AUTOSAR - Challenges and Solutions from a Software Vendor's Perspective

Thomas M. Galla and Roman Pallierer
Elektrobit Austria GmbH
Kaiserstrasse 45/2
A-1070 Vienna, Austria
E-mail: {thomas.galla, roman.pallierer}@elektrobit.com

***Abstract*— The automotive standard AUTOSAR provides a broad standardized basis for ECU software development consisting of over 80 software modules and libraries accompanied by an associated development methodology. The potential benefits of (re-)using these standardized software modules are undisputed and make the use of AUTOSAR very attractive. When deploying AUTOSAR, however, both the OEMs and the Tier1 are confronted with significant challenges.**

This paper highlights some of these major challenges from a software vendor's perspective, namely the dealing with multiple AUTOSAR versions, the handling of the complexity, the need for optimizations to cope with limited hardware resources, the integration with legacy modules, and the support of the migration between different different OEM projects as well as between different AUTOSAR versions. Possible approaches approaches and solutions to these challenges are illustrated using Elektrobit's products EB tresos Studio and EB tresos AutoCore as examples.

***Index Terms*— Automotive embedded software, AUTOSAR**

I. Introduction

The AUTomotive Open System ARchitecture (AUTOSAR) has been founded in 2003 as development partnership including automotive manufacturer, suppliers and tool developers with the goal to create an open and standardized automotive architecture that paves the way for innovative electronic systems that further improve performance, safety and environmental friendlinesss (Hansen, 2004). In 2008, the first series projects exploiting the AUTOSAR technology have been introduced by BMW, followed by Daimler, Audi, VW, PSA and others.

The AUTOSAR technology provides both a software architecture and a configuration methodology. The software architecture (AUTOSAR Consortium, 2009b) clearly distinguishes between application software and basic software. The *application software components* encapsulate all application specific software items (i.e., control loops, interaction with sensor and actuators etc.), while the *basic software modules* provide functionality like protocol stacks for automotive communication protocols, a real-time operating system, and diagnostic modules. The *Runtime Environment* (`Rte`) acts as standardized interface between the application software components and the basic software modules, in order to achieve the technical goals of AUTOSAR such as modularity, scalability, transferability and re-usability of application functions developed by different manufacturers and suppliers. The AUTOSAR configuration methodology (AUTOSAR Consortium, 2009a) provides system-wide and ECU-specific configuration parameters and standardized XML based exchange formats. The *system description* includes all system-wide parameters, i.e., parameters that are relevant for all ECUs (e.g., the communication matrix and the topology). The *ECU extract of the system description* contains only those parts of the system description that are relevant for one specific ECU (e.g., the signals the ECU sends and receives). The *ECU configuration* includes parameters required to configure the basic software modules for one specific ECU. This ECU configuration can be partly derived from the ECU extract of the system description and has to be partly configured locally for ECU-specifica (e.g., number of receive buffers). This configuration methodology allows to clearly separate the configuration tasks between OEM (system configuration) and Tier1 (ECU configuration) and enable both system-wide and ECU-specific optimizations.

Automotive Tier1s are now confronted with this AUTOSAR technology in almost every new project replacing legacy basic software drivers with this standardized architecture. The main questions therefore are how to use this new technology, what are the main challenges, how can these be managed, and what solutions are available? Elektrobit (EB) is one of the main software vendors that implement the AUTOSAR technology: The product EB tresos AutoCore contains all AUTOSAR basic software modules and the `Rte`. EB tresos Studio is an universal configuration tool for all these basic software modules and the `Rte`. This tool can import an AUTOSAR system description or ECU extract of the system description and convert it into the corresponding ECU configuration parameters in order to configure all AUTOSAR basic software modules and the `Rte` for a specific ECU.

Based on the experiences of the last years, this article tries to give some answers to the main challenges when exploiting the AUTOSAR technology by using the EB tresos products.

