Model-Based Design for AUTOSAR Software Components

Ulrich Eisemann

Model-Based Design for AUTOSAR Software Components

Ulrich Eisemann

dSPACE GmbH, Technologiepark 25, 33100 Paderborn, Germany

Abstract: The AUTOSAR initiative is without doubt one of the most forward-looking and important developments in the automotive industry. Tool support is essential for efficient software development according to AUTOSAR, particularly for developing the actual application software in the form of AUTOSAR software components. This paper deals with the adaptation of model-based design and automatic production code generation techniques to the proposed AUTOSAR workflow. It shows that existing approaches are well suited for the development of AUTOSAR software, preserving all the advantages of model-based design such as early testability, precise specifications, and last but not least, automatic production code generation.

Keywords: AUTOSAR, Software Components, Code Generation

1. Introduction

For years now, the automotive industry has faced the problem of development times becoming shorter while the complexity of vehicle software grows. Modern development methods such as model-based design and automatic production code generation have become established as a result, see [1], [2]. Alongside these activities, which are primarily oriented to the actual application part of the software, the AUTOSAR initiative aims to establish an open standard for entire electrics/electronics (E/E) architectures in vehicles, see [3]. The topic of this article is the adaptation of model-based design and automatic production code generation to efficient, AUTOSAR-compliant development of the application software itself, called AUTOSAR software components. A tool chain based on MATLAB®/Simulink®/Stateflow® and TargetLink is taken as an example to demonstrate the concepts.

2. The AUTOSAR Software Architecture

According to the AUTOSAR standard, the software architecture of each AUTOSAR-compliant electronic control unit (ECU) has the general structure shown in Figure 1. The ECU software is basically subdivided into three different layers: the AUTOSAR software components, the Run-Time Environment (RTE), and the basic software. The AUTOSAR software components (SWCs) contain the application's actual functional code, while the basic software serves to abstract from the specific hardware, provides basic services for task management, hardware access, and communication drivers. The Run-Time Environment constitutes the "glue code" between the application software and the basic software, i.e., it provides well-defined, standardized interfaces for data exchange with the services of the basic software and between the AUTOSAR software components themselves. Data exchange is performed exclusively via the RTE, so not only is the application software completely abstracted from the hardware, it is also possible to distribute SWCs across different ECUs in a network independently of one another, without needing any modifications to the SWCs themselves. Thus, each SWC can be developed regardless of whether a second component is run on the same ECU or another. It is up to the RTE to make sure that communication is established between software components and basic software. Depending on the locations of the components, the RTE allows data exchange either directly via a shared memory or by sending messages via a bus. During software development, these differences are taken into account by what is called the RTE Generator, which generates the Run-Time Environment for each ECU appropriately. This approach means that AUTOSAR software components can be deployed fairly freely on different ECUs and different platforms. The reuse and replacement of individual components are also greatly simplified.

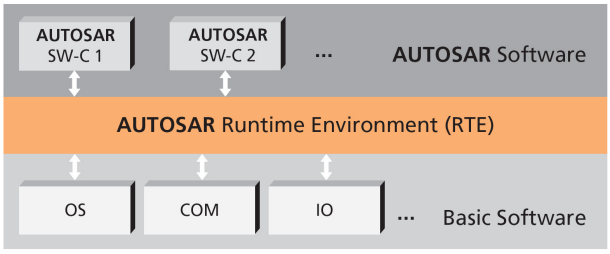

Figure 1: The AUTOSAR software architecture with the layers AUTOSAR software components (SWCs), Run-Time Environment (RTE), and basic software.

3. Model-Based Design for AUTOSAR Software Components

For designing AUTOSAR software components, a model-based approach using automatic production code generation is helpful. This ensures that AUTOSAR-compliant design benefits from the

known advantages of model-based development, such as immediate simulatability and automatic production code generation, leading to considerably reduced development times, cost savings, and enhanced quality. AUTOSAR software components consist not only of AUTOSAR-compliant code, which is basically C code with special RTE macros for communication. In addition, each AUTOSAR SWC also entails a so-called standardized software component description listing all the SWC's structural AUTOSAR elements. The description serves to integrate the software component in an overall AUTOSAR application. Hence, to generate AUTOSAR software components from model-based designs, code and component description files have to be generated, see Figure 2.

