The AUTOSAR standard – the experience of applying Simulink according to its requirements

Anila Mjeda, Gabriel Leen
Lero – The Irish Software Engineering Research Centre, University of Limerick

Eamonn Walsh
John Holland Research Center, University of Limerick

ABSTRACT

Software has become the driving force behind innovation in the automotive industry. According to a recent Frost & Sullivan market research report, “Strategic Analysis of the European Market for Software in Passenger Cars”, the software for passenger cars in Europe accounted for 5.8% of the cost of vehicles in 2004. The same report projects an annual growth rate (CAGR) of 15.5% until 2011, when software is expected to represent 11.0% of the cost of vehicles.

This paper discusses the emerging AUTOSAR standard (AUTOSAR release 2.0), which aims to address the need for more efficient E/E (Embedded Electronic) automotive architecture solutions. The paper also provides a review of the mapping of a few key AUTOSAR concepts to their Simulink counterparts, which should help designers using Simulink deliver AUTOSAR compatible solutions.

Finally the paper features a component of an experimental engine management system that has been remodeled in Simulink to conform to the AUTOSAR requirements.

INTRODUCTION

The automotive industry has a history of each supplier developing its own proprietary automotive software and of high confidentiality surrounding this development. Currently there is growing awareness of the need for customization and common solutions.

AUTOSAR (Automotive Open System Architecture) is a partnership of major players in the automotive industry formed with the objective to establish an open standard for automotive E/E architecture.

According to the AUTOSAR official website (www.autosar.org) its vision is “…to improve complexity management of integrated E/E architectures through increased reuse and exchangeability of SW modules”.

AUTOSAR aims to achieve modularity, scalability, transferability and re-usability of functions in automotive software.

Heinecke [1], reports that the main focus for AUTOSAR is to provide a layered architecture (see Figure 1), which allows for abstraction from hardware on the lower interface and abstraction to application layer software on the upper interface. This is complemented with the provision of specifications for “compatible functional interfaces and the design of a coherent methodology based on standardized templates and data exchange formats”.

![AUTOSAR Architecture Overview](www.autosar.org)

AUTOSAR supports the C, C++ and Java programming languages and identifies the driving forces behind the standardization as follows [2]:

- “Manage the increasing E/E complexity associated with the growth of functional scope
• Improve flexibility for product modification, upgrade and update
• Improve the scalability of solutions within and across product lines
• Improve the quality and reliability of E/E systems
• Enable the detection of errors in early design phases."

AUTOSAR PROJECT OBJECTIVES

AUTOSAR aims to achieve a number of project objectives (PO-s). The following table provides the full list of these objectives as documented in AUTOSAR Release 2.0.

<table>
<thead>
<tr>
<th>No</th>
<th>Project Objectives</th>
<th>Abbreviation</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Consideration of availability and safety requirements</td>
<td>AUTOSAR PO1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Redundancy activation</td>
<td>AUTOSAR PO2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Scalability to different vehicle and platform variants</td>
<td>AUTOSAR PO3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Implementation and standardization of basic system functions as an OEM-wide Standard Core solution</td>
<td>AUTOSAR PO4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Transferability of functions throughout the network</td>
<td>AUTOSAR PO5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>Integration of functional modules from multiple suppliers</td>
<td>AUTOSAR PO6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>Maintainability throughout the whole product life cycle</td>
<td>AUTOSAR PO7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>Increased use of commercial off the shelf hardware (COTS)</td>
<td>AUTOSAR PO8</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>Software updates and upgrades over vehicle lifetime</td>
<td>AUTOSAR PO9</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 1 AUTOSAR Project Objectives [3]

where the priority levels have the following meaning:
• 1 = High, Required
• 2 = Medium, recommended
• 3 = Low, optional

LAYERED ARCHITECTURE

AUTOSAR features a layered architecture with standardized and fully described interfaces. An overview of its architecture is shown in Figure 1.

The level of abstraction enforced by its standard interfaces and fully specified communication mechanisms allows the applications to be hardware-independent. The architecture choice also introduces the concept of AUTOSAR Runtime Environment (RTE), which caters to the transferability of functions and redundancy activation.

Virtual Functional Bus (VFB)

According to the AUTOSAR specification [2], “the Virtual Functional Bus (VFB) is the abstraction of the AUTOSAR software components interconnections to the entire vehicle.”

