2012-01-0962

AUTOSAR-Compliant Development Workflows: From Architecture to Implementation – Tool Interoperability for Round-Trip Engineering and Verification & Validation

Guido Sandmann
MathWorks GmbH

Michael Seibt
Mentor Graphics GmbH

ABSTRACT

AUTOSAR is on the road. Many OEMs and suppliers have established processes that are project-proven, flexible, and efficient for developing AUTOSAR-compliant applications. These development processes require a variety of tools that support requirements management, system architecture development, Model-Based Design, and verification and validation. Hence, interoperability of these tools is essential for the completion of high-quality projects on time. Building an interface from one tool to the next is often insufficient because processes for production projects are typically more complex than a top-down or bottom-up flow. However, a tool chain must support iterative development across all phases. For example, when a change in customer requirements triggers modifications to the software architecture, definitions for components, runnables, interfaces, or ports all need to be updated accordingly. Alternatively, while modeling the functional design, an engineer may realize that additional sensor data requirements can trigger a change in the software architecture.

AUTOSAR offers standardized formats that allow a consistent exchange of data between tools and development phases. To realize, test, and implement software components, software component description files can be exported from system architecture tools and imported into tools for Model-Based Design. The challenge of these processes is to ensure consistent data exchange between phases without losing or corrupting design information.

This paper describes the interoperability of tools that support Model-Based Design and are used to create software architectures. Specifically, this paper shows the mechanisms defined by AUTOSAR and capabilities to verify and validate these designs at different stages in the workflow.

INTRODUCTION - WORKFLOW OVERVIEW

In an AUTOSAR-compliant development process, it is crucial that the development tools work well together and provide seamless round-trip engineering. This compatibility enables engineers in the automotive industry to follow iterative development processes. AUTOSAR authoring tools help engineers create architectures that describe vehicle electronics and software functions. Tools that model the behavior of these functions specify their functional algorithms using Model-Based Design.

Figure 1 illustrates a high-level overview of these workflows. Starting top-down from an architecture model developed in a tool such as VSA from Mentor Graphics, the AUTOSAR software component description files are exported through AUTOSAR standardized format (ARXML). These description files are imported into a behavior modeling tool such as Simulink® from MathWorks. In Simulink, functional behavior is modeled and tested – by simulation, verification, and validation – until the model is ready for production code generation using Embedded Coder. When generating production code, updated software component description files are also created simultaneously. This helps to ensure consistency between in the resulting artifacts (C-code and SWC description files). The architecture tool imports and processes these files for integration and download to the target.