To fulfill AUTOSAR requirements, the production code generator TargetLink has been extended to provide comprehensive support for designing AUTOSAR software components and performing automatic production code generation for them. This closes a gap in the area of tool support for AUTOSAR-compliant software development, as the actual application code can now be designed on a model basis with tool support. In conjunction with appropriate OS, COM, and RTE generators, this opens the way to developing AUTOSAR-compliant ECUs with great efficiency in the future.

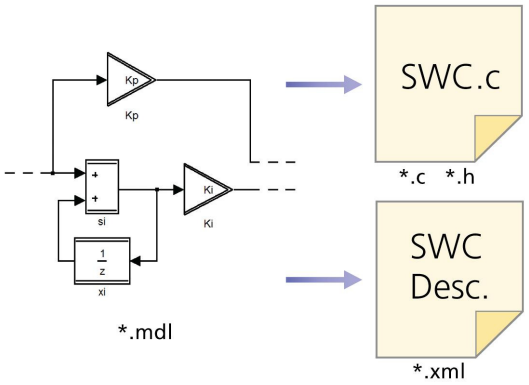

Figure 2: To adapt model-based design to AUTOSAR requirements, it is necessary to generate AUTOSAR-compliant code and software component descriptions in XML format from models.

4. Modeling

As a production code generator, TargetLink is fully integrated in Simulink/Stateflow and enables users to implement ECU functions designed as block and state diagrams directly as production-ready code. For modeling AUTOSAR software components, TargetLink offers users special AUTOSAR blocks, see Figure 3, for specifying AUTOSAR structural elements such as runnable entities and ports. These special AUTOSAR blocks are now combined with the established TargetLink blockset used for modeling the algorithmic parts of the controller.

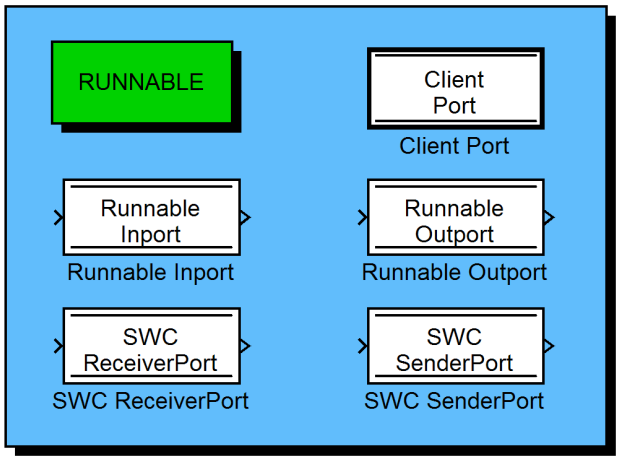

Figure 3: Library with special AUTOSAR blocks for modeling AUTOSAR SWCs.

The TargetLink AUTOSAR blockset consists of the following elements:

- A Runnable block to identify a particular subsystem as a runnable, i.e., an executable unit according to the AUTOSAR standard. Runnables may be triggered periodically or by external events. During code generation, a C function is generated for each runnable.
- Runnable Inports and Runnable Outports to associate signals in TargetLink with individual data elements of an AUTOSAR interface. The Runnable Ports specify the data exchange and communication mechanisms between individual runnables. TargetLink supports interrunnable communication, sender/receiver communication, and synchronous client/server communication, all of which are specified in the AUTOSAR standard. All communication mechanisms translate into special code patterns known as RTE macros during code generation later on.
- Receiver and Sender Ports for software components. These are mostly optional and can be used to highlight the connection between signals in TargetLink and the ports of a software component. The blocks are the direct equivalent of AUTOSAR ports on the model level.
- A Client Port block to model a client call of a method provided by the basic software or by another software component. The most common use cases are service calls, for instance to an NV-RAM manager.

From the modeling and code generation perspective, it makes little difference whether communication takes place between SWCs within the application layer or between an SWC and, for instance, services of the basic software. The latter have well-defined AUTOSAR interfaces just like software components, and communication is therefore specified in a similar way.

