RTE Generation and BSW Configuration Tool-Extension for Embedded Automotive Systems

Georg Macher*[||],Rene Obendrauf[||], Eric Armengaud[||], Eugen Brenner* and Christian Kreiner*

*Institute for Technical Informatics, Graz University of Technology, AUSTRIA
Email: {georg.macher, brenner, christian.kreiner}@tugraz.at

[||]AVL List GmbH, Graz, AUSTRIA
Email: {georg.macher, rene.obendrauf, eric.armengaud}@avl.com

Abstract—**Automotive embedded systems have become very complex, are strongly integrated and the safety-criticality and real-time constraints of these systems are raising new challenges. Distributed system development, short time-to-market intervals, and automotive safety standards (such as ISO 26262 [8]) require efficient and consistent product development along the entire development lifecycle. The challenge, however, is to ensure consistency of the concept constraints and configurations along the entire product life cycle. So far, existing solutions are still frequently insufficient when transforming system models with a higher level of abstraction to more concrete engineering models (such as software engineering models).**

The aim of this work is to present a model-driven system-engineering framework addon, which enables the configurations of basic software components and the generation of a runtime environment layer (RTE; interface between application and basic software) for embedded automotive system, consistent with preexisting constraints and system descriptions. With this aim in mind a tool bridge to seamlessly transfer artifacts from system development level to software development level is described. This enables the seamless description of automotive software and software module configurations, from system level requirements to software implementation and therefore ensures both consistency and correctness for the configuration.

Keywords*—*automotive, embedded systems, Model-based development, basic software configuration, traceability, model-based software engineering.

I. Introduction

Embedded systems are already integrated into our everyday lives and play a central role in all domains including automotive, aerospace, healthcare, manufacturing industry, energy, or consumer electronics. Current premium cars implement more than 90 electronic control units (ECU) per car with close to 1 Gigabyte software code [4], these are responsible for 25% of vehicle costs and bring an added value between 40% to 75% [18]. This trend of making use of modern embedded systems, which implement increasingly complex software functions instead of traditional mechanical systems is unbroken in the automotive domain. Similarly, the need is growing for more sophisticated software tools, which support these system and software development processes in a holistic manner. As a consequence, the handling of upcoming issues with modern real-time systems, also in relation to ISO 26262 [8], model-based development (MBD) would appear to be the best approach for supporting the description of the system under development in a more structured manner. Model-based development approaches enable different views for different stakeholders, different levels of abstraction, and provide a central storage of information. This improves the consistency, correctness, and completeness of the system specification. Nevertheless, such seamless integrations of model-based development are still the exception rather than the rule and frequently MBD approaches fall short due to the lack of integration of conceptual and tooling levels [3].

The aim of this paper is to present a tool approach which enables a seamless description of safety-critical software, from requirements at the system level down to software component implementation in a bidirectional way. With the presented tool available hardware- software interfacing (HSI) information can be used to generate basic software (BSW) component configurations, as well as, automatic generation of the runtime environment layer (RTE; interface between application software (ASW) and basic software).

The tool consists of a basic software configuration generator and a software interface generator producing *.c* and *.h* files for linking ASW and BSW. To ensure more versatility of the tool the required HSI information can either be imported from a HSI spreadsheet template or the system model representation. The goal is, on one hand, to support a consistent and traceable refinement from the early concept phase to software implementation, and on the other hand, to combine the versatility and intuitiveness of spreadsheet tools (such as Excel) and the properties of MDB tools (e.g., different views, levels of abstraction, central source of information, and information reuse) bidirectionally to support semi-automatic generation of BSW configuration and the SW-SW interface layer (in AUTOSAR notation known as runtime environment - RTE).

The document is organized as follows:
Section II presents an overview of related works as well as the fundamental model-based development tool chain on which the approach is based. In Section III a description of the proposed tool and a detailed depiction of the contribution parts is provided. An application and evaluation of the approach is presented in Section IV. Finally, this work is concluded in Section V with an overview of the presented approach.