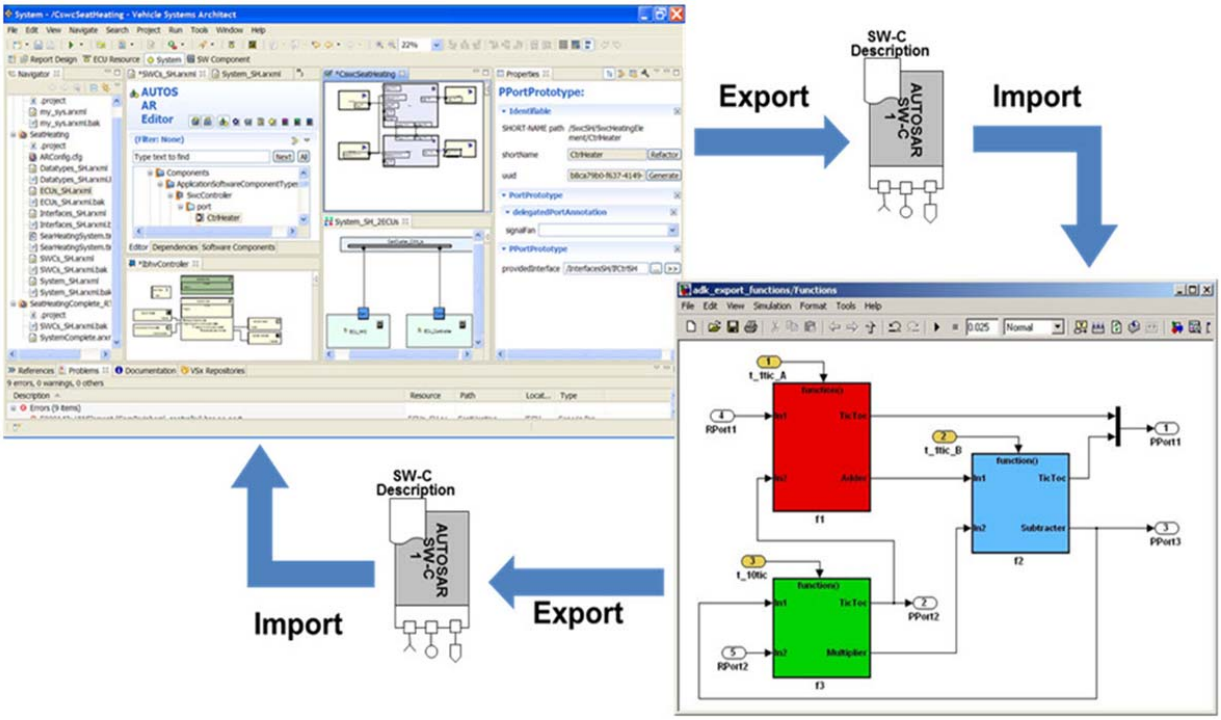

Figure 1 - AUTOSAR-compliant workflows for enabling round-trip engineering.

DESIGNING SOFTWARE ARCHITECTURES FOR AUTOSAR WORKFLOWS

One entry point of AUTOSAR-compliant system design is the creation of software architecture in terms of software components and optionally their internal behavior. The individual components are connected to a top-level software composition that defines the overall software architecture and is typically developed with an AUTOSAR authoring tool. In parallel, the ECU topology is defined as a prerequisite for the software-to-ECU-mapping. This mapping determines intra-ECU and inter-ECU communication. Inter-ECU communication is realized through system signals, which transport the information as data elements through a communication cluster (CAN, LIN, or Flexray) between the ECUs. These data elements need to be mapped to system signals. To achieve the most efficient bus communication and to reduce the busload, signals are then mapped into frames and in most cases, this mapping is predefined through the OEM. Some OEMs compute this mapping based on timing constraints. Finally, the information for a dedicated ECU, selected by the ECU developer, is extracted from the overall system description. The generated file, the ECU extract, serves as input to the basic software configuration.

The different steps performed with an AUTOSAR-compliant authoring tool require iterative verification though consistency checks and design rule checks (DRCs). Figure 2 shows VSA with a top-level software composition, including an excerpt on internal behavior from the software component (SW-C) description file. VSA provides different domain-specific editors for creating designs with varying levels of complexity. Depending on the requirements, engineers can choose a tree-based editor, tabular editor, or graphical editor.

