

Model-Based Development of AUTOSAR-Compliant Applications: Exterior Lights Module Case Study

Devendra Rai

Delphi Automotive Systems Deutschland GmbH

T. K. Jestin

Delphi Automotive Systems

Lev Vitkin

Delphi Electronics and Safety

Copyright © 2008 SAE International

ABSTRACT

The complexity of automotive software and the needs for shorter development time and software portability require the development of new approaches and standards for software architectures. The AUTOSAR project is one of the most comprehensive and promising solutions for defining a methodology supporting a function-driven development process. Furthermore, it manifests itself as a standard for expressing compatible software interfaces at the Application Layer. This paper discusses the implementation of AUTOSAR requirements for the component structure, as well as interfaces to the Application Layer in a model-based development environment. The paper outlines the major AUTOSAR requirements for software components, provides examples of their implementation in a Simulink/Stateflow model, and describes the model-based implementation of an operating system for running AUTOSAR software executables ("runnables"). In addition, it describes the use of xml files for the architectural and interface descriptions of software. The architectural development and simulation of the model, along with auto-generation of AUTOSAR-compliant software, are demonstrated based on the Exterior Light Module of an automotive body controller.

INTRODUCTION

Model-based development involves the creation of a graphical executable model representing the requirements and/or design, validation of the model, automatic code generation from the model, and model-based verification of the software. The process makes use of modern modeling tools, like MATLAB/Simulink/Stateflow from The Mathworks, Inc. and code generation tools like Real-Time Workshop Embedded Coder from The Mathworks, Inc. or

TargetLink from dSPACE, Inc. Model-based development is a proven way of increasing the robustness of the design and the quality of software. At the same time, the model-based approach leads to a reduction of project development time, which is a major requirement in the highly competitive automotive market [1, 2].

Standardization is essential to handle the complexity of software and enhance modularity, flexibility, portability, reusability, maintainability etc. Standardization can be achieved via a set of procedures and guidelines, as well as via well-defined architectural and structural requirements for software. The guidelines for modeling and code generation, for example, developed by the MathWorks Automotive Advisory Board for Matlab-based models, is one of the ways to achieve such standardization [7]. Architectural requirements specified by AUTOSAR aim to achieve the objectives of modularity, flexibility, transferability, and re-usability of functions [13]. In order to comply with AUTOSAR requirements, the model should be constructed in such a way that the component configuration, interface definition and interface abstraction would allow easier implementation of AUTOSAR-compliant software.

Two major autocode generation tools supporting MATLAB/Simulink/Stateflow models, TargetLink from dSPACE, Inc, and Real-Time Workshop Embedded Coder from The MathWorks, Inc, have recently introduced the blocksets that assist in generating the AUTOSAR-compliant interfaces to the abstraction layer for software components [8, 9]. Along with technical merit, business merit influences the decision of whether to buy the available tools' add-ons vs. making these add-ons. This paper describes the alternative for generation of AUTOSAR-compliant software interfaces without the dSPACE AUTOSAR blockset.

or the AUTOSAR Demonstration Kit (ADK) for Real-Time Workshop Embedded Coder.

OVERVIEW OF AUTOSAR CONCEPTS

The main objective of AUTOSAR is to develop encapsulated application software that is abstracted from the hardware and is independent of the communication technology, operating system etc [3]. Such a software component can be relocated to different ECUs, and is reusable across different vehicle manufacturers and suppliers. The high-level AUTOSAR concept is displayed in Figure 1.

