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

Devendra Rai

Delphi Automotive Systems Deutschland GmbH

T. K. Jestin

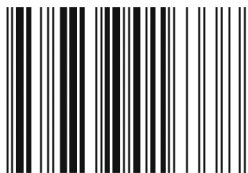
Delphi Automotive Systems

Lev Vitkin

Delphi Electronics and Safety

**Reprinted From: In-Vehicle Networks and Software, 2008
(SP-2197)**

ISBN 978-0-7680-1633-8



9 780768 016338

SAE *International*[™]

**2008 World Congress
Detroit, Michigan
April 14-17, 2008**

By mandate of the Engineering Meetings Board, this paper has been approved for SAE publication upon completion of a peer review process by a minimum of three (3) industry experts under the supervision of the session organizer.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of SAE.

For permission and licensing requests contact:

SAE Permissions
400 Commonwealth Drive
Warrendale, PA 15096-0001-USA
Email: permissions@sae.org
Tel: 724-772-4028
Fax: 724-776-3036



For multiple print copies contact:

SAE Customer Service
Tel: 877-606-7323 (inside USA and Canada)
Tel: 724-776-4970 (outside USA)
Fax: 724-776-0790
Email: CustomerService@sae.org

ISSN 0148-7191

Copyright © 2008 SAE International

Positions and opinions advanced in this paper are those of the author(s) and not necessarily those of SAE. The author is solely responsible for the content of the paper. A process is available by which discussions will be printed with the paper if it is published in SAE Transactions.

Persons wishing to submit papers to be considered for presentation or publication by SAE should send the manuscript or a 300 word abstract of a proposed manuscript to: Secretary, Engineering Meetings Board, SAE.

Printed in USA

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 for TargetLink

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.

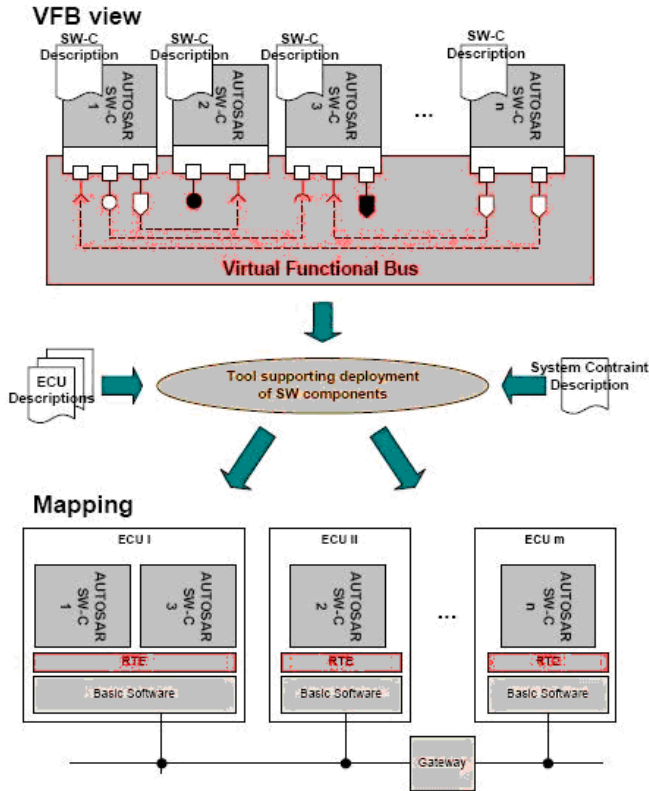


Figure 1: The AUTOSAR Concept

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.