When modeling runnables, users can now combine the regular TargetLink blockset with the TargetLink AUTOSAR blocks. The algorithmic aspects of a runnable are modeled using the regular TargetLink blocks, and the partitioning of the algorithm into separate runnables and data exchange between them is modeled by adding TargetLink AUTOSAR blocks. Figure 4 shows the design of a simple controller for the AUTOSAR use case. Whenever a subsystem is to become an AUTOSAR runnable during code generation, the Runnable block must be dragged into it. For communication with other runnables, the Runnable Inports and Outports have to be inserted to receive and send input and output signals. This reflects the AUTOSAR principle that all communication between runnables is routed through an AUTOSAR interface, which is implemented in the form of an RTE macro during code generation. The algorithmic part of the controller, however, is modeled in the exactly same way as in the non-AUTOSAR use case.

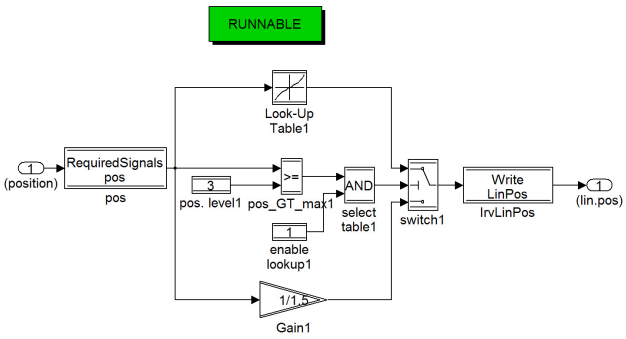

Figure 4: Designing a controller for the AUTOSAR use case by combining algorithmic blocks such as Gain, Switch, and Look-Up Table blocks with AUTOSAR blocks.

Developers are given convenient and extremely powerful modeling options for turning function models into AUTOSAR-compliant components. Since the modeling of runnables follows a proven workflow, existing models can be easily migrated to AUTOSAR. Users must primarily split the designed functionality into runnable entities in the same way as they would partition it into subsystems or functions. Moreover, users continue to work in their familiar environment, making the modeling of AUTOSAR software components particularly attractive and efficient.

5. Managing AUTOSAR-Related Data

Alongside the use of AUTOSAR blocks at model level, AUTOSAR-related information is kept in a data dictionary. This serves as a central data container and is used not only for AUTOSAR specifications but also for code generation information in general, such as data types, scalings and variables. These specifications define how individual signals appear in the generated code. The AUTOSAR-related information stored in the data dictionary are software components, runnables, ports, interfaces, etc. All these elements require a set of associated attributes which are defined by the AUTOSAR standard. Interfaces, for instance, must specify the data types of their enclosed data elements, while runnables must define the events by which they are activated.
To connect the AUTOSAR blocks on the model level with the AUTOSAR specifications in the data dictionary, the latter are simply referenced from the AUTOSAR blocks. Thereby, the appropriate AUTOSAR specifications are assigned to the individual signals in the model, see Figure 5, and the proper RTE macros are generated during code generation later on.

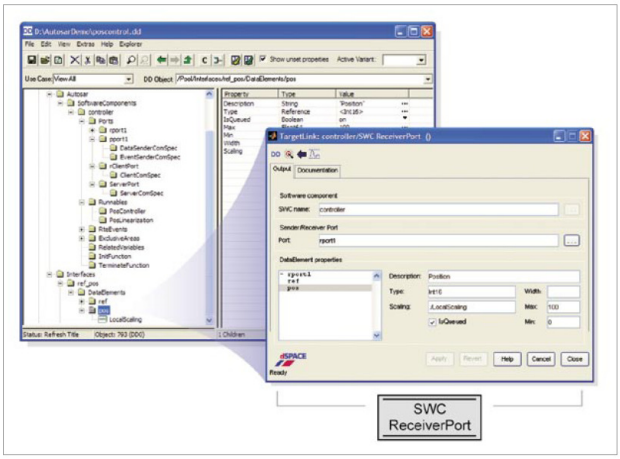

Figure 5: Specifying AUTOSAR properties in the data dictionary and referencing them in AUTOSAR block dialogs.