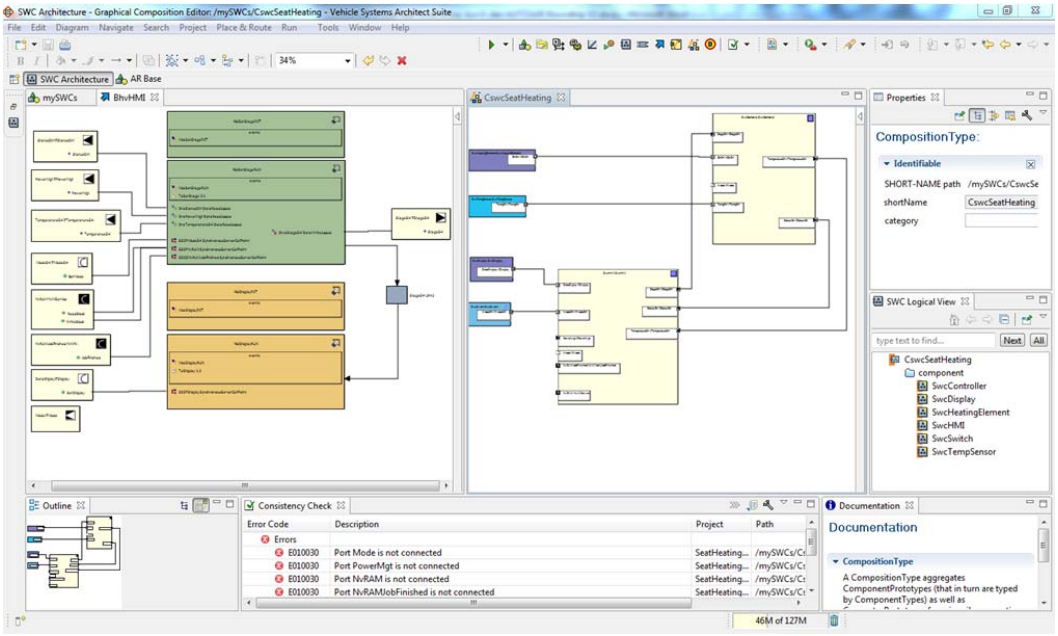

Figure 2 - VSA tool for creating AUTOSAR-compliant software architectures.

Existing software components, interface specifications, and other design artifacts can also be imported as libraries and reused in the new design. Through the VSA meta-model technology, a complete AUTOSAR-compliant model specification is available to support a complete export, including different AUTOSAR versions with respective model-to-model transformations. This support is provided to Simulink through the export of a complete software component description and a MATLAB script that enables the import into Simulink.

AUTOSAR SOFTWARE COMPONENTS WITH MODEL-BASED DESIGN

As described earlier in this paper, software architectures are one possible entry point for developing the functional behavior of the components. In this particular case, engineers reference a top-down workflow where an AUTOSAR authoring tool such as VSA provides the software architecture. For Model-Based Design, AUTOSAR XML formats (ARXML) describing AUTOSAR software components are exported from the authoring tool and imported into a behavior modeling environment such as Simulink. This step builds the structural information of the model.

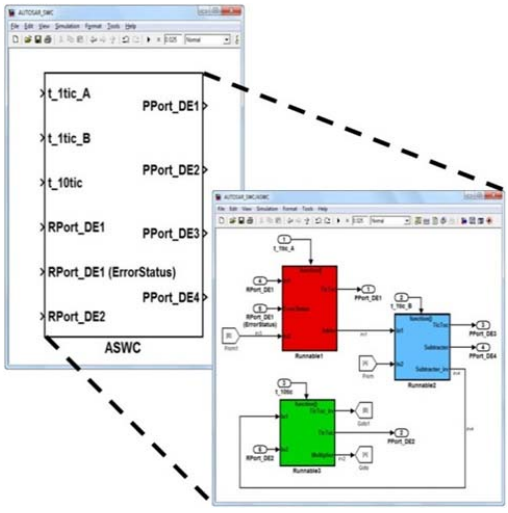

Figure 3 - Imported SW-C and internal behavior.

Figure 3 shows the result of the import: the creation of a skeleton model of the software component description file (SW-C). This skeleton model is represented as a Simulink subsystem (top left) that consists of the interfaces, internal behavior (runnables (bottom right colored), represented as function-call subsystems), and communication between the runnables via inter-runnable variables (IRV). In this subsystem, engineers use Model-Based Design to develop the algorithmic behavior.