The VFB can be thought of as a middleware layer technology that allows the communications between software components and between software components and the rest of the environment to be technology-independent. A schematic view of AUTOSAR components and services connected to the VFB is depicted in Figure 2.

![Figure 2 Representations of Atomic Software Components and AUTOSAR Services connected to the VFB][2]

In the above illustration, the “Complex Device Drivers”, “ECU Abstraction” and “Services” form what is called the “Basic Software”.

The AUTOSAR RTE is the run-time implementation of the Virtual Functional Bus (VFB) on a specific ECU (Electronic Control Unit) [2] and provides the same interface and services to the Software Components attached to it for either inter-ECU (via CAN, LIN, Flexray or MOST) or intra-ECU communication. In other words, this makes the software components independent from the communication mechanisms and channels.

The RTE is a novel middleware layer technology that abstracts the Application Layer from all the implementation details of the basic software and hardware, enabling Application Layer software components to be transferable across the network.

AUTOSAR Software

The AUTOSAR Software is the layer that sits above the AUTOSAR Runtime Environment and is composed of AUTOSAR software components, which in turn are mapped on the ECU.

The AUTOSAR Software Component

A fundamental design concept of AUTOSAR is the separation between application and infrastructure. Consequently, the AUTOSAR software component here is designed to be independent from infrastructure and interacts with other components through well defined ports. Figure 3 illustrates three AUTOSAR software components interconnected to each other.

Here, it is important to mention that an AUTOSAR software component is an “Atomic Software Component”, i.e. it is designed to be deployed in only one ECU.
All interaction between the AUTOSAR software components is done via the AUTOSAR RTE, and the AUTOSAR interface assures their connectivity.

According to the specification, the AUTOSAR software component should be described in detail, as indeed should all AUTOSAR components. The description for the AUTOSAR software component should include:

- the operations and data elements that the software component provides and requires
- the requirements that a software component has on the infrastructure
- the resources needed by the software component
- information regarding the specific implementation of the software component

As shown in Figure 4, the specification recommends that the description of a software component in AUTOSAR should be done in three levels: Virtual Functional Bus Level, Run-time-Environment level and Implementation level.

AUTOSAR Basic Software

The AUTOSAR Basic Software is located below the AUTOSAR Runtime Environment. It contains standardized and ECU-specific components, and while it does not carry out any job itself, it provides standardized services to the AUTOSAR Software Components.

AUTOSAR METHODOLOGY

The AUTOSAR development process starts with formal descriptions of the interfaces of software and hardware components. Templates for the description of these interfaces conforming to the AUTOSAR standard are given, and the former are used as an input throughout the development process.

The common technical approach for the steps taken in developing the system is called the “AUTOSAR methodology”.

According to the AUTOSAR specification:

“The AUTOSAR Methodology is neither a complete process description nor a business model and ‘roles’ and ‘responsibilities’ are not defined in this methodology. Furthermore it does not prescribe a precise order in which activities should be carried out. The methodology is a mere work-product flow: it defines the dependencies of activities on work-products.”

AUTOSAR describes the methodology using the Software Process Engineering meta-model (SPEM). SPEM is a standard developed by the Object Management Group (OMG), which is designed to describe software development processes. SPEM includes the description of the appropriate graphical notations for a range of concepts such as Work Product, Activity, Guidance, Work-Flow Products, Dependency, etc.

Figure 5 provides an overview of the AUTOSAR methodology. It follows SPEM graphical notation to depict the design steps from the system configuration to the generation of an ECU executable.

AUTOSAR supports the formal description required in each step by requesting the use of an information exchange format and the use of templates. According to AUTOSAR: “A Template is a structured collection of attributes that are required to formally describe AUTOSAR artifacts like software components or configurations of ECUs”[5].

The templates in AUTOSAR are in the form of simplified UML class diagrams1, and the resulting UML model is called the template model. Figure 6 shows the full AUTOSAR metamodel hierarchy.

The AUTOSAR methodology promotes an organic integration of tool support and strongly supports model-based design techniques.

---

1 AUTOSAR uses an UML profile described in the AUTOSAR specification (in the Template UML Profile and Modelling Guide [6]).
MAPPING OF AUTOSAR SOFTWARE COMPONENTS TO SIMULINK

Simulink is a model-based design environment that provides an interactive graphical environment and a customizable set of block libraries that allows users to accurately design, simulate, implement, and test dynamic systems.