II. Multiple AUTOSAR Versions

A. Challenges

A new technology – as big as AUTOSAR – requires a development process over several years. The AUTOSAR partnership has subdivided the specification development into three main periods as shown in Figure 1.

1) *Basic development* of the standard resulting in the releases 2.1 and 3.x, the first releases to be used in mass production projects

2) *Selective enhancement* of the standard based on a stable architecture and methodology, resulting in the releases 4.0 and the upcoming 4.1
3) *Maintenance* of the different releases used *for series production*

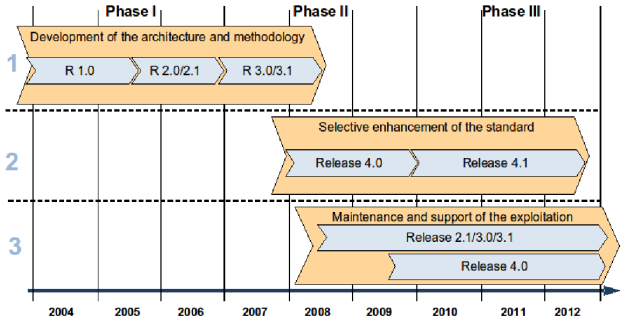


Fig. 1: AUTOSAR – Release Overview (Stefan Bunzel, 2010)

Furthermore, AUTOSAR uses a release and revision numbering scheme that supports incremental extensions. Within a release a complete set of the specifications is available for all AUTOSAR domains: the basic software modules and the `Rte`, the design and configuration methodology (including the corresponding templates), and the application interfaces. Within a revision, smaller updates and extensions of some or all parts of a release are provided.

As long as the specification work is ongoing, OEMs introducing AUTOSAR into mass production projects have the challenging task to determine which AUTOSAR release and revision to be used by the ECUs within one series project. Furthermore, if some OEM requirements are not (yet) covered by the chosen AUTOSAR version, OEMs have to add additional requirement specifications and/or implementations of certain basic software modules, e.g. diagnostic modules, to cover all required features. Tier1s have then the task to use a basic software stack that corresponds to the specified AUTOSAR release and revision and that fulfills all of these additional OEM-specific requirements for a certain series project.

B. Solutions

Based on the AUTOSAR principle "cooperate on specifications and compete on implementations", software vendors are providing implementations of the AUTOSAR standard. However, it is not sufficient for a Tier1 to buy an AUTOSAR stack just according to a certain release and revision, but also to consider all OEM-specific extensions of the targeted mass production project. EB works closely together with all major OEMs introducing AUTOSAR to provide solutions of basic software stacks that fulfill all requirements of a certain mass production project. Tier1s can simply choose between such pre-integrated basic software stacks to immediately start developing their application software on top of the AUTOSAR stack.

III. Handling of Complexity

A. Challenges

An AUTOSAR ECU configuration consists of a *vast amount of configuration parameters* that have to be set properly in order to obtain a working basic software stack consisting of over 80 basic software modules (AUTOSAR Consortium, 2009c) that make up the AUTOSAR software architecture. For example, the AUTOSAR `Com` module, which provides signal-based communication, alone supports over 180 configuration parameters (AUTOSAR Consortium, 2009d). This huge configuration space can only be partly filled by importing the ECU extract of a system description provided by the OEM.

Many of those parameters depend on other parameters, e.g., the range of allowed values of parameter `A` is limited in case parameter `B` is set to a certain value. These *parameter dependencies* have to be considered when crafting a configuration.

Additionally AUTOSAR has the notion of (typed) references to other parameters. Having these references in mind, the *sequence of configuration steps* is worth looking at in order to prevent permanent switching between modules to configure to resolve dangling references and finally getting lost in the huge AUTOSAR configuration space, ending up with many partly configured modules not knowing which modules are completely configured and which ones are not.

Last but not least the Tier1 has to cope with *continuous updates* of the ECU extract provided by the OEM during the various integration phases of the project.