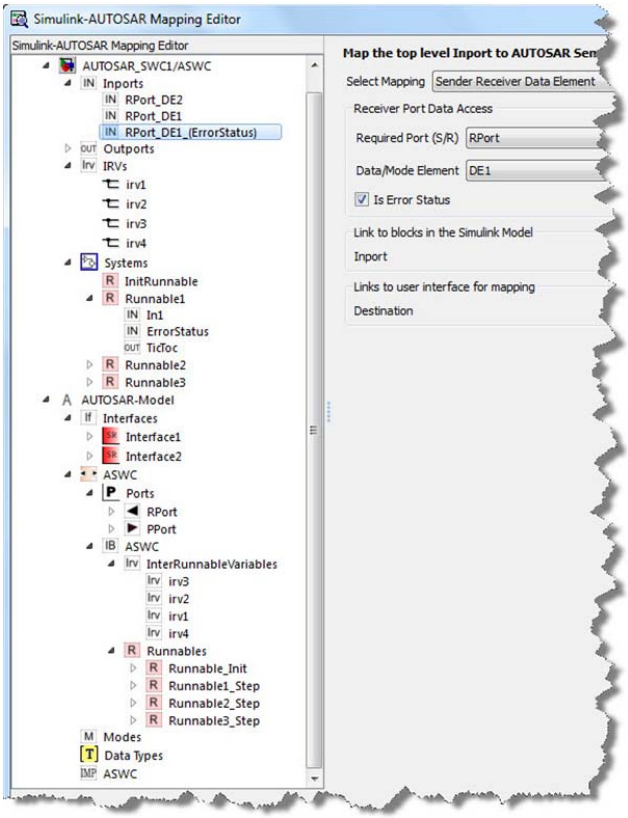

Figure 4 - Simulink-AUTOSAR Mapping Editor

The skeleton model is the basis for refining the model iteratively with algorithmic details. As Figure 4 shows, with the import of the SW-C description file, the necessary information for modeling AUTOSAR software components within Simulink is available and accessible through the Simulink-AUTOSAR Mapping Editor. The AUTOSAR-model part of the project tree contains all AUTOSAR definitions imported from the ARXML-file. Additionally, the engineer can modify or add definitions, such as new Sender/Receiver or Client/Server Interfaces, and map these definitions to explicit Simulink artifacts listed in the upper part of the tree.

This explicit mapping helps preserve all information provided by the authoring tool and to keep the AUTOSAR information and the Simulink model synchronized throughout the round-trip engineering. The final definitions can be exported collectively into the SW-C description file and remain consistent with C code generated in compliance with AUTOSAR.

In this phase of the process, engineers are following mixed workflows (top-down versus bottom-up). In the top-down workflow, after the systems architect provides the software architecture and final software component description to the function designer, the function designer returns adaptions and modifications of the SW-C description files to the systems engineer who then integrates this information with the overall architecture.

At this point, interoperability between the architecture tool and the tool for modeling internal behavior is essential.

AUTOSAR SOFTWARE COMPONENT TESTING

AUTOSAR-compliant software components are tested within a simulation environment such as Simulink. Figure 5 illustrates the test environment for a sample software component. To realize a scheduling scheme of RTE events for triggering runnables of the software component, Stateflow, the extension to Simulink for developing state charts and flow graphs, is often implemented. Scheduling can be either periodical or event driven. Depending on the application domain, open-loop or closed-loop testing approaches are available.