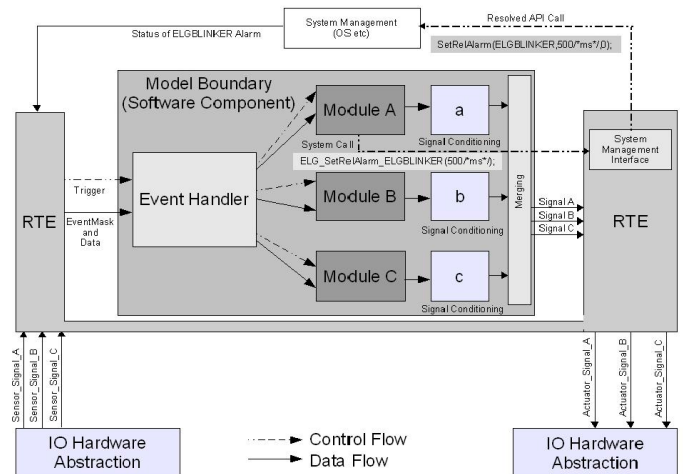


Figure 2: Architectural model of software

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 Electrobit®\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.

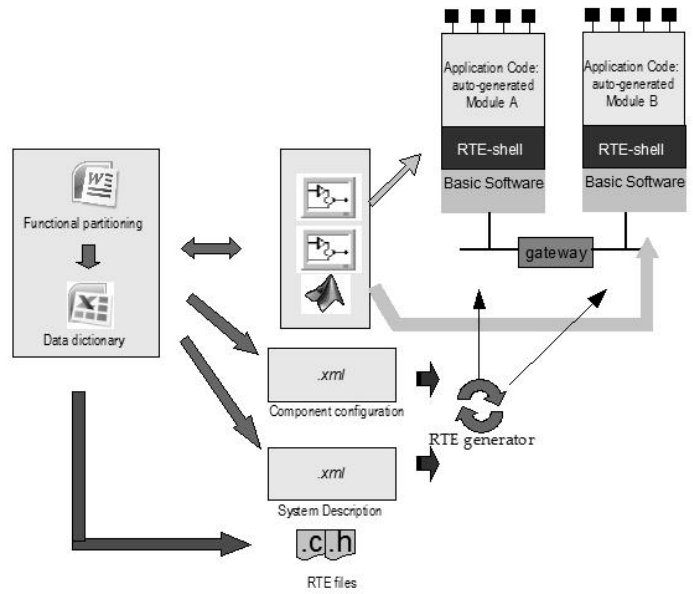


Figure 3: Components of the development process of AUTOSAR-compliant software components

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.

As an example, one ELG SWC interface has the following attributes: ELG_LIGRHFTurnLightFailSts, dataflow, output, from 0 to 8, precision 1, 8-bit, initial value = 0, updated only when the signal value is changed from last time.

DataElement	Signal Type	RTE Data Type	Array Size	Precision	Init Value	Data Update Mode	InternalEvent	Description	Conversion
ELG_LIGLHFTurnLightFailSts	Simple	uint8	None	1	0	OnDataChanged	None	Turn light left fail status indication	0 = fail not present 1 = fail present def: 0
ELG_LIGRHFTurnLightFailSts	Simple	uint8	None	1	0	OnDataChanged	None	Turn light right fail status indication	0 = fail not present 1 = fail present def: 0
ELG_LIGLHFParkTailLightFailSts	Simple	uint8	None	1	0	OnDataChanged	None	Park light left fail status indication	0 = fail not present 1 = fail present def: 0
ELG_LIGRHFParkTailLightFailSts	Simple	uint8	None	1	0	OnDataChanged	None	Park light right fail status indication	0 = fail not present 1 = fail present def: 0
ELG_LIGLHStopLightFailSts	Simple	uint8	None	1	0	OnDataChanged	None	Stop lights fail status indication (include left, right and chassis stop lights)	0 = fail not present 1 = fail present def: 0
ELG_LIGLHRRearFogLightFailSts	Simple	uint8	None	1	0	OnDataChanged	None	Rear fog light left fail status indication	0 = fail not present 1 = fail present def: 0
ELG_LIGORLFAut	Simple	uint8	None	1	0	OnDataChanged	None	Daytime running light fail status indication	0 = fail not present 1 = fail present def: 0

Figure 4: Excel-based Data Dictionary

It is worth mentioning the importance of the data update strategy. Data can be updated either time-based or event-based when the signal value is changed from the previous one. Choosing an update strategy wisely can significantly reduce software throughput.