B. Solutions

In order to fill a large amount of the AUTOSAR ECU configuration, an AUTOSAR ECU configuration tool must be capable to import the relevant information from various file formats containing configuration information of the whole system (i.e., the whole car), e.g., proprietary formats like DBC files (for the communication on CAN), or standardized formats as LDF (for the communication on LIN), FIBEX (for the communication on CAN, LIN, FlexRay, and MOST), and last but not least AUTOSAR's ECU extract of the system description (for the communication on CAN, LIN, FlexRay, and Ethernet). EB tresos Studio provides *powerful importers* for each of these formats.

The remaining part of the configuration parameters (i.e., those that cannot be filled by an import) contains a subset of parameters whose value can be derived via some formula from the values of other (already configured) parameters. In order to automate this process the parameter's attributes must be augmented by an attribute containing the *calculation formula* to enable a configuration tool to calculate the derived parameter upon user request.

```
<v:var name="CanIfNumberOfCanRxPduIds" type="INTEGER">
  <a:da name="DEFAULT" type="XPath"
    expr="num:i(count(../CanIfRxPduConfig/*))"/>
```

Fig. 2: Calculation Formula for Derived Parameter

Figure 2 illustrates this approach by showing a snippet of the EB tresos AutoCore `CanIf` module's configuration scheme where the default value for the parameter `CanIfNumberOfCanRxPduIds` is computed based on the number of `CanIfRxPduConfig` elements.

Based on this information the calculation of the default value for the parameter `CanIfNumberOfCanRxPduIds`

can either be *triggered manually* for this single derived parameter alone or for all derived parameters via EB tresos Studio's *AutoCalc Wizard* .

Another part of the parameters that cannot be filled via an import can however be set to OEM specific default values (e.g., a German OEM mandates that the parameter `CanNmWaitBusSleepTime` is set to a value of 0.75 seconds). For this purpose EB tresos Studio provides the possibility to define *pre- and recommended configurations* (i.e., pre-selections of configuration sets). By means of these pre- and recommended configurations certain configuration parameters can be fixed to OEM specific values thus allowing to perform OEM specific customizations to further reduce the configuration space to be filled by the Tier1 (and thus further reducing the Tier1's configuration effort).

The still remaining parameters have to be filled manually by the Tier1. In order to aid the Tier1 in this endeavor, EB tresos Studio on the one hand provides so-called *assistants* that help the user to chose correct values for these configuration parameters in a domain specific GUI (see Figure 3 for an example dealing with the configuration of the `FrIf` module's communication operations).

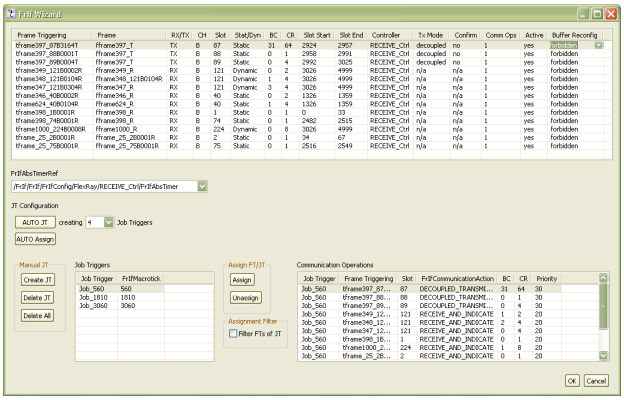


Fig. 3: Screenshot FlexRay Assistant

As far as the configuration order is concerned it is advisable to configure the modules with referenced parameters first and the modules with referencing parameters afterwards. When following this sequence the configuration tool can assist the user during configuration of the referencing parameter by providing a list (e.g., via a drop down box in the GUI) of the syntactically valid choices of possible referenced parameters. This way dangling or even wrong references can be avoided. To ensure that the user configures the various modules in the optimal sequence EB tresos Studio guides the user through the configuration process via *workflows* (see Figure 4). Since there is no single "optimal AUTOSAR workflow", these workflows can be extended, re-arranged and cascaded via XML files for optimal customizations in customer specific environments.

