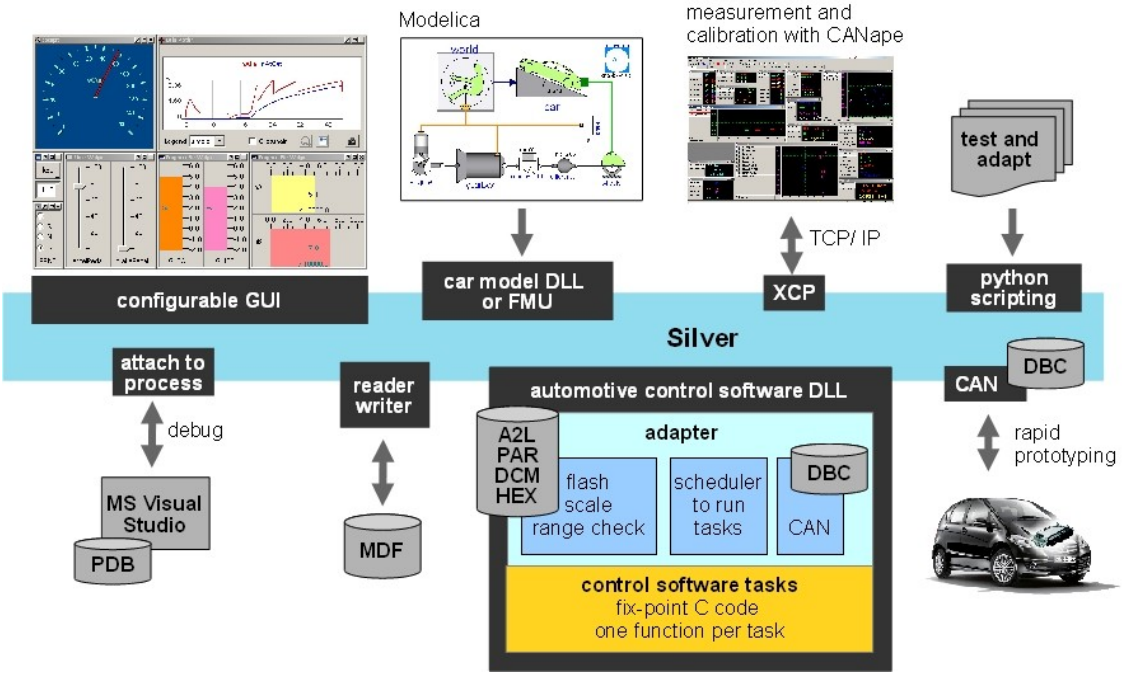

Figure 5: SiL environment an its interfaces to automotive standards

ous standards and quasi-standards used for automotive software development. Developers are familiar with these standards and know how to use them. Data is available in these formats already as part of the existing tool chain and reuse is virtually free of cost. Furthermore, using these data sources in the virtual development process allows early validation of these data sources. A virtual development environment should therefore mimic, emulate, or else how support these standards. A few examples of how the SiL tool supports automotive standards is shown in Fig. 5.

Developers typically use tools such as CANape (Vector) or INCA (ETAS) to measure signals and calibrate (fine-tune) parameters of the control software in the running car or on a test rig using standard protocols such as CCP or XCP. The SiL environment implements this protocol. Seen from a measurement tool such as CANape, a SiL simulation behaves just like a real car. Developers can therefore attach his favorite measurement tool to the SiL to measure and calibrate using the same measurement masks, data sources and procedures they are using in a real car.

Likewise, automotive developers use MDF files to store measurements. The SiL can load and save this file format. A measured MDF file can e. g. be used to drive a SiL simulation.

Another example is A2L. This is a database format used to store key information about variables and (tunable) parameters of automotive control software. A2L contains e. g. the address of variables in the ECU, its physical unit, comment and scaling information that tells how to convert the raw integer value to a physical value. The SiL-environment reads A2L files and uses the information to automate many tasks, such as scaling of the integer variables of the control software to match the physical variables of the vehicle model.

The SiL-environment also knows how to read DBC files. These describe how the control software communicates with other controllers using the CAN protocol. The SiL-environment uses this e. g. to implement rapid prototyping: Load the control software and the DBC into the SiL tool on your laptop, connect the laptop to car using a CAN card, and switch the ECU to 'remote control' mode. The control software running in the SiL tool controls then the corresponding system of the real car, e.g. an automatic transmission. The main advantage of such a setup is, that it saves time. Getting the control software running in a real ECU is typically much more time consuming than using a SiL tool or any other tool for rapid prototyping.

Finally, the SiL tool can process PAR and HEX files. These files may contain calibration data, i. e. values for all the tunable parameters of the control software. The SiL tool knows how to load these values into the control software running in the SiL, emulating thereby the 'flash' process of the real ECU. In effect, the SiL tool is actually not only running the control software, but the fine-tuned version of the software, which enables much more detailed investigation and testing of the control software's performance.