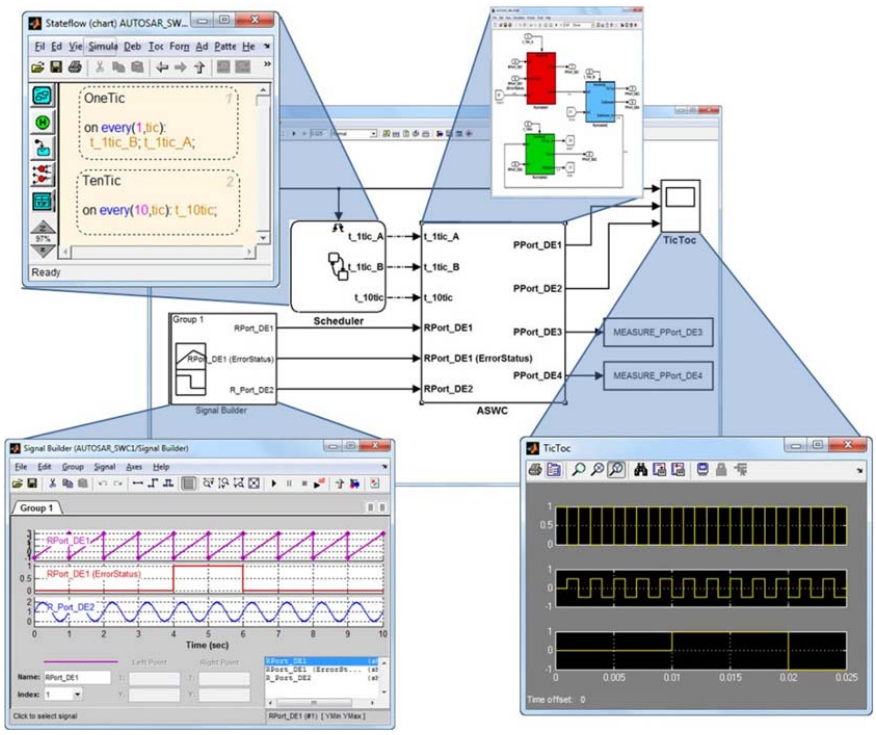

Figure 5 - Model-in-the-loop (MIL) testing of SWC description files.

Figure 5 also highlights the Simulink Signal Builder block. This tool enables engineers to define test stimuli for software components that can be reused across all steps of the component verification and validation process. After model-in-the-loop (MIL) testing the component's functional behavior, AUTOSAR-compliant target C code can be generated for software-in-the-loop (SIL) testing by automatically generating an S-Function block. For regression testing, the model can then be replaced by the SIL block as illustrated in Figure 6.

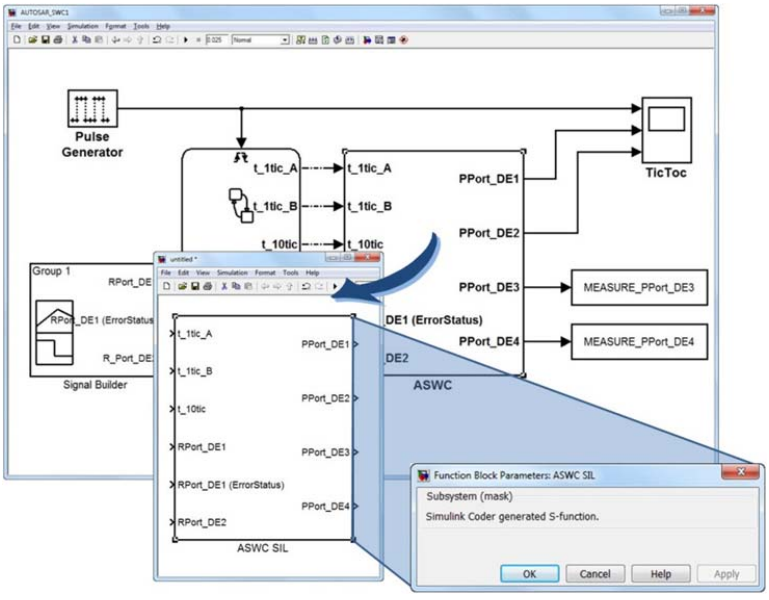

Figure 6 - Software-in-the-loop (SIL) testing by creating an S-Function block.

INTEGRATION TESTING OF AUTOSAR SOFTWARE COMPONENTS AND COMPOSITIONS

Model-based simulation and verification can be performed early in the development phase, without hardware. Only a subset of information is required to achieve initial results on the system's functionality. Changes can be made quickly to influence and optimize the software's overall behavior. Based on the abstract model description, C code is automatically generated and includes more information such as the implementation of different AUTOSAR access point types. This additional information can lead to behavior different than the original simulation in terms of timing and functionality. Moreover, communication between the different software component ports is managed through an AUTOSAR RTE (Run Time Environment) , which implements different AUTOSAR features, such as communication modes (implicit and explicit), concurrency, or multiplicity.

Scheduling of the software on the target ECU is realized through the OS. Instead of performing expensive test cycles directly on the real ECU, C-code integration testing is recommended to test the integration of application C code (typically comprising multiple software components), the RTE, and the operating system.