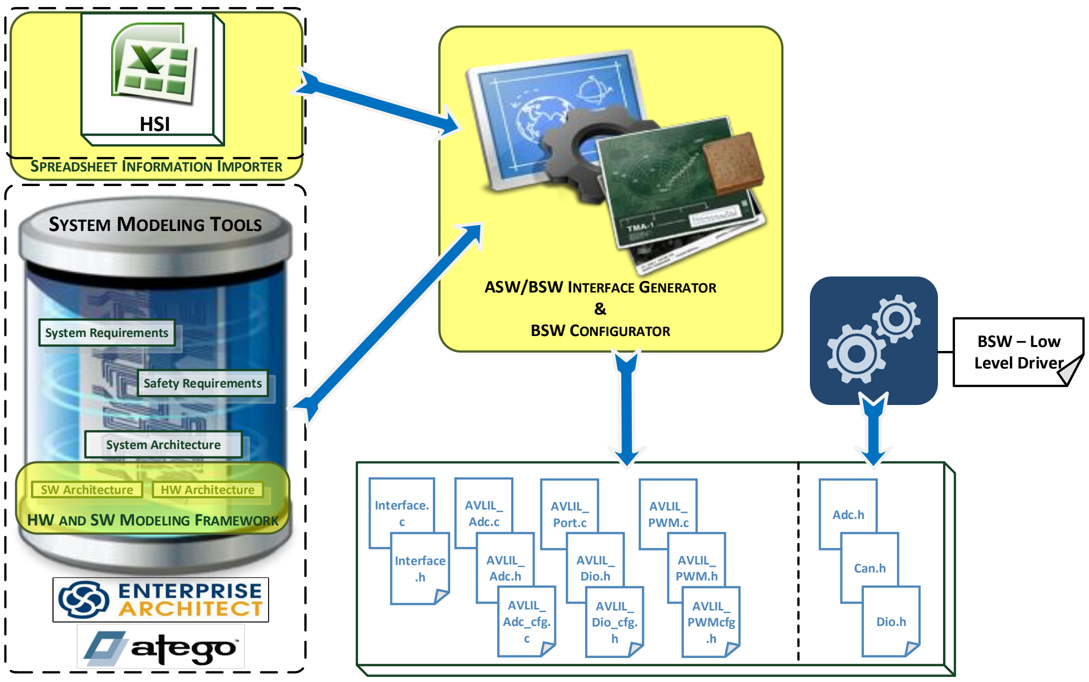


Fig. 2. Portrayal of the Approach for Generation of BSW Configuration and SW Interfaceing Files for the SW Development Phase

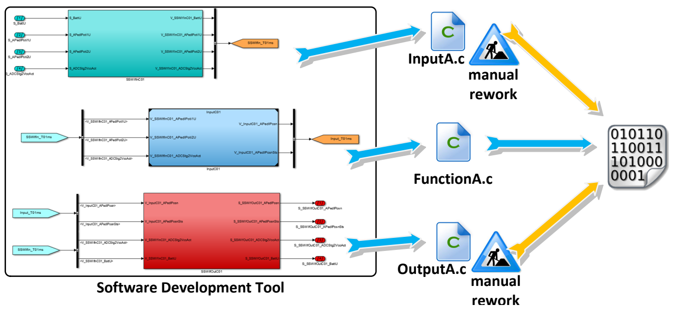


Fig. 1. ICC1 AUTOSAR Approach Methodology with Required Manual Intervention

II. Related Work

Development of automotive embedded software as well as the configuration of the underlying basic software and embedded systems are engineering domains and research topics aimed at moving the development process to an automated work-flow for improving the consistency and tackling the complexity of the software development process across expertise and domain boundaries. Recent publications are mainly based on AUTOSAR [1] methodology.