Simulink is widely used in industry [7]. It allows the user to model and simulate a wide range of linear, nonlinear, continuous, discrete and hybrid dynamic systems. It has the capability to model just about anything that can be modeled mathematically.

Atomic Software Component

Atomic in the AUTOSAR context denotes a software component that has to be deployed in only one ECU, and it must not be confused with the concept of the atomic execution of Simulink subsystems. An AUTOSAR atomic software component can be represented in Simulink by a subsystem or a model.

The AUTOSAR specification gives the following canonical pattern for modeling atomic software components in Simulink (see Figure 7).

Figure 7 Canonical Pattern for Software Components [8]

Here it is important to highlight the fact that the first import in the Simulink block used to model an AUTOSAR atomic software component should be used to route RTE events (RTEEvents) to the software component’s runnables.
AUTOSAR Ports

AUTOSAR uses specific ports requiring or providing data or a service from/to a server, named respectively Require-Ports (R-Ports) and Provide-Ports (P-Ports). An AUTOSAR R-Port is modeled in Simulink by an in-port for sender/receiver communication, while the P-Port is modeled by an out-port for sender/receiver communication.

ComSpec

The communication specification (ComSpec) classes in AUTOSAR capture the information regarding the quality of data communication. Figure 8 and Figure 9 show a summary of how to capture information on the quality of communication in Simulink.

Sender/Receiver Interface

A SenderReceiverInterface is an AUTOSAR port interface used for sender/receiver communication, and it is recommended that it be modeled in Simulink via a Simulink BusObject, as shown in Figure 10.

Here it is important to emphasize that this interface only describes the structure of the data elements required or provided by these ports, and does not contain information about the quality of the communication (this information as described above is to be found in the ComSpec classes).

Client Server Interface

This interface is used for client/server communication. The AUTOSAR specification to date does not cover client/server communication among atomic software components, but it does tackle client/server communication between atomic software components (in the role of the client) and AUTOSAR services, as well as sensor actuator software components (as client) and the ECU abstraction. These ports can be modeled in Simulink as a masked s-function or a Stateflow chart.

Sender Receiver Annotation

SenderReceiverAnnotations are basically comments on the data elements in a sender/receiver port and are modeled as such in Simulink; hence they do not influence the code generation, but serve as information to help with the design of the system.

Sensor Actuator Software Components

The sensor and actuator software components offer real-world signals to the application software components, and they are the only components in that level of the AUTOSAR architecture that can access ports from the ECU abstraction below the RTE layer. They are modeled in Simulink as virtual subsystems that contain both “normal” ports and ECU abstraction ports.

Services

AUTOSAR services are functions provided by the RTE, and they can be modeled in Simulink as virtual subsystems.

Runnable

Runnables are the smallest code fragments inside an atomic software component that can be activated by the RTE using RTEEvents independently from the other parts (runnables) of the atomic software component. A
Runnable is modeled in Simulink as a function call subsystem.

**Figure 11 Canonical Pattern for Runnables.**

Runnable Simple is activated by multiple RTEEvents [8].

**RTEEvents**

RTEEvents are used by the RTE to trigger the execution of runnables. In Simulink they are modeled as function calls.

**Exclusive Areas**

Exclusive areas are regions of code within which RTE blocks concurrent data access. In Simulink they are represented as atomic subsystems marked as an exclusive area. In the event when an entire runnable runs inside an exclusive area, the top-level runnable subsystem should be marked as an exclusive area in Simulink.

The Simulink representation of a runnable only containing parts that belong in an exclusive area is done via modeling nested subsystems (within the runnable) which are marked as exclusive areas.

**Composition**

Compositions are used in AUTOSAR to create higher abstraction levels. They encapsulate collaboration of components, and are represented in Simulink by virtual subsystems.

**Datatypes**

The primitive data types used by AUTOSAR are mapped in Simulink as shown in Table 2.

AUTOSAR also uses two types of composite datatypes, arrays and records. Arrays in AUTOSAR have zero indexing, and are represented in Simulink by wide signals with individual elements concatenated together using mux blocks and separated using demux or selector blocks. Records can be represented by bus objects or as Simulink StrucType objects.