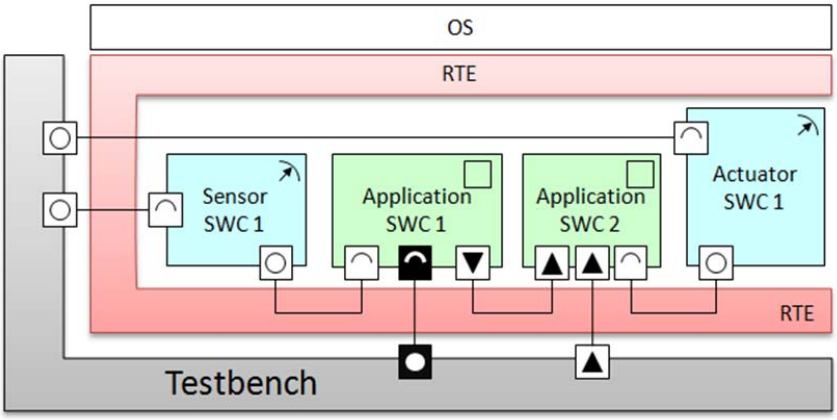

Figure 7 - Testing application code with the RTE and OS.

Through vehicle system integrator (VSI), individual software components and software architectures can be tested. As Figure 7 shows, application code can be tested by taking the real AUTOSAR RTE and OS into account. As simulation is performed on the virtual function bus (VFB) level next to the application code, no additional information is necessary for integration testing on this level. Additionally, the test bench previously defined in the Simulink environment can be reused.

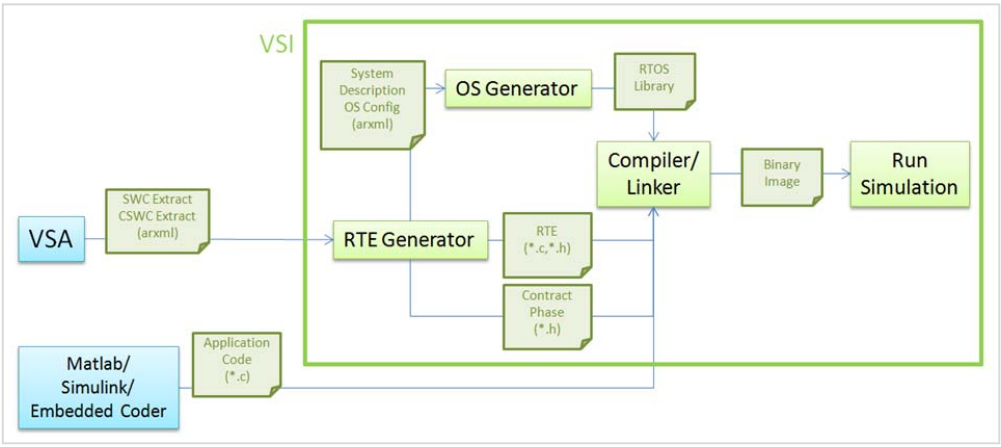

Figure 8 - Automatic configuration of the RTE and OS during build.

The run-time environment and operating system are automatically configured to some extent during the build process as illustrated in Figure 8. However, existing AUTOSAR OS configurations, including their scheduling tables, can be imported and reused in the simulation.

Two basic use cases for software testing occur at this stage: white-box testing and black-box testing. White-box testing is typically performed by software developers who need to validate and test software on a code-line level. In this case, monitoring and debugging capabilities are needed. In contrast, by applying a set of well-defined test-vectors, black-box testing targets the output behavior and its comparison with the specification.