Having all these standards available in the SiL eases the task of actually getting automotive control software running on a PC, and doing useful things with the resulting setup. Control software is typically decomposed into a number of so-called tasks (i. e. functions implemented in C) that are run by an RTOS (real-time operating system) such as OSEK. Many tasks are periodically executed with a fixed rate, e. g. every 10 ms. To get such tasks running in SiL, the user has to build an adapter as shown in Fig. 5, i. e. a little C program that implements the Silver module API and emulates the RTOS by calling each task once at every (or every 2nd, 3rd, ...) SiL macro step. The SiL tool is shipped with the SBS (Silver Basis Software), i. e. C sources that make it easy to build such an adapter by adapting template adapter code. A cheap alternative to writing an adapter is to use the SiL tool's support for MATLAB/Simulink and Realtime Workshop (RTW). Automotive software is often developed by first creating a model of the controller using Simulink. The model is then used to automatically generate fix-point integer code, e. g. using tools like the Embedded Coder from MathWorks, TargetLink from dSPACE, or Ascet from ETAS (model-based development). The SiL tool contains support for exporting a Simulink model using RTW. The result will not use fix-point integer but floating point arithmetic, so it is Model-in-the-loop (MiL), as opposed to Software-in-the-loop (SiL). This is a fast push button solution for exporting a controller model to SiL, which does not require any hand coding, and is therefore attractive.

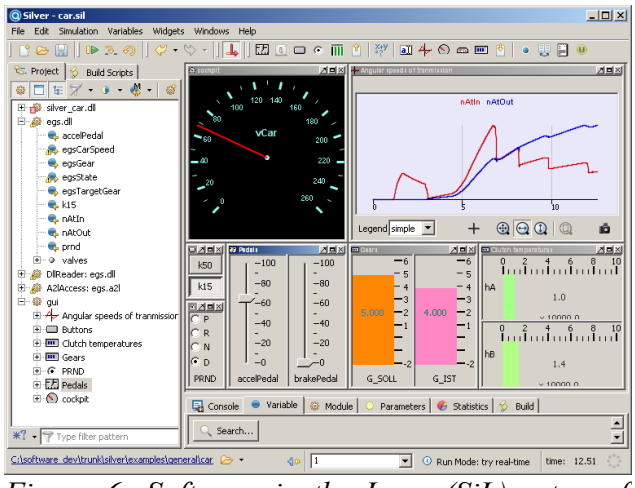

Figure 6: Software in the Loop (SiL) setup of transmission control software and vehicle model

4 Using the system model during automotive development

So far we have mainly described what is needed to get automotive control software running on a Windows PC, in a closed loop with the simulated vehicle. This section describes how such a SiL setup can then be used to support the development process. Supported activities include

- *Virtual integration*: Automotive control software for a single ECU typically consists of dozens of software modules, developed independently by a team of developers. Having a SiL helps to detect problems in the interplay of these modules early, long before an attempt is made to run all the module in a real car. For example, before releasing a new version of his module, a developer can quickly check on his PC whether the module works together with the modules of other developers. To do this, he only needs access to compiled modules (object files), not to the sources of other modules [2]. An additional benefit here is the isolation of developers from the changes of others when validating their modifications early on as his changes are only local to his own sources. Later integration efforts build on modifications already validated, albeit in isolation.
- *Debugging*: In contrast to the situation in a real car or on a HiL test rig, simulation can be halted in SiL. It is then possible to inspect all variables, or to change certain values to simulate a fault event. In conjunction with a debugger (such as Microsoft Visual Studio), it is even possible to set breakpoints or to single-step through the controller code, while staying in closed loop with the simulated car. The SiL tool can also be used to debug problems measured in a real car, if a measurement file (MDF) is available. In this case, simulation is driven by the measurement, and the SiL complements this measurement by computing the missing signals to provide a full picture needed to debug the problem.
- *Fault simulation*: Using a SiL, it is possible to create and explore scenarios that would be difficult or impossible to realize in a real car or on a test rig. For example, you can simulate strong wind [7] or inject arbitrary component faults into the simulation.
- *Comparing versions*: The SiL tool offers a function to compare the behavior of different

software versions by comparing all signals computed by these versions. This is e. g. useful when checking for equivalence after refactoring or clean up of modules.

- *Scripting*: A SiL simulation can be driven by a script, written e. g. in Python. This can be used to implement optimization procedures, for performing tests, or to trigger self-learning algorithms that adapt the control software to certain properties of the (simulated) car, e. g. to compensate aging of components.
- *Systematic testing*: In conjunction with the test case generator TestWeaver, the SiL tool allows the systematic testing of control software. TestWeaver generates thousands of test cases which are then executed by the SiL tool.
- *Virtual endurance testing*: calculation of load collectives for gearbox and drivetrain, e. g. to develop and test measures for safeguarding of the drivetrain components.
- *Application/Calibration*: of the control software on the PC.

