Virtual integration of Automotive Hard- and Software with Silver

Andreas Junghanns
QTronic GmbH, Alt-Moabit 91d, D-10559 Berlin

Abstract

Silver is a tool used by automotive development engineers to integrate and test control software virtually using simulation on Windows PCs. Silver provides built-in support for automotive standards such as A2L, MDF, CAN, and XCP to perform co-execution of control software and vehicle simulation models. In this paper, we describe Silver and its application to automotive software development.

1 Introduction

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

- be easy to set up and use by automotive developers
- support short turn-around times, i.e. minimize the time between editing of control software and validation of the resulting behaviour on system level
- provide built-in support for standards and de-facto standards used in automotive software development
- support distributed development and exchange of work products between OEMs, suppliers, and engineering service providers. This requires e.g. measures to protect intellectual property contained in model sources.
- 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.

Silver has been designed to address all these requirements.

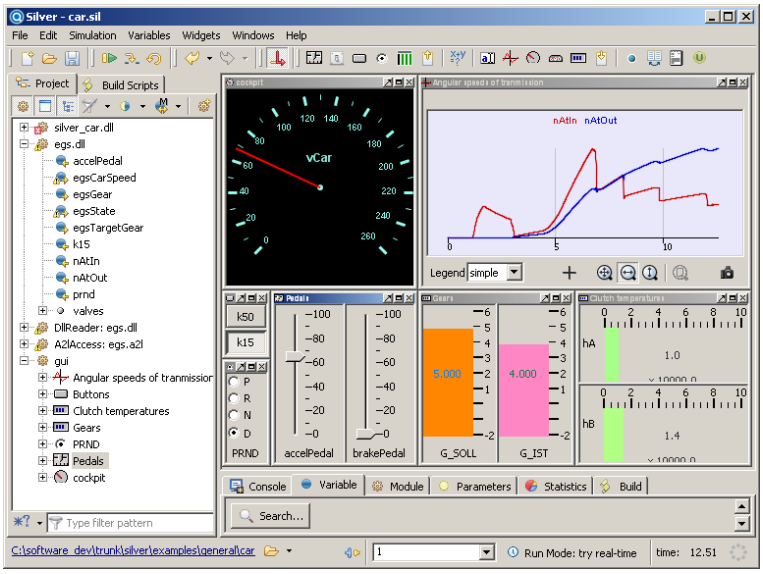

Fig 1: Automotive control software running in Silver

Technically, Silver is a tool that allows several executable programs called *modules* to share data during their execution on a Windows PC. Typically, a module contains control software, or is a simulation model of the controlled system. The modules participating in a Silver simulation exchange variable values at fixed macro steps (typically 1 to 20 ms), such that output values computed by one module at macro step n serve as input for other modules at macro step $n + 1$. Modules may be self-integrating or use Silver's own ODE or DAE solver to perform numerical integration.

Silver provides a rich set of widgets used to configure, save and load a graphical experiment environment for a set of modules. Fig. 1 shows an example of such a user specified interface for an automotive transmission controller.

To minimize work load for setting up a co-simulation, Silver does not require that a user defines the signal flow between modules by defining 'interface objects' or by connecting the inputs and outputs of the modules with each other. Instead, such connections are made automatically, based on variable names and flow direction (input, output or undeclared) of the variables as declared by the modules that contain the variables. Silver provides means to introduce alias names for variables in cases where automatic matching fails. All this helps to build and maintain Silver setups involving thousands of variables.

2 Getting automotive control software into the Silver loop

Automotive software development involves many standards and quasi-standards. Developers are familiar with these standards and know how to use them. A virtual development environment such as Silver should therefore mimic, emulate, or else how support these standards. A few examples of how Silver supports this is shown in Fig. 2.