Generation of the architectural model

This step creates a framework of the subsystems.

This process is automated by the internally-developed Component Configuration Tool. This tool is based on the Excel Data Dictionary, and consists of VB macros and m-scripts. Based on SWC and interface descriptions in the Data Dictionary, the Component Configuration Tool performs several tasks.

- Generates the top layer of the model implementing the software component. This layer consists of empty shells for all sub-components with defined and interconnected interfaces.
- Generates m-files containing the tunable parameters and constants. Populates Matlab workspace with these data.
- Defines the model configuration in the model workspace.
- Creates an RTE shell in the model for simulation.
- Creates extra logic that emulates the RTE of software: resolves events, creates a conduit for data and events to ELG and creates an event list for the built-in event handler.
- Populates the TargetLink data dictionary with relevant software data for interfaces, calibrations and constants.

- Synchronizes the Data Dictionary and a model: Ensures that whenever the model is accessed, the scripts check to see if the model interfaces are consistent with the content in the workspace.
- Generates software header files: Generates the necessary interface files for the exterior lights software component. (.c/.h files).
- Generates a component description file as per the AUTOSAR format. Necessary component description files are generated in xml format that allows development of the RTE using a suitable tool chain to satisfy AUTOSAR requirements.

As described, the Excel VB macros generate xml description files of the component as well as .c and .h files. The same products can be generated by special AUTOSAR blocksets of TargetLink or RTW Embedded Coder if the avenue “buy” vs. “make” is chosen.

Generation of a simulatable RTE –

This logic is implemented in addition to the software component ELG. The shell simulates software RTE to provide a realistic environment for model tests. As described, the RTE delivers data and event information to the model. The Component Configuration Tool generates a framework of the RTE in the model to achieve the following:

- Route signals from the IO Hardware abstraction layer (input RTE) to input ports of a software component.
- Read the incoming signals and prepare an event list which is used by the Event Handler (described in later sections) for execution control of various sub-functions. Such an event list is a set of flags packed in a bit field which form the criteria for triggering one of the several sub-functions inside the RTE. For example, a change in power mode status or change in the brake pedal switch would activate a flag associated with stoplights logic of the ELG SWC. The Event Handler reads such an event list and controls the execution of the ELG SWC.
- Re-map the signals: This was done manually. As an example, the control for low beams and parking lights are controlled by a single resistor coded switch. Hence, the IO Hardware abstraction layer provides only one coded signal for both low-beam and high-beam switch positions. The RTE splits such complex signals into simpler Boolean signals so that the model interfaces remain standard.
- Trigger software component: Based on conditions on the input signals, the RTE generates function calls to invoke the ELG SWC at appropriate times.

The RTE also reads signals from communication media like CAN and determines whether the signals have been timed-out. In such cases the RTE, in addition to supplying the data contained in such messages, also provides the validity flag associated with such signals so that the application can decide on an appropriate course of action. This data validation logic was modeled and simulated using Matlab/Simulink/Stateflow.

AUTOSAR Perspective: The input signals to the ELG model represent the interfaces from the IO-Hardware abstraction layer. These signals are just wired through the RTE to the model's ports. It should be noted that the model of RTE is just for simulation of functionality and not for code generation. In the real software RTE, signals from the IO-Hardware abstraction layer are read in by the access functions. These values are judged for correctness based on criteria like timeouts (in case of communication signals) and written to the buffers owned by the ELG SWC. This task of copying the necessary data values from the other software entities like the IO Hardware abstraction layer into the ELG's buffer variables may be done every time before the ELG is executed. This strategy, called the "copy strategy" in AUTOSAR parlance, provides the necessary data integrity [3].