6. Production Code Generation

An essential advantage of model-based development is that the model serves as a reference, and executable code can be generated from it at any time, practically with just a click. This obviously also applies to AUTOSAR software components modeled with TargetLink. After all specifications have been made at block level or in the dSPACE Data Dictionary, actually generating

AUTOSAR-compliant code is a matter of only a few clicks.
The generated code is characterized, according to AUTOSAR requirements, in that all accesses to I/O, basic services, and communication with other components are implemented as RTE macros in the following way:

- Runnables become functions in the generated code.

- Incoming or outgoing signals - i.e., Runnable Inports and Outports - appear in the code as standardized RTE macros. Instead of referencing connected software components directly, which would make it impossible to reuse each component, these macros reference the components' own ports. In this sense ports are proxies for the components that will be connected to them later on. Figure 6 shows a small example: The "`linearization_runnable`" runnable reads one variable using the RTE macro **`Rte_Receive_RequiredSignals_pos`** and writes the result using the macro **`Rte_IrvIWrite_PosLinearization_LinPos.`** The definitions of these macros are generated later on during the RTE generation phase once software components have been mapped to individual ECUs. AUTOSAR standardizes the syntax of these macros.

- Communication between the runnables of one software component is handled specially using so-called interrunnable variables, as in the macro **`Rte_IrvIWrite_PosLinearization_LinPos`**.

```
void linearization_runnable(void)
{
   /* storage class for local variables */
   SInt16 S13_switch1 /* LSB: 2^-10 OFF: 0 */;
   S16_pos pos /* LSB: 2^-10 OFF:  0 */;

   /* call of
      Required signal pos */
Rte_Receive_RequiredSignals_pos(&(pos));

   ......
   ......
   ......

   /* # combined # update(s) for inport
   controller/Position_Linearization_Runnable */

Rte_IrvIWrite_PosLinearization_LinPos(S13_switch
   1);
}
```

Figure 6: Example code fragment for the simple controller in Figure 4.

7. Generating Software Component Descriptions

As indicated in Figure 2, an AUTOSAR software component consists not just of the C code itself but also a software component description. This is standardized by AUTOSAR and constitutes an XML file which contains a precise characterization of the structural elements contained in a component, such as runnable entities, ports, and communication interfaces. The component description provides the basis for a software architecture tool to read off the relevant properties of each software component, which enables the software architect to link the individual components in an AUTOSAR tool chain.

When model-based design and automatic production code generation are used, it is obviously not necessary to create and edit the software component description manually. Instead, all the relevant information is stored on the model level or in the data dictionary, and the description can be generated automatically at any time, for instance along with the AUTOSAR-compliant code. This ensures consistency between the model, code, and software component description. TargetLink supports the generation of AUTOSAR software component descriptions at a click, with the required information being retrieved from the dSPACE Data Dictionary, see Figure 7.

```
<!-- Definition of the component Windowlifter -->
<ATOMIC-SOFTWARE-COMPONENT-TYPE>
<SHORT-NAME>WINDOW_LIFTER</SHORT-NAME>
<!-- Port definition -->
<PORT-PROTOTYPES>
        <P-PORT-PROTOTYPE>
                <SHORT-NAME>P_MOVE</SHORT-NAME>
                <CLIENT-SERVER-INTERFACE-TREF>
                        /autosar/IF_WL_MOVE
```

Figure 7: Excerpt from a software component description file, which can easily be generated along with the AUTOSAR compliant-code.

8. Simulating Software Components

One of the great advantages of model-based design has always been the ability to simulate functional behavior at very early stages of the development process. This paves the way for immediate validation and verification techniques and hence greatly reduces the risk involved in the development process. These advantages also hold true for the modeling of AUTOSAR SWCs.

For the purpose of simulation, the modeled AUTOSAR software components are included in a special simulation frame realized by a TargetLink subsystem, see Figure 8. This makes sure that software components can not only be simulated on the block diagram level, usually called model-in-the-loop mode (MIL), but also in software-in-the-loop mode (SIL), in which the generated code is executed for the controller part of the model.