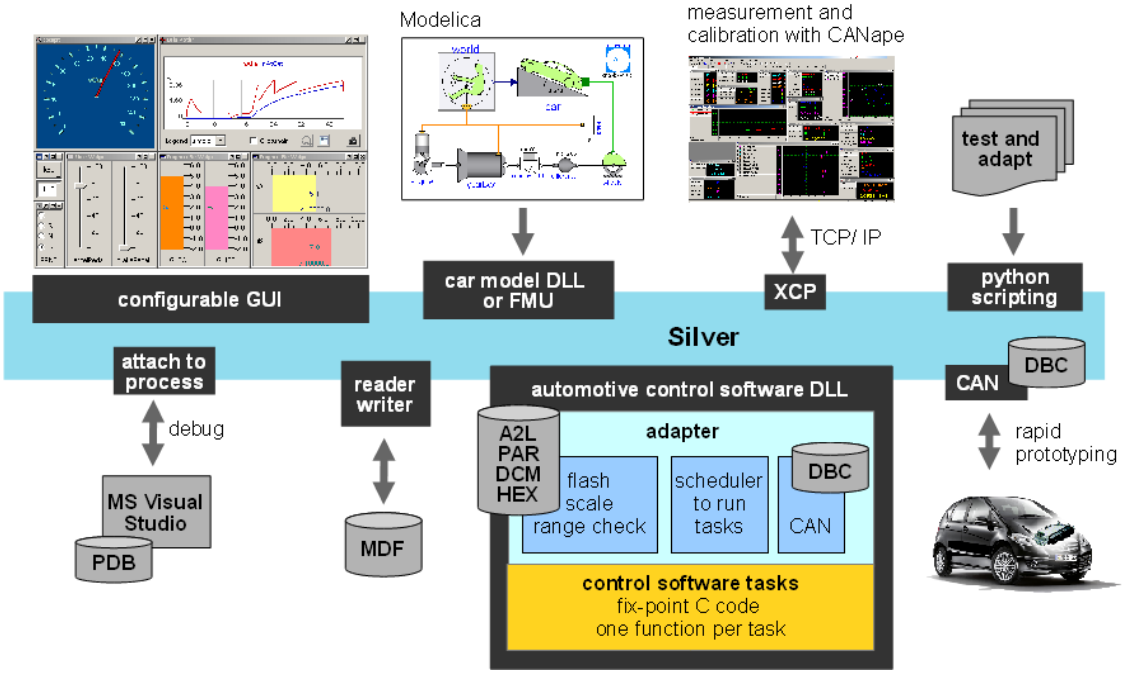

Fig 2: Automotive standards supported by Silver

Developers typically use tools such as CANape or INCA 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. Silver implements this protocol. Seen from a measurement tool such as CANape, a Silver simulation behaves like a real car. A developer can therefore attach his favourite measurement tool to Silver to measure and calibrate using the same measurement masks and procedures he is using in a real car.

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

Another example is A2L. This is a database format used to store key information about all signals (variables) and tunable parameters of automotive control software. A2L contains e.g. the address of each variable in the

ECU, its physical unit, comment, and scaling information that tells how to convert the raw integer value to a physical value. Silver 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.

Silver also knows how to read DBC files. These describe how the control software communicates with other controllers using the CAN protocol. Silver uses this e.g. to implement rapid prototyping: Load the control software and the DBC into Silver 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 Silver controls then the corresponding aggregate 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 Silver (or any other tool for rapid prototyping).

Finally, Silver can process PAR and HEX files. These files may contain calibration data, i.e. values for all the tunable parameters of the control software. Silver knows how to load these values into the control software running in Silver, emulating thereby the 'flash' process of the real ECU. In effect, Silver 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 Silver eases the task of actually getting automotive control software running in Silver, 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 just periodically executed with a fixed rate, e.g. every 10 ms. To get such tasks running in Silver, the user has to build an *adapter* as shown in Fig. 2, 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, ...) Silver macro step. Silver 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 Silver's support for MATLAB/Simulink 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). Silver contains support for exporting a Simulink model using RTW. The result will not use fix-point integer but floating point arithmetic, so its just *Model-in-the-loop* (MiL), as opposed to *Software-in-the-loop* (SiL). On the other hand, this is a fast push button solution for exporting a controller model to Silver, which does not require any hand coding, and is therefore attractive.

3 Importing simulation models into Silver

Silver offers means to import models from AMESim, Dymola, SimulationX and SIMPACK. Since Silver 2.0 however, the preferred way to exchange models is via the FMI, a standard for the exchange of models, that was recently published [9]. Dymola, and SimulationX already offer a push-button solution for exporting a model as FMU. You can run one or more FMUs simultaneously in Silver. Silver handles the case of multiple FMUs by converting the ODEs defined by the FMUs into a single DAE (with connections between FMUs showing up as algebraic constraints) and solving the resulting system using a DAE solver.