For example, a port/variable named *ELG_STATUS_BCMKeySts* is used by the model to get information about the power mode of the car. The RTE copies data to this variable from the IO-Hardware abstraction layer:

```
..
if (IO_ELG_STATUS_BCMKeySts.value==ELG_INVALID)
{
ELG_STATUS_BCMKeySts = (DataType)
IO_ELG_STATUS_BCMKeySts.value;
}
else
{/*assign failsafe value*/
ELG_STATUS_BCMKeySts==ELG_SWITCH_OFF;
}
..
```

Where:

IO_ELG_STATUS_BCMKeySts is a complex signal containing data from a switch sensor and its validity state.

ELG_SWITCH_OFF and *ELG_INVALID* are defined to have suitable values.

Since the RTE cannot access variables owned by the IO-Hardware abstraction layer module, it uses access functions:

```
Rte_Read_PR_SIG_ELG_STATUS_BCMKeySts(self,
&IO_ELG_STATUS_BCMKeySts);
```

Where:

"self" is the instance of the IO component that is to be accessed.

The "System Description File" contains information about the interconnection between various SWCs. A software component interface description file generated by the Component Configuration Tool, along with the SystemDescriptionFile, is used to create software RTE for the ELG SWC.

Emulation of system services

Since the model of ELG is event-driven, the component utilizes various alarms for its functionality. A scaled down model of OSEK was developed by the authors [10] and is used as an auxiliary software component outside ELG SWC that interacts with the ELG through the RTE. Complying with AUTOSAR requirements on communication, the ELG software component does not call the OS services directly. Instead, the software component utilizes a system of APIs that are resolved in RTE to provide OS services to a software component.

For example, to set an alarm, the hazard lamp function handler uses an API as shown:

```
ELG_SetRelAlarm_ELGBLINKER(500/*ms*/);
```

This API is resolved in the RTE as:

```
SetRelAlarm(ELGBLINKER,500/*ms*/,0);
```

This example allows activation of multiple blinker functions that utilize the same software alarm.

Functional Modeling

Though AUTOSAR does not specify the internal structure of the component, there are some implied requirements targeting the modularity and reusability of the components. Reusability further implies flexibility in terms of functionality/functional configuration, data variance, and the ability to optimize to the particular configuration. All such requirements are addressed by the mechanisms of event handlers, clearly defined sub-function handlers, and signal merge areas. As a final step, it categorizes the functionality into core and auxiliary functions.

Event Handling

An event handler as shown in Figure 5., is used to read the event list as generated by the RTE to trigger the appropriate function handlers.

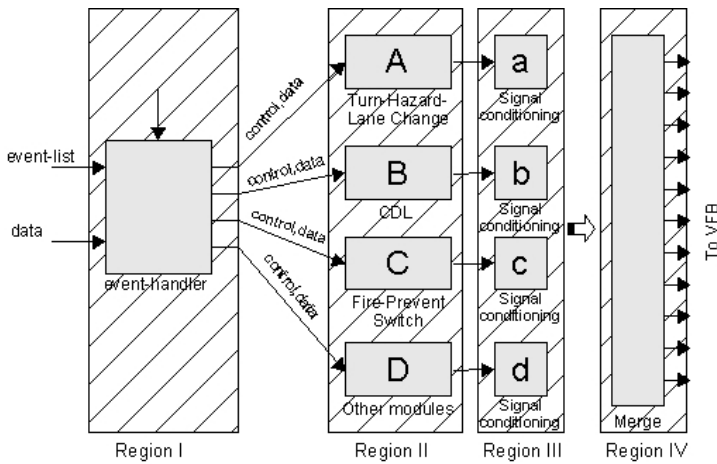


Figure 5: Event handling and sub-component triggering

This mechanism of triggering only the appropriate sub-function handler optimizes run-time performance. For example, 'Brake Press' would trigger only 'Stop Lights' logic. The RTE packs the information on events into an event list which is read by the Event Handler of the software component. In some cases, a combination of function handlers is triggered by the same event. For instance, turning on the low beams should also turn on the parking lights. The Event Handler is responsible for triggering the function handlers in the appropriate order as per the priority implicit in the Event Handler.

Sub-function handlers

The entire functionality of the software component of the ELG is further partitioned into several sub-functions. In the case of exterior lights, functions are grouped according to the relation to a particular external lamp. This scheme of partitioning ensures that software is modular, so that the components can be reused.