IV. Optimizations

A. Challenges

Due to the strict requirements on cost efficiency, automotive micro-controller devices provide *limited hardware resources*.

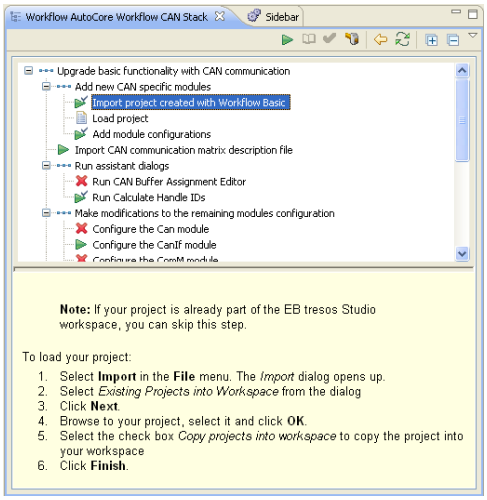


Fig. 4: Screenshot Workflow

The main challenge is to optimize the whole AUTOSAR basic software stack in such a way that the target application can be implemented even on hardware devices with restricted resources. Therefore, the optimal trade-off in the usage of the resources has to be found in the context of the application. Fundamental conflicts of optimization goals, for reducing memory consumption vs. increasing CPU performance, have to be considered.

B. Solutions

Generally, we see three different types of optimization strategies in an AUTOSAR basic software stack:

1) *Functionality optimizations* include methods that deactivate small or large parts of module source code in order to reduce the target code size and to increase the performance. The functional deactivation can be applied on three levels, the deactivation of global functionality (e.g., `Det` reporting), the deactivation of module functionality (e.g., deactivate `PduR` gateway functionality), or the deactivation of module sub-functionality (e.g., disable DLC check of CAN messages in the `CanIf` module).
2) *Resource restrictions* include methods to reduce the number of required resources to fit but not exceed required conditions of an application (Thomas M. Galla, 2008). The main impact of this method is a reduced generated configuration size. This method can be applied on two levels, the limitations of available resources (e.g., reduce the number of CAN controllers to one, thus rendering it unnecessary to explicitly identify the controller in APIs and data structures), or the optimized usage of data types (e.g., using an 8 bit integral data type for an index variables which do not need to address more than 256 objects).
3) *Implementation variants* include methods to use modules in an efficient way. The main impact of this method may differ but often leads to an increase of performance. These methods include zero-cost implementations, i.e., to bypass a module in case the application does not need services from this module (e.g., `PduR` zero cost),

or macro implementations, i.e., to provide pre-processor macros instead of functions (e.g., `SchM` API calls).

The AUTOSAR stack EB tresos AutoCore implements all three types of optimizations. In order to apply the optimizations, the configuration tool EB tresos Studio provides support via following functions: Dedicated assistants (e.g., to deactivate global functionality like `Det` reporting), pre-/recommended configurations for modules, and detailed module configuration editors. The on-line documentation contains for each AUTOSAR configuration parameter a field describing the optimization effect.

C. Examples

To demonstrate the effectiveness of these optimizations, we have conducted measurements regarding the RAM and ROM consumptions with the configurations depicted in Table I.

Configuration Name	# PDUs and Signals	Settings
1sig	1 RX signal (1 bit) 1 TX signal (1 bit) 2 PDUs	`Det` off
1sigOpt	1 RX signal (1 bit) 1 TX signal (1 bit) 2 PDUs	`Det` off `Com` optimized `PduR` zero cost
100sig	50 RX signals (1 bit) 50 TX signals (1 bit) 10 PDUs	`Det` off
100sigOpt	50 RX signals (1 bit) 50 TX signals (1 bit) 10 PDUs	`Det` off `Com` optimized `PduR` zero cost
1000sig	500 RX signals (1 bit) 500 TX signals (1 bit) 100 PDUs	`Det` off
1000sigOpt	500 RX signals (1 bit) 500 TX signals (1 bit) 100 PDUs	`Det` off `Com` optimized `PduR` zero cost

TABLE I: Sample Configurations

Hereby in the optimized configurations (suffix "Opt" in the name) the zero cost optimization has been activated for the `PduR` (strategy 3 in the above list), and the above mentioned optimizations to reduce functionality (strategy 1) and to apply resource restrictions (strategy 2) have been applied to `Com`.