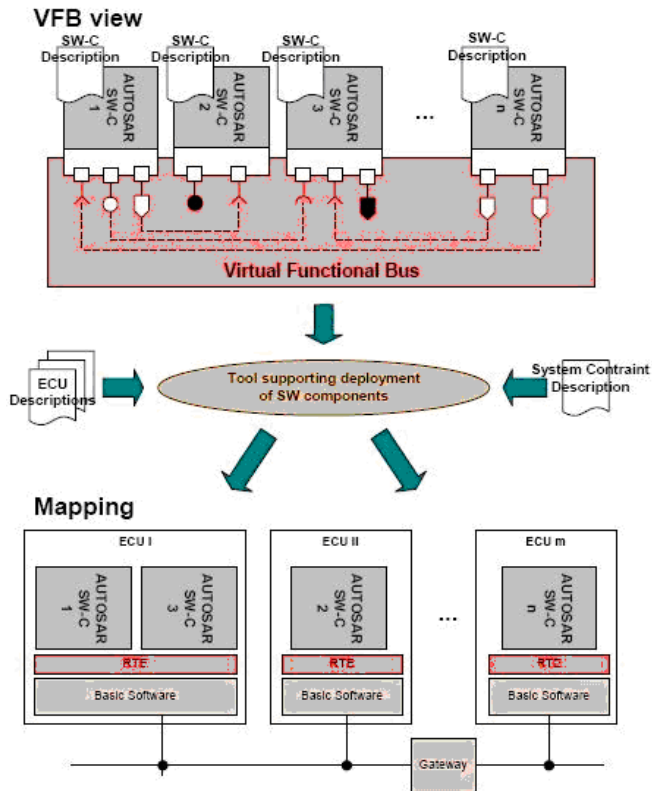


Figure 1: The AUTOSAR Concept

AUTOSAR SOFTWARE COMPONENT (SWC) – An encapsulated software component leads to separation of application and the infrastructure. Each software component is self contained as to the functional features, with well-defined interfaces to interact with other software components. AUTOSAR specifies a standard description format, i.e. “Software Component Description” to describe the component interface and the requirements on infrastructure and computing resources [3].

RUN-TIME ENVIRONMENT (RTE) – AUTOSAR specifies the communication between those software components which are residing on a technology independent abstraction layer. Implementation of a communication abstraction layer within an ECU is called a Run-time Environment [4, 5]. In a networked environment consisting of several ECUs; the corresponding RTEs jointly create an

abstraction of the communication network. In AUTOSAR terminology it is called a Virtual Functional Bus (VFB) [6]. Collaboration among RTEs on several ECUs creates standardized interfaces and services for Intra- or Inter-ECU communication, independent of a technology like CAN, LIN, Flexray etc.

AUTOSAR BASIC SOFTWARE (BSW) – The application software components run over an infrastructure software that hides the specific details of the microprocessor, ECU hardware, complex device drivers, and services. This infrastructure software is specific to the ECU, and provides a software platform for the application software [3]. The interaction of the application software component and the basic software is always done through the common APIs defined in RTE.

INTEGRATING AUTOSAR REQUIREMENTS IN MODEL BASED DEVELOPMENT

COMPONENT-BASED DESIGN – Modeling the application software starts in a framework derived from a component-based architecture, as shown in Figure 2. The component is mapped to an encapsulated subsystem in Simulink, then separated and decoupled from the hardware, communication and operating system. The component contains the “runnables”, which are modeled as atomic sub-functions. Logic for executing the sub-functions is encapsulated within the component so that the entire component is reusable. The component exposes well-defined interfaces in the form of ports, through which the component communicates to other software components. Different types of ports are available to support either Sender-Receiver (for data) or Client-Server (for services) communication as specified by AUTOSAR [3]. The component is mapped to a software module(s) containing the functions that implement the runnables and the execution logic of the runnables.

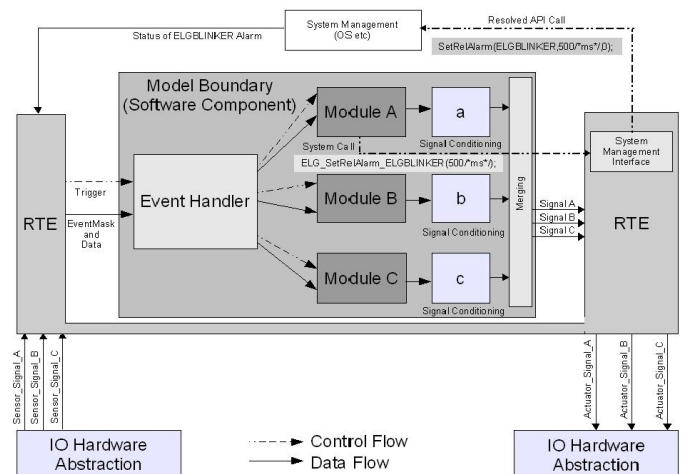


Figure 2: Architectural model of software

DATA & SERVICES – This relates to the communication mechanism as specified in the AUTOSAR technical

overview [3]. The model requires a communication layer that simulates the communication involving data and services. This contains the appropriate signal mapping, necessary signal conditioning, signal transformation functions etc. In software, this layer is mapped to the complex I/O abstraction and communication layers. That setup provides a common platform for the application layer independent of the target environment.