It is also possible to run a MATLAB/Simulink S-function, (i. e. a .mexw32 file) directly in Silver. Such a function can e.g. be hand coded in C or generated in MATLAB from a Simulink subsystem block using the S-function target of Simulink/RTW.

4 Using Silver during automotive development

So far we have mainly described what is needed to get automotive control software running in Silver, 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 to this, he only needs access to compiled modules (object files), not to the sources of other modules [2].

- *Debugging*: In contrast to the situation in a real car or on a HiL test rig, simulation can be halted in Silver. 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 step through the controller code, while staying in closed loop with the simulated car. Silver 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 Silver complements this measurement by computing the missing unmeasured signals to provide a full picture needed to debug the problem.
- *Fault simulation*: With Silver, 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 [4] or inject arbitrary component faults [3] into the simulation.
- *Comparing versions*: Silver offers a function to compare the behaviour of different software versions by comparing all signals computed by these versions. This is e.g. useful when checking for equivalence after implementation or clean up of modules.
- *Scripting:* A Silver 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 ageing of components.
- *Systematic testing*: In conjunction with the test case generator TestWeaver, Silver allows the systematic testing of control software. TestWeaver generates thousands of test cases which are then executed by Silver. This process contains an optimization procedure that tries to construct 'worst-case scenarios' automatically, based on a good-bad scale provided by the user.

A typical use case of Silver is shown in Fig. 3. The test case generator TestWeaver [7] 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 Silver. 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 of using Silver

Main cost factors of using Silver for automotive software development are the costs for development and maintenance of the simulation model and of the adapter code (see Section 2). These investments must be justified by the expected benefit of such a Software in the loop (SiL) setup for developing control software, which is

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. Found problems can be exactly reproduced as often as needed.

parallelize the development process: A Silver 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.

s**haring results without sharing IP**: With Silver, 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: Silver runs modules (simulation models, control software) developed using very different tools. Silver runs these modules without accessing the corresponding tools. This greatly reduces the complexity of Silver setups (no tool coupling).