Figure 5 illustrates the resulting memory consumption of the `Com` module, while Figure 6 depicts the resulting memory consumption of the `PduR` module, showing a complete reduction of the code size in all three optimized examples due to the zero cost optimization.

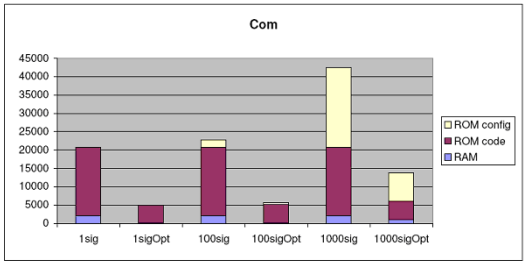


Fig. 5: Memory Consumption `Com`

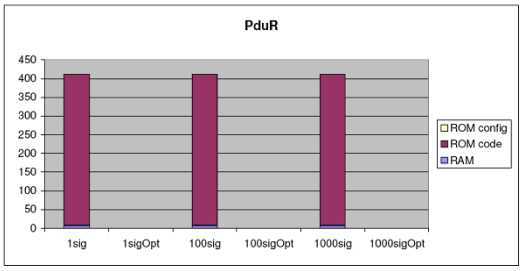


Fig. 6: Memory Consumption `PduR`

V. Integration with Legacy Modules

A. Challenges

In addition to the over 80 software modules and libraries specified by AUTOSAR, each OEM has a certain number of *legacy modules* that need to be integrated into the AUTOSAR basic software stack. For a convenient and efficient configuration process a *seamless integration* with respect to the configuration process, the look and feel in the GUI, and the generation of configuration data structures based on the AUTOSAR ECU configuration XML files is needed.

B. Solutions

To provide such a seamless integration EB tresos Studio supports a *plugin concept* where each basic software module is a plugin from EB tresos Studio's perspective. This plugin concept is easily extensible to the OEM's legacy modules, giving them the same look-and-feel as the standard modules.

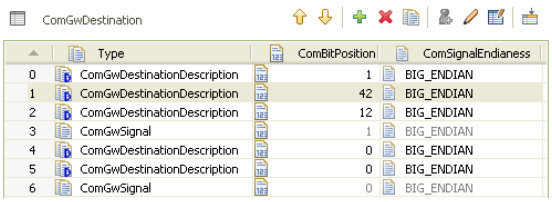
Based on *XML configuration schema files* that are derived from the AUTOSAR ECU configuration XML files and augmented by *GUI annotations* EB tresos Studio renders the GUI on-the-fly when displaying the configuration. Hereby the GUI annotations are used to control the rendering process, e.g., by instructing the renderer to display multiple parameters as columns in a list view (see Figure 7a for a sample XML snippet that instructs the GUI renderer to display various configuration parameters like `ComBitPosition` as columns of a table GUI element and Figure 7b for a screenshot of the resulting GUI layout).

By means of a *template-based generator* the XML configuration files are transformed into configuration data structures in the C programming language. Since this transformation is not encoded in the binary executable of EB tresos Studio but described by means of code generator templates, this transformation is under full control of the user and can easily be modified by simply altering the code generator templates. Figure 8 depicts a snippet of a code generator template that produces a `#define` named `CANIF_DEV_ERROR_DETECT` based on the configuration parameter `CanIfDevErrorDetect` in the `CanIf` module's ECU configuration.