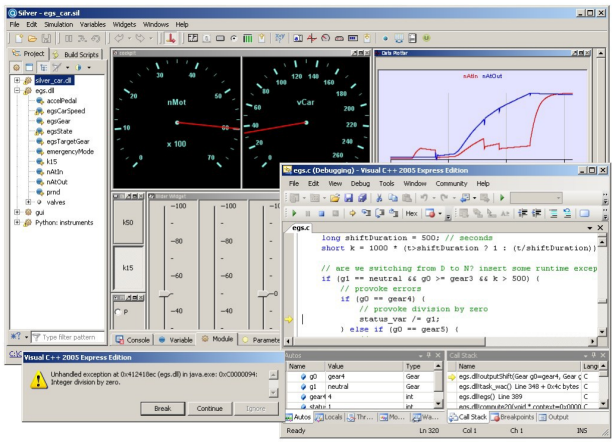

Figure 7: A debugger attached to Silver

A typical use case of the SiL tool is shown in Fig. 7. The test case generator TestWeaver [8] has found a scenario where the control software of a transmission performs a division by zero. This is clearly a bug. The user replays the recorded scenario, with Microsoft Visual Studio attached to the SiL tool. When the division by zero occurs, the debugger pops up as shown in the figure, showing the line in the controller source code that causes the exception.

5 Costs and benefits

Main cost factors of using the simulation-based tool chain for automotive software development are

- *development and maintenance of the simulation model*: Here is where modern modeling languages and tools such as Modelica and SimulationX help reduce costs by reuse of components and easy parameterization
- *continuous calibration efforts* to keep such a model up to date with the plant simulated: SimulationX allows continuous enhancements based on existing models and libraries by replacing components and models of varying complexity throughout all development phases. Reusing models including all interfaces necessary for calibration in combination with a wide range of tool options, e. g. VariantsWizard, COM-scripting or optimization tools, leads to an increasing efficiency in the workflow.
- *Building the adapter code for the controller software*: With the introduction of the Silver Basic Software package, this effort is significantly reduced.

Despite continuing cost-reduction efforts, these investments are still significant.

They are compensated by the benefits of such a Software in the Loop setup for developing control software, namely

- *extremely fast development cycles*: due to comfortable integration of software and vehicle components on the PC of the developer. This helps to detect problems early.
- *excellent debugging and test support*, e. g. with Microsoft Visual Studio Debugger or QTronic TestWeaver [1,2,3,6]. Found problems can be exactly reproduced as often as needed.
- *parallelize the development process*: A SiL configuration can easily be duplicated at low cost. This way, every member of a team can use its personal 'virtual' development environment 24 hours a day, without blocking rare resources like HiL test rigs, or physical prototypes.
- *sharing results without sharing IP*: All members of a team exchange working results by exchanging compiled modules (DLLs), not sources. This helps to protect intellectual property.
- *executing others contributions without their tools*: Our SiL runs modules (simulation models, control software) developed using

very different tools without accessing these tools. This greatly reduces the complexity of the SiL setups (no tool coupling).

6 Conclusion

We presented the tool chain used by Daimler for simulation-based development of transmission control software. The environment is based on Modelica, provides build-in support for automotive standards, imports vehicle models via the standard FMI and uses these models to perform closed-loop simulation of automotive control software. The virtual development environment created this way helps to shorten development cycles, eases test and debugging, helps to parallelize and hence to speed up development and provides a convenient platform for collaboration between Daimler's transmission development departments and its suppliers and engineering service providers.

Acknowledgments

Our work on the FMI [8] presented here has been funded by the Federal Ministry for Education and Science (BMBF) within the ITEA2 project MODELISAR (Förderkennzeichen 01IS08002).

References

[1] A. Rink, E. Chrisofakis, M. Tatar: Automating Test of Control Software - Method for Automatic TestGeneration. ATZelektronik 6/2009 Volume 4, pp. 24-27.

[2] H. Brückmann, J. Strenkert, U. Keller, B. Wiesner, A. Junghanns: Model-based Development of a Dual-Clutch Transmission using Rapid Prototyping and SiL. International VDI Congress Transmissions in Vehicles 2009, Friedrichshafen, Germany, 30.06.-01-07.2009

[3] M. Hart, R. Schaich, T. Breitinger, M. Tatar: Automated test of the AMG Speedshift DCT control software 9th International CTI Symposium Innovative Automotive Transmissions, Berlin, 30.11. - 01.12.2010, Berlin, Germany.

[4] SimulationX, http://www.simulationx.com/

[5] Silver, http://qtronic.de/en/silver.html

[6] A. Junghanns, J. Mauss, M. Tatar: TestWeaver - A Tool for Simulation-based Test of Mechatronic Designs. 6th International Modelica Conference, Bielefeld, March 3 - 4, 2008, pp. 341 - 348, 2008.

[7] Hilf, Matheis, Mauss, Rauh: *Automated Simulation of Scenarios to Guide the Development of a Crosswind Stabilization Function.* 6th IFAC Symposium on Advances in Automotive Control, Munich, Germany, July 12 - 14, 2010.

[8] FMI Specification 1.0, available for free from http://www.functional-mockup-interface.org/