Due to the ever increasing software complexity of the last few years more and more efforts are becoming necessary to manage the development process of automotive embedded software. To handle this complexity the AUTOSAR consortium was founded and the AUTOSAR methodology provides standardized and clearly defined interfaces between different software components. The AUTOSAR approach features three different classes of implementation (ICC - implementation conformance class). The main benefit of the AUTOSAR ICC1 approach clearly relies on the time-saving in terms of no additional familiarization with usually very complex and time-consuming AUTOSAR tools compared to the full AUTOSAR approach (ICC3). The ICC1 approach does not take advantage of the AUTOSAR benefits from the full AUTOSAR tool-chain supporting tools for RTE configuration and communication interfaces, but standardized component interfaces for exchange of data between the ASW and BSW and therefore features the separation of application specific and hardware specific software parts (like native non-AUTOSAR development). ICC1 mainly focuses on SW engineering and more specifically on the integration of ASW into a given SW architecture. However, the aspects related to control systems engineering (including HW/SW co-design) are not integrated and aspects such as HW/SW interface definition must be performed manually, as indicated in Figure 1. The tool approach introduced in this work enhances this aspect by providing a framework for the visualization of both ASW and BSW interface configuration and automated generation of these interfacing $.c$ and $.h$ files (see Figure 2). Furthermore, the available hardware- software interfacing (HSI) information can be used to generate basic software (BSW) components configurations and the HSI information import functionality can also handle HSI spreadsheet templates to ensure more versatility of the tool.

An approach for an AUTOSAR migration of existing automotive software is described in the work of Kum et. al [10]. The authors highlight the benefits of separating the application software and the basic software and present an approach to configuration of basic software modules instead of time consuming and error-prone manual coding of embedded software. The automatic generation of automotive embedded software and the resultant configuration of the embedded systems thus improves quality as well as re-usability.

In [11], the authors describe a framework for a seamless configuration process for the development of automotive embedded software. The framework is also based on AUTOSAR which defines the architecture, methodology, and application interfaces. The configuration process is established via system configuration and ECU configuration. All the configurations and descriptions used are stored in separate XML (Extensible Markup Language) files, containing described and classified parameters and the associated information. The authors additionally specify a meta-model for the AUTOSAR exchange formats that describe the ECU configuration parameter definition and the ECU configuration description.

Jo et al. [9] describe an approach for the design of a vehicular code generator for distributed automotive systems. The increasing complexity during development of an automotive embedded software and systems and the manual generation of software have the effect of leading to more and more software defects and problems. The authors thus integrated a RTE module into their earlier development phase tool to design and evolve an automated embedded code generator with a predefined generation process. The presented approach saves time through automated generation of software code, compared to manual code generation, it reduces error-prone and time-consuming tasks and is also based on an AUTOSAR aligned approach. The output of the code generator tool is limited to the RTE source code and the application programming interface (API) of the input information. As in our approach, the configuration of software modules, is not focused.

Piao et al. [15] illustrate a design and implementation approach of a RTE generator for automotive embedded software. The RTE layer is located in the middle-ware layer of the AUTOSAR software architecture and combines the top layer mentioned as application software with the underlying hardware and basic software. Automated code generation aims at moving the development steps closer together and thus improving the consistency of the software development process. The output of the automated RTE generator are communication API functions for AUTOSAR SW components of the ASW.

Focusing on software complexity, Jo et al. [7] presents a design for a RTE template structure to manage and develop software modules in automotive industry. The authors focus on the design of a RTE structure also based on the AUTOSAR methodology. Within this design they describe the Virtual Functional BUS (VFB) which establishes independence between the Application Software (ASW) and the underlying basic software (BSW) and hardware.

In [14], an approach for realizing location-transparent interaction between software components is shown. The proposed work illustrates the relationship between the RTE and the VFB and shows which artifacts of the VFB are necessary for the generation of the RTE.

A work depicting the influence of the AUTOSAR methodology on software development tool-chains is presented by Voget [19]. The tool framework presented, named ARTOP (AUTOSAR Tool Platform), is an infrastructure platform that provides features for the development of tools used for the configuration of AUTOSAR systems. The implemented features are base functionalities required for different AUTOSAR tool implementations. The work does not, however, focus on a specific tool integration.

To summarize, none of the approaches described above supports (1) the generation of source code and (2) configuration of the basic software from information available at system level and from system models. The approach we present by contrast, supports not only the automatic generation of the RTE source code, but also the automated generation of basic software configuration of embedded systems from system models.