Finally EB tresos Studio provides *generic assistants* for the automatic calculation of dedicated configuration parameters. When properly parameterized, these generic assistants can be registered and used for the legacy modules as well (e.g., for automatic numbering off all protocol data units (PDUs) of the

```
<v:lst name="ComGwDestination" type="MAP">
  <a:a name="COLUMNS">
    <a:v>ComGwDestinationDescription/Type</a:v>
    <a:v>ComGwDestinationDescription/ComBitPosition</a:v>
    <a:v>ComGwDestinationDescription/ComSignalEndianess</a:v>
  </a:a>
```

(a) GUI Annotations in XML



(b) GUI Annotations Screenshot

Fig. 7: GUI annotations

```
/** \brief Switch for DET usage */
[!IF "CanIfPublicConfiguration/CanIfDevErrorDetect"!]
#define CANIF_DEV_ERROR_DETECT STD_ON
[!ELSE!]
#define CANIF_DEV_ERROR_DETECT STD_OFF
[!ENDIF!]
```

Fig. 8: Code Generator Template Snippet

VW legacy module `Ksb`). This way legacy modules exhibit the same configuration comfort as standard AUTOSAR modules.

VI. Migration Issues

A. Challenges

Once a Tier1 has successfully configured the AUTOSAR basic software for a specific OEM project, the Tier1 usually wants to *reuse this configuration* (together with the corresponding application code) in other OEM projects – at least to a large degree. This re-use needs to be addressed in the following three dimensions: the other OEM project

1) mandates a different pre-/recommended configuration
2) uses a different communication matrix
3) uses a different AUTOSAR version

For an optimal support of the Tier1 each of these dimensions must be addressed properly by the configuration tool.

B. Solutions

EB tresos Studio addresses the first two dimensions with a clever merge strategy. Hereby each configuration parameter obtains an additional *origin attribute* which carries the information where the parameter obtained its value from (e.g., from a user input, from an import by a specific importer profile (see below), or via a pre-/recommended configuration). Configuration parameters are treated as identical if the path of the parameter **and** the origin attribute are identical.

When merging a "new" configuration into an "old" one, the merge algorithm behaves as follows: if a specific parameter is

- present in both configurations the value of the parameter in the "old" configuration is replaced by the value in the "new" configuration during the merge process.
- only present in the "old" configuration the parameter is removed during the merge process.
- only present in the "new" configuration the parameter is added during the merge process.

This way even *incremental merging* from different sources is possible if each source is given a unique origin attribute.

When replacing an existing pre-/recommended configuration by a new one, this merge process takes place. – Since the existing parameters that have been initially added by means of a pre-/recommended configuration carry the origin attribute of this configuration, these parameters can be replaced or changed, if the parameters of the new pre-/recommended configuration use the same origin attribute. Other parameters in the existing configuration that do not carry the origin attribute of the new configuration are not affected by this process.

In order to illustrate this process we use the parameter `CanNmWaitBusSleepTime` as an example: In case the Tier1 starts with a project for our German OEM for example, this parameter will be set to a value of 0.75 seconds by means of German OEM's pre-configuration. The fact that this parameter obtained its value from a pre-configuration is recorded in the parameter's origin attribute. If the Tier1 afterwards wants to re-use this configuration in a project for an American OEM that mandates that the parameter `CanNmWaitBusSleepTime` is set to a value of 1 second, the Tier1 simply has to use the American OEM's pre-config. This will cause all parameters of the existing configuration that obtained their values from German OEM's pre-config (i.e., the ones that have their origin attribute set accordingly) to be overwritten with the values of the new pre-configuration. This way the parameter `CanNmWaitBusSleepTime` will be set to a value of 1 in the new configuration.