Core and Auxiliary functions

An increase in the level of modularity of the software component might require additional computer resources, such as memory and throughput. A highly modular architecture essentially aims at creating plug-and-play components, each of which has to provide its own buffer, OS support etc. On the other hand, high computer resource efficiency requires sharing some or all of the RAM areas (global variables), but involves considerable design, maintenance, upgrade and debugging costs. It is always a compromise between the component modularity and the efficiency of the resources. In case of exterior lights, different customers require different functionality. However, the basic functionality of the exterior light control remains the same for both cases. As shown in Figure 6, the functions TurnLamps, LaneChange, CentralLock (CDL) and Hazard are the core functions, where EmergencyStopSignaling (ESS) and FirePreventionSwitch (FPS) are add-on functions as per customer specifications.

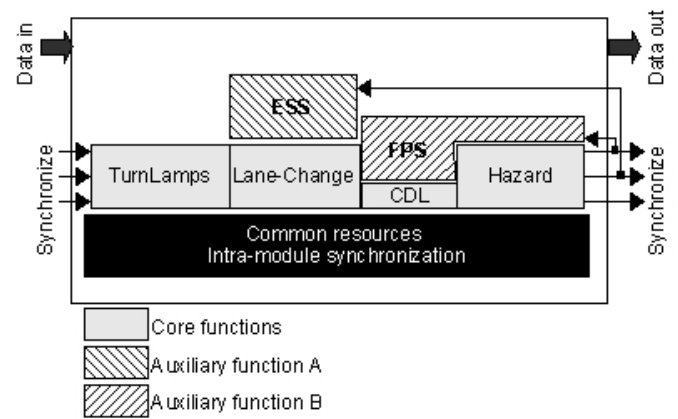


Figure 6: Core and auxiliary functions

This approach brings in the best of both efficiency and modularity. We could reduce the RAM budget by bringing down the independent interfaces and sharing OS resources inside the core function to some extent, while maintaining sufficient modularity.

The AUTOSAR Perspective

AUTOSAR handles modularity using a cluster of "runnables". Runnables are analogous to sub-functions in the ELG, although the entire ELG SWC can also be made as one runnable. For example, the 'Stop Lights' function handler inside the ELG SWC can be made into a runnable. In this case, AUTOSAR specifies that a set of conditions for execution of the runnables shall be established. Targetlink's AUTOSAR blockset enables the user to specify the list of events that will form the criteria for executing such a runnable. Our Data Dictionary can build the list of events as well. In this case, the Event Handler is not required, since the RTE directly controls the execution of each of the runnables forming the SWC. This approach to software design makes a software component like ELG a simple collection of all the runnables, the task of correctly triggering each of such runnables being performed by RTE itself. On one hand, this leads to independent, highly modular and distributed development; on the other hand, it leaves the design of the RTE more complex and prone to errors. Specifying the trigger conditions for all such runnables may be a cumbersome task, and in many cases, execution of a runnable may be dependent on other runnables. For example, the low beams function handler (low beams handler runnable) needs to be executed every time, and also just after the high beams function has been executed. In summary, this method of software design shifts some of the logic complexity from the SWC to the RTE.

Signal Merging

Partitioning the functionality into core and auxiliary functions requires a well-defined signal merging layer. For example, the core function "Hazard" attempts to write into the port ELG_RHTurnLights that modifies the

electrical state of “right turn lamps”. However, the auxiliary function FPS also controls the same port under certain conditions. Therefore, the signals coming out of these two functions can be merged into one output signal based upon priority. This merging of signals is carried out in the signal merging layer. This method of merging saves the modular nature of the model, since complex merging logic stays outside of both the FPS and “Hazard” function handlers.

Code Generation

Fill-out of the Data Dictionary is a major task of the code generation process. After the Data Dictionary is completed, the Component Configuration Tool automates the setting of attributes of the model's objects. Thereafter, just a handful set of additional attributes of the objects is needed to be assigned directly by the TargetLink editor.