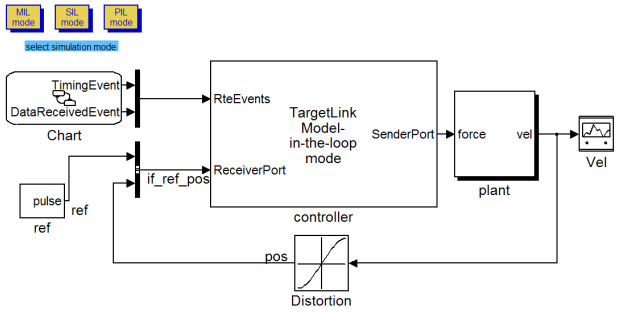

Figure 8: Modeling style to properly simulate AUTOSAR software components. Runnables are included in the TargetLink *controller* subsystem and activated by events emitted by the Stateflow chart called *Chart*.

The AUTOSAR standard specifies that runnables are activated by so-called RTE events. Simulink provides a comparable concept in the form of function-call-triggered subsystems, whose execution is actuated by events from other blocks. Consequently, runnables are implemented as function-call-triggered subsystems, see Figure 9. It is a common practice to activate function-call-triggered subsystems, i.e. the runnables, by events from a Stateflow chart residing outside the software component, see Figure 8. The latter is therefore used only for simulation purposes and has no bearing on subsequent code generation steps or the software component itself, see [4]. If a runnable is time-triggered, it can also be implemented as an ordinary Simulink subsystem with the proper sample rate.

For software-in-the-loop simulation, code for the AUTOSAR SWC is generated and wrapped in a Simulink S-function, a concept for including user code in a Simulink simulation. To build the S-function, wrapper code resembling a simple Run-Time Environment has to be generated along with the actual component's code. During software-in-the-loop simulation, the blocks inside the TargetLink subsystem in Figure 8 are replaced by the generated S-function. The same concept can be used to perform what is known as processor-in-the-loop simulation (PIL), in which the code is executed on an evaluation board containing the target processor.

However, the capacity to realistically simulate AUTOSAR SWCs in Simulink, be it in MIL, SIL or PIL mode, is limited. Simulink simulations can serve to test the functional behavior, but they are not sufficient for proper simulation of real-time behavior, which arises, for instance, from later mapping of runnables to the tasks of an operating system.

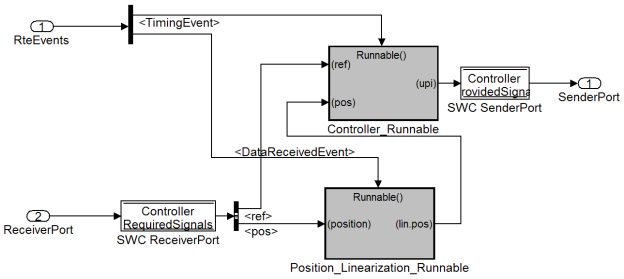

Figure 9: Runnables are implemented as function-call-triggered subsystems and activated by function calls emulating AUTOSAR RTE events.

9. Integration in an AUTOSAR Tool Chain

Production projects in the automotive industry nowadays often take individual software components as the starting point for development. After an AUTOSAR software component has been successfully developed, for instance with TargetLink, it has to be integrated in an overall AUTOSAR application. This procedure is of course supported by the AUTOSAR methodology and system design tools such as SystemDesk serve that purpose, see [5].
A software integrator uses SystemDesk to integrate individual SWCs provided by function developers to make a system. The AUTOSAR SWC descriptions are imported into the tool, providing all the necessary information about the ports and interfaces of the required software components. Using model-based design on the system level, the integrator now simply connects individual components to model communication between AUTOSAR SWCs and the basic software, see Figure 10. Tool support is applied to check compatibility between the interfaces of software components. If the interfaces are not compatible, for example because different fixed-point scalings were used, the function developers have to adjust the interfaces.

Rather than starting from individual function models (bottom-up approach), it is to be expected that an architecture-driven procedure (top-down approach) will be pursued in the future. A software architect first designs the whole architecture in a system design tool and specifies the connections between SWCs and their formal interfaces at an early stage in the development process. Individual developers then

transform functional models into AUTOSAR SWCs based on the specified software architecture. The advantages of this approach are that compatible interfaces are created systematically and that all the signals required are provided by other SWCs. In addition, each function developer can use the software component description to generate a frame model as a starting point to migrate existing function models to AUTOSAR more easily.