**Table 2 Autosar and Simulink's built in datatypes [8]**

<table>
<thead>
<tr>
<th>AUTOSAR Type</th>
<th>BSW Type</th>
<th>Simulink Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>UInt4</td>
<td>uint8</td>
<td>uint8</td>
</tr>
<tr>
<td>SInt4</td>
<td>sint8</td>
<td>int8</td>
</tr>
<tr>
<td>UInt8</td>
<td>uint32</td>
<td>uint32</td>
</tr>
<tr>
<td>SInt8</td>
<td>sint32</td>
<td>int32</td>
</tr>
<tr>
<td>UInt16</td>
<td>uint16</td>
<td>uint16</td>
</tr>
<tr>
<td>SInt16</td>
<td>sint16</td>
<td>int16</td>
</tr>
<tr>
<td>UInt32</td>
<td>uint32</td>
<td>uint32</td>
</tr>
<tr>
<td>SInt32</td>
<td>sint32</td>
<td>int32</td>
</tr>
<tr>
<td>Float_withNaN</td>
<td>float32</td>
<td>float</td>
</tr>
<tr>
<td>Float</td>
<td>float32</td>
<td>float</td>
</tr>
<tr>
<td>Double_withNaN</td>
<td>float64</td>
<td>double</td>
</tr>
<tr>
<td>Double</td>
<td>float64</td>
<td>double</td>
</tr>
<tr>
<td>Boolean</td>
<td>boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>Char8</td>
<td>uint8</td>
<td>uint8</td>
</tr>
<tr>
<td>Char16</td>
<td>uint16</td>
<td>uint16</td>
</tr>
</tbody>
</table>

**Characteristics**

Characteristics are values that can be changed in an ECU, and they can be modeled by extending the Simulink.Parameter class to a new class: AUTOSAR.Characteristic.

**APPLYING SIMULINK FOR AUTOSAR IN PRACTICE**

A pre-existing Simulink system was redesigned according to the AUTOSAR methodology. The system in question controls an experimental engine management system for use in the automotive industry. The example in Figure 12 is a modification of the PID controller segment of the Simulink system. The modifications were made so that the operation of the PID controller could be tested without the need of an operational plant. The plant featured in the example is a rheostat. A voltage is placed across the rheostat and the desired output (the Setpoint) is selected by the user as the input to the system. When the PID controller is started it actuates an external motor controller, which moves the rheostat wiper position, thus altering the rheostat output voltage, i.e. the output of the system. The “Speed Signal Conditioning” block keeps the value of the output motor speed signal within a preset safe range. The “Position Limiting” block ensures that the rheostat is not damaged during model execution.

**Figure 12 High level view of the PID test system**
This System was modelled in Simulink according to the AUTOSAR methodology and the model in Figure 13 is the result.

![Figure 13 PID Test System in Simulink](image)

This model takes the discrete blocks shown in Figure 12 and recreates them as AUTOSAR runnables in the form of Simulink function-call subsystems. All communication between the runnables takes place via the RTE. Each runnable is called as necessary and the final output of the system is routed to the External Motor Controller Plant.

The process of redesigning the Simulink model to conform to the AUTOSAR standard was found to be relatively straightforward once the AUTOSAR requirements were understood.

**CONCLUSION**

The emerging AUTOSAR standard presents a very interesting development for the automotive software field. It presents an innovative, layered architecture composed of modular components with standardized interfaces. Furthermore, AUTOSAR maximizes hardware independence for in-vehicle software.

The AUTOSAR methodology promotes a thorough integration of tool support and model-based design techniques.

A model-based design environment like Simulink is thoroughly supported by AUTOSAR, which describes in its specification the mapping of key AUTOSAR concepts into Simulink. This greatly helps system designers to model AUTOSAR conformant systems in Simulink.

**ACKNOWLEDGMENTS**

We wish to thank AUTOSAR for allowing the authors to reproduce illustrations from the AUTOSAR specification. Additionally, the authors would like to acknowledge the contribution and support of: Lero – The Irish Software Engineering Research Centre; the John Holland Research Centre; the Circuits and Systems Research Centre. The authors wish to thank the SFI – Science Foundation Ireland for funding this research.

**REFERENCES**


**CONTACT**

Anila Mjeda: Anila.Mjeda@lero.ie