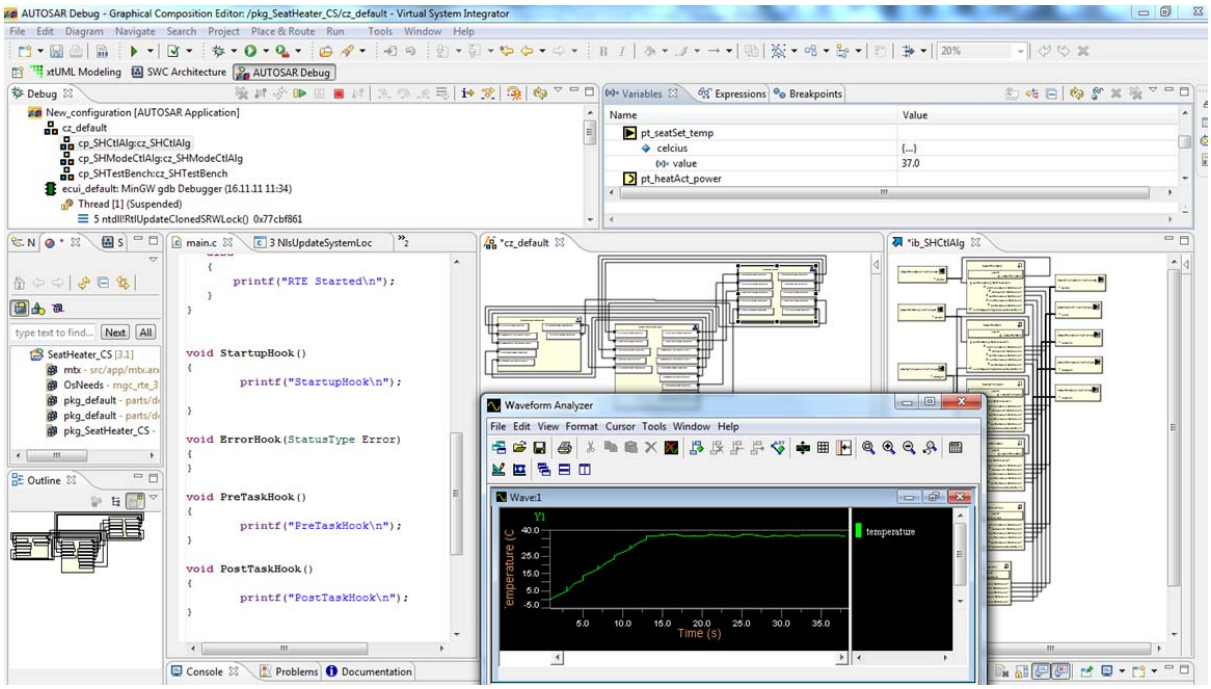

Figure 9 - VSI for monitoring and debugging at C-code level of software component.

As AUTOSAR introduces a standard meta-model and terminology for describing software, AUTOSAR oriented debugging views are needed. Consequently, ECU software can be monitored and debugged on both the C-code level and the runnable software component level, as highlighted in Figure 9.

SUMMARY/CONCLUSIONS

Based on ARXML format and the methodology defined by AUTOSAR, the combination of domain-specific tools represents a comprehensive validation framework that also supports round-trip engineering. VSA as an AUTOSAR authoring tool is the central point for maintaining and administrating all AUTOSAR artifacts. Every AUTOSAR Identifiable gets a unique ID (UUID) and a short name to help track them during the round-trip. Simulink with Embedded Coder serves as the software component design, implementation, and verification tool while enabling AUTOSAR file import and export. VSI can then execute the generated AUTOSAR application code with consideration for an AUTOSAR-compliant OS and RTE.

CONTACT INFORMATION

Guido Sandmann, Automotive Marketing Manager, EMEA, MathWorks

Guido Sandmann works for MathWorks as the automotive marketing manager responsible for the European region. In this technical marketing role, he is responsible for message creation concerning MATLAB and Simulink products and solutions dedicated to the automotive industry. He works closely with customer-facing organizations as well as with customers directly, discussing their requirements and suggested improvements to the company's product portfolio.

Before joining MathWorks, Guido worked in marketing and sales positions for different companies with a focus on verification, validation, and test.

Guido holds a Diploma degree in Computer Science from the University of Oldenburg, Germany. You can contact Guido at Guido.Sandmann@mathworks.de.

Michael Seibt, Product Manager, Mentor Graphics

Michael Seibt works for Mentor Graphics as the product manager for AUTOSAR Tooling. In this position, he acts as an interface between potential customers and Mentor Graphics research and development. This role also involves gathering customer requirements, writing concepts, and introducing products to customers.

Before joining Mentor Graphics, Michael worked for different companies in the field of embedded design and mechatronics simulation.

Michael holds a degree in Aviation Engineering from the Munich University of Applied Sciences, Germany, and a degree in Electrical Engineering from the Salzburg University of Applied Sciences, Austria. You can contact Michael at Michael_Seibt@mentor.com.