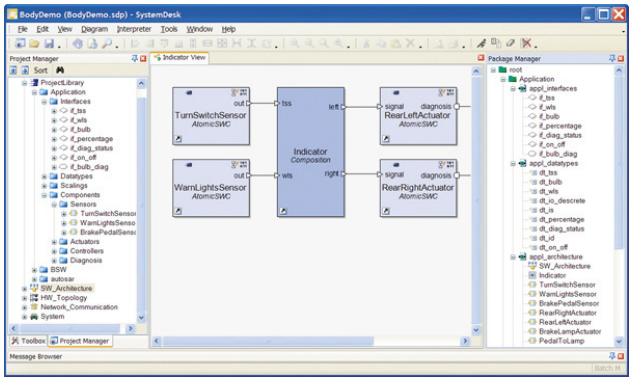

Figure 10: Using model-based design techniques for software architecture design.

To build the overall AUTOSAR application, it is of course necessary to perform an RTE generation step. As mentioned before, AUTOSAR-compatible application software must not use any direct calls to basic software layers such as I/O drivers, to other software components, or to the communication stack. Instead, standardized RTE macros in the components' code are used for that purpose. It is now up to the RTE generator to provide implementations for these macros according to the configuration of the overall system.
Once the software components have been properly connected by the software architect/integrator, individual software components have been mapped to ECUs and runnables have been mapped to tasks, an RTE generator has the necessary information to provide the implementation of the RTE macros. Hence, platform-independent communication patterns used in software component code are implemented with the proper communication means such as CAN messages, inter-task messages, and global variables. The RTE generator also generates the required tasks for the real-time application.
A system design tool like SystemDesk also supports OS and COM configurations as well as exporting this information to other tools like EB tresos, see [6], which then serve to generate the OS and the COM stack to build the final AUTOSAR application. The whole tool interaction is summarized in Figure 11.

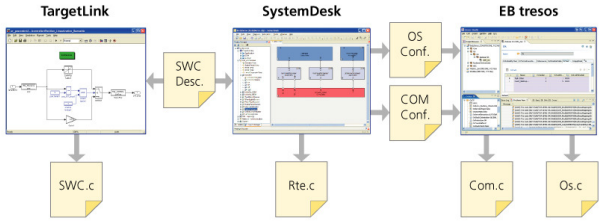

Figure 11: Interaction between functional behavior modeling, RTE generation, and OS and COM stack generation.

10. Conclusion

The AUTOSAR initiative is likely to achieve a substantial increase in the efficiency and quality of automotive software development if the development process is supported by proper tools. As regards the actual application software, model-based development is the most obvious approach. The paper therefore shows how a tool chain based on Simulink/Stateflow and TargetLink can be adapted to model AUTOSAR software components on the basis of block and state diagrams. This enables users to design function models in their familiar development environment, with practically the same workflow and with intensive tool support. Most importantly, the software components can be directly implemented in the form of highly efficient AUTOSAR-compliant production code ready to be integrated on an AUTOSAR ECU.

The paper also highlights the importance of seamless integration of model-based design techniques in an AUTOSAR tool chain, especially their interaction with architecture tools and RTE generators. As a whole, the development of AUTOSAR software components benefits greatly from model-based design techniques, particularly since the success of AUTOSAR is heavily dependent on efficient tool support.

11. References

[1] M. Beine, U. Eisemann, R. Otterbach. Transforming a Control Design Model into an Efficient Production Application, CACSD Conference, October 2006

[2] M. Beine, R. Otterbach and M. Jungmann. Development of safety-critical software using automatic code generation. SAE World Congress, Detroit, 2004

[3] AUTOSAR. Internet Homepage http://www.autosar.org

[4] AUTOSAR. Applying Simulink to AUTOSAR, http://www.autosar.org/download/AUTOSAR_SimulinkStyleguide.pdf

[5] D. Stichling, O. Niggemann, J. Stroop, R. Otterbach. From Function Design to System Design in Model-Based Software Development, ATZ, No. 1, 2007

[6] EB Internet Homepage. http://www.elektrobit.com/static/en/eb_tresos.html

12. Glossary

AUTOSAR *AUTomotive Open System ARchitecture*

RTE Run-Time Environment

SWC Software Component