Combining the import capability of EB tresos Studio with the merge strategy described above changes in the communication matrix can be addressed as well. EB tresos Studio supports the definition of so-called *importer profiles* consisting of the type of the used importer (e.g., FIBEX importer, AUTOSAR system description importer) and the chosen settings for this importer. By using the same importer profile and only changing the imported source file an existing communication matrix can be replaced in the configuration by a new communication matrix trough the merge process without affecting the other parameters of the configuration (e.g., parameters that have been added or changed manually or by some assistant).

Consider the following scenario to illustrate this update process of the communication matrix: The Tier1 initially obtains an ECU extract of a system description from the OEM that contains the signals A, B, and C, where signal A has a length of 8 bits, signal B a length of 16 bits, and signal C a length of 12 bits. In order to import this ECU extract of a system description the Tier1 creates a new importer profile (profile X) containing the type of the importer (i.e., AUTOSAR system description importer in this example) together with the

corresponding importer settings. After an initial import of the ECU extract of the system description these three signal are added to the ECU configuration of the Tier1. Hereby the origin attribute of each of these signals is set to the profile (profile X) used for the import. Afterwards the Tier1 manually adds two additional signals named R and T for debugging purposes. – The origin attribute of these two signals is set to "user input". Later on in the project the Tier1 obtains an update of the ECU extract of a system description from the OEM that contains the signals A, B, and D, where signal A still has a length of 8 bits, signal B's length has changed to 32 bits, and signal D has a length of 4 bits. An import of this ECU extract of the system description using the *same* importer profile causes the signal C to be removed from the Tier1's ECU configuration (since its origin attribute matches the importer profile used for the current import, and signal C is no longer part of the imported system description). Signal D is added to the ECU configuration and the length of signal B is changed to 32 bits. Signal A is not modified (since its length has not changed). Signals R and T remain untouched as well, since they have not been created by the importer profile X (i.e., their origin attribute contains a value that is different from the importer profile X).

The third dimension is addressed by providing dedicated *configuration transformers* for each basic software module. This configuration transformer is implicitly invoked when loading an old ECU configuration. The transformer transforms a configuration which conforms to a previous AUTOSAR versions (or a previous version of the module) into a configuration that conforms to the current AUTOSAR version (or the current version of the module). By applying such a transformer to each module's ECU configuration the Tier1 is able to upgrade a whole configuration project from a previous AUTOSAR version to the most recent supported one.

VII. Conclusion

The AUTOSAR standard is ready to be used in mass production projects. OEMs and Tier1s introducing this powerful technology however have to master some main challenges, such as considering all OEM specific requirements, handling the configuration complexity, optimal usage of restricted hardware resources, and the integration of existing legacy modules. Software vendors such as EB working closely together with OEMs, can provide Tier1s with powerful configuration tools and AUTOSAR basic software stack implementations that fulfill all these requirements. Such product solutions considerably help Tier1s to start with new AUTOSAR projects.

References

AUTOSAR Consortium (2009a). AUTOSAR Methodology, *Technical Report Version 1.4.0, Release 4.0, Rev 0001*, AUTOSAR Consortium.

AUTOSAR Consortium (2009b). Layered Software Architecture, *Technical Report Version 3.0.0, Release 4.0, Rev 0001*, AUTOSAR Consortium.

AUTOSAR Consortium (2009c). List of Basic Software Modules, *Technical Report Version 1.4.0, Release 4.0, Rev 0001*, AUTOSAR Consortium.

AUTOSAR Consortium (2009d). Specification of Communication, *Technical Report Version 4.0.0, Release 4.0, Rev 0001*, AUTOSAR Consortium.

Hansen, P. (2004). AUTOSAR Standard Software Architecture Partnership Takes Shape, *The Hansen Report on Automotive Electronics* **17**(8): 1–3.

Stefan Bunzel (2010). Overview on AUTOSAR Cooperation, *2nd AUTOSAR Open Conference*, Tokyo, Japan.

Thomas M. Galla (2008). Beyond AUTOSAR - Optimized AUTOSAR Compliant Basic Software Modules, *FlexRay Product Day*, Stuttgart, Germany.