RUN-TIME ENVIRONMENT (RTE) – The functions within the RTE facilitate the interaction of the Application Layer with the communication and operating system. Every software component interacts with the RTE, which is responsible for data and event delivery to the module. An RTE links different software components integrating the entire application [3, 4, 5]. This also connects management services to the application from the Operating System. In the implementation, the RTE provides the access functions and macros through which an application communicates with other modules. Modeling, simulation, and implementation of the RTE are the focus area that complements the model-based development process to create AUTOSAR-compliant software. Third party tools like Electrobitt®\Tresos® create the RTE using the xml file of component description.

OPERATING SYSTEM (OS) – A scaled down model of the operating system is required to simulate the OS services like timers, alarms, etc. which is separate from the Application Layer [10]. The Application Layer uses the OS services to get the desired event driven behavior. During implementation, an OSEK-compliant operating system [12] will be used and the services will be appropriately mapped.

CASE STUDY: EXTERIOR LIGHTS MODULE

The Exterior Light Module (ELG) is an application software component that controls the exterior lights of a vehicle such as stop lights, turn blinker, head light, parking light, etc. The component receives the inputs from the brake pedal switch, turn light switch, beam control switch, gear position and other sensors. The component usually is found in the body computer ECU.

Objective

The objective is to implement the AUTOSAR concept in a model-based development environment. The project consists of the executable model for the Application Layer as well as simulatable RTE and OS. The component interface description automatically generates an RTE shell in the model, an xml-based component description file and RTE code. This automation is done via a limited set of m-scripts and VB macros developed in-house as an alternative to the commercially available AUTOSAR blockset from dSPACE.

The components of the development process of AUTOSAR-compliant software components are shown in Figure 3.

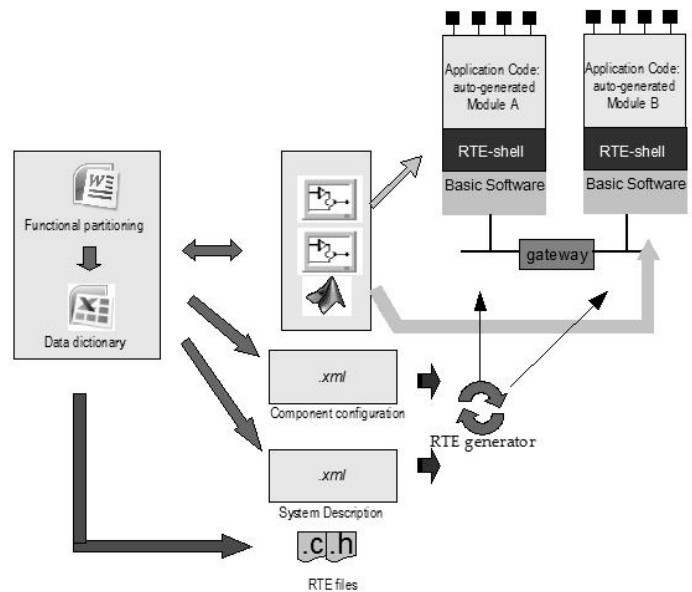


Figure 3: Components of the development process of AUTOSAR-compliant software components

Architectural description of the model

The architecture of the model is described by partitioning and by the definition of the interface.

Functional partitioning defines the functionality bounded by software components by grouping together all the related logic within the scope.

The output of this stage is the specification of the stand-alone software component. For example, all the functions related to the operation of exterior lights (turn lamps, stop lamps, headlights, etc.) are grouped into the ELG software component.

The interface definition describes the input and output interfaces for every software component. There are several attributes of the component interfaces: for example, the name, type (dataflow or control flow), systems type (range and resolution), direction (input, output) software data type, data update strategy, etc.

The accurate description of interfaces is one of the most important steps in creating AUTOSAR-compliant software. All interfaces are described in the Data Dictionary. An Excel-based Data Dictionary made in-house has been used for the ELG project. A snippet of the Data Dictionary is shown in Figure 4. The Data Dictionary contains several books, one book per SWC. In addition to interfaces, the Data Dictionary contains information about calibration (tunable parameters), constants and software configuration switches.