Simulation of system services like the OS during the software-in-the-loop (SIL) stage requires the code of the OS to be compiled and linked to the application code. In the ELG project, the model of the OS was developed for Model-in-the-loop simulation. Auto-generated code for the OS has been used for SIL. Once the SIL tests were over, the contents of auto-generated OS files were replaced by the existing hand-made OS implementation at the integration stage.

System Configuration

Since the components are independent of the environment, the component description file does not contain the information regarding the source of the data and services. Also, it does not contain the information on which the ECU component is going to reside. Data and services may be available within the same ECU, or they may be from another ECU. The SystemDescriptionFile provides details of data and services networking. System configuration involves a complex process of optimally allocating the components in the ECUs and satisfying the resource constraints. Also the system configuration involves designing the communication matrix [3]. Using the Component Configuration Tool, the system architect configures the system and generates a SystemDescriptionFile in an xml format as specified by AUTOSAR.

CONCLUSION

A model-based development process with AUTOSAR compliance enables the development of standardized software with high quality and efficiency. Tools are evolving in the market to create AUTOSAR-compliant software. Alternatively, low-cost tools made in-house can be efficiently used for the development of AUTOSAR-compliant software components. The paper described that approach supported by a Component

Configuration Tool (a set of VB macros and m scripts) in conjunction with an Excel-based Data Dictionary. Although this approach might lack some flexibility and variety of features that commercially available tools offer (like the AUTOSAR blockset from TargetLink), it serves the low-budget projects well where the use of expensive licenses is not feasible.

ACKNOWLEDGMENTS

The paper is based on the project carried out at Delphi Automotive Systems Deutschland GmbH, Megamos, Germany. We express our heartfelt thanks to Ralf Rahbach, Delphi Megamos, for his excellent management of the project. We are indebted to Georg Zoeller, software architect at Delphi Megamos for his leadership and mentoring on software design. Big thanks to Yi Ding, software engineer, Delphi Megamos, for the support to create the models and for offering suggestions and contributing to the project.

REFERENCES

1. L.Vitkin, T.K.Jestin, “Incorporating Autocode Technology into Software Development Process”, ICSE 2004, pp.51-57.
2. Peter J. Schubert, Lev Vitkin, Frank Winters, “Executable Specs: What makes one, and how are they used?” SAE World congress 2006.
3. AUTOSAR GbR, “Technical Overview”, document version 2.1.0.
4. AUTOSAR GbR, “Requirements on RTE”, document version 1.0.0. Section 3.2.5.4 through 3.2.5.6.
5. Work-Package 4.2.1.1, “Specification of RTE”, document version 0.16.46880, section 5.1.4
6. AUTOSAR Specification of the Virtual Functional Bus, v.1.0.1, section 4.1.4.3.2.
7. MathWorks Automotive Advisory Board (MAAB) “Controller style guidelines for production intent using MATLAB®, Simulink® and Stateflow® Version 2.0”, Jan 29th, 2007
8. Ulrich Eisemann and Michael Beine, “Transforming function models into AUTOSAR software components”, ECE, Feb 2007.
9. Andreas Köhler and Tillman Reck “AUTOSAR-Compliant Functional Modeling with MATLAB®, Simulink®, Stateflow® and Real-Time Workshop® Embedded Coder of a Serial Comfort Body Controller”
10. Devendra Rai, T K Jestin, “Model based software development of non-preemptive multi-tasking system”, The 2nd IEEE/ASME MESA 2006, Beijing, China
11. Thierry Rolina, “Past, Present, and Future of Real-Time Embedded Automotive Software: A Close look at basic concepts of AUTOSAR”, 2005 SAE International
12. OSEK/VDX Operating System Specification, v 2.2.3, release: February 17, 2005
13. www.AUTOSAR.org

CONTACT INFORMATION

Devendra Rai, devenrai@rediffmail.com or,
devendra_rai@virginia.edu

Jestin TK, t.k.jestin@delphi.com

Lev Vitkin, lev.vitkin@delphi.com