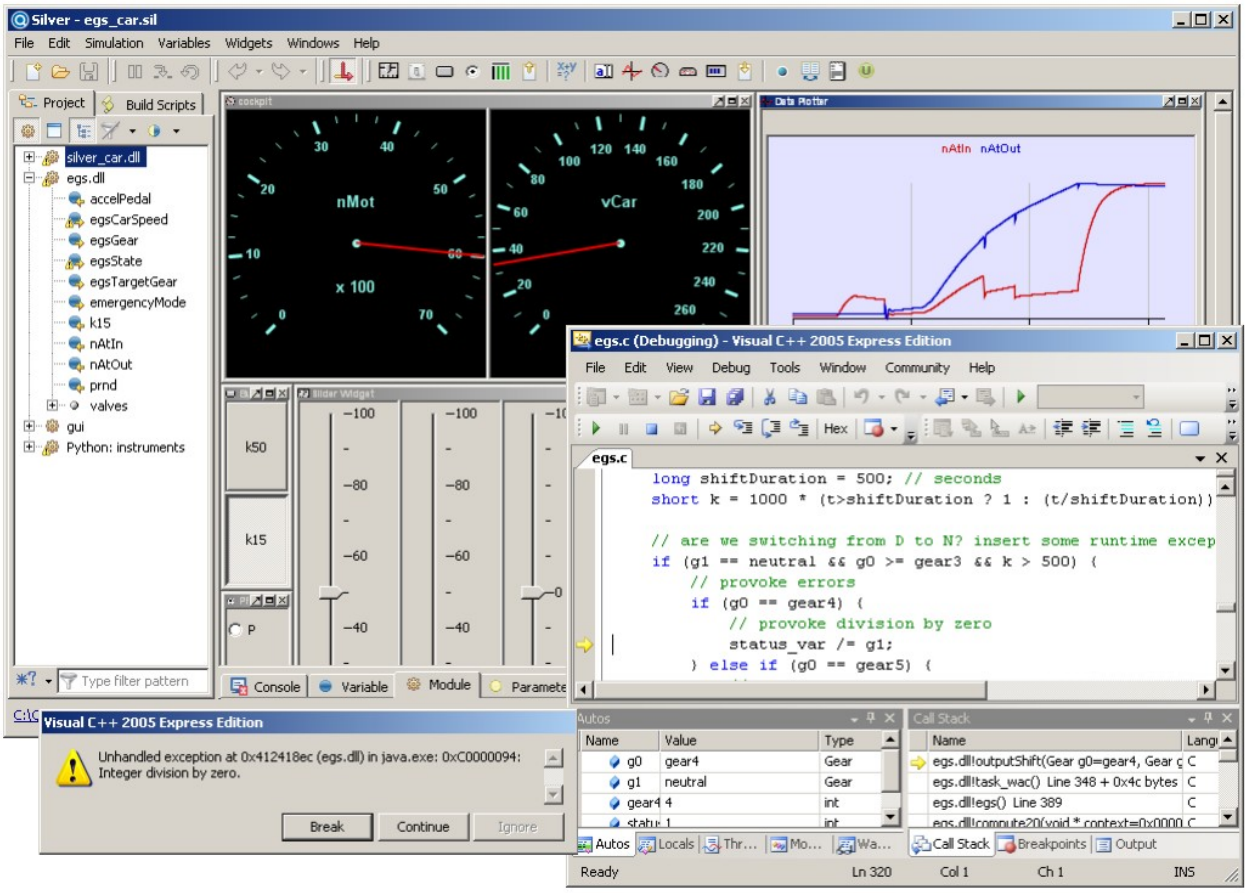

Fig 3: A debugger attached to Silver

6 Automotive applications of Silver

Today, Silver is in use for automotive software development at Mercedes-Benz, AMG, Continental, IAV and others. Since its initial release in May 2008, Silver has been used in many automotive development projects, often in combination with QTronic's test case generator TestWeaver [6]. Reference applications include

- Development of the control software for the AMG SPEEDSHIFT MCT 7-speed sports transmission, based on the 7GTronic transmission and released April 2008 in the SL 63 AMG. Silver was used as platform for virtual integration of the software modules (SiL), and as test execution platform [1]. Every release of the control software was stress tested on several Silver installations on Windows PC. For each test run, thousands of driving scenarios where generated by TestWeaver [7], and evaluated using Silver.
- Development of a dual clutch transmission by Mercedes-Benz [2]. This application of Silver is similar to [1]. In addition, a tool for measuring code coverage has been integrated into the SiL setup. The controller code has been developed using Simulink / TargetLink.
- Development of a brake assistance function for heavy trucks [3]. Silver was used to virtually integrate control software developed with Simulink and a vehicle model developed in Modelica. The resulting Silver MiL setup was then tested using TestWeaver, with focus on the fault-tolerance of the brake system.
- Development of a crosswind stabilization function for the 2009 S class [4]. For this application, a Simulink model exported using Real-time workshop was co-simulated with a vehicle model developed using Daimler's vehicle simulation system CASCaDE, together with a wind and road model developed in C. TestWeaver was then used on top of Silver to generate about 100000 different driv-

ing scenarios, each 45 sec., with varying road and wind conditions. The entire setup was used during development to iteratively improve key algorithms of the stabilization function.

- Development of the AMG Speedshift DCT control software, used in the Mercedes-Benz SLS AMG super sports car [5]. This application is similar to the one reported in [3].

Full papers with details on all these applications can be downloaded form [6] and [7]. All applications listed above, except the fourth one, use Modelica [8] as the language for physical modelling. Starting with release 2.0, Silver provides also support for on-line calibration of Modelica models (tunable parameters), which considerably speeds up the calibration of vehicle models.

7 Conclusion

We presented Silver, a tool for integration, simulation and test of automotive software on Windows PCs. Silver provides build-in support for automotive standards, imports models from various simulation tools 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 OEMs, suppliers and engineering service providers.

References

[1] A. Rink, E. Chrisofakis, M. Tatar: Automating Test of Control Software - Method for Automatic Test Generation. 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] Gäfvert et al.: Simulation-Based Automated Verification of Safety-Critical Chassis-Control Systems. 9th International Symposium on Advanced Vehicle Control (AVEC2008), Kobe, Japan, 6. - 9.10.2008.

[4] K.-D. Hilf, I. Matheis, J. Mauss, J. 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.

[5] 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.

[6] Silver Product Information, http://qtronic.de/en/silver.html

[7] TestWeaver Product Information, http://qtronic.de/en/weaver.html

[8] Modelica Association, http://www.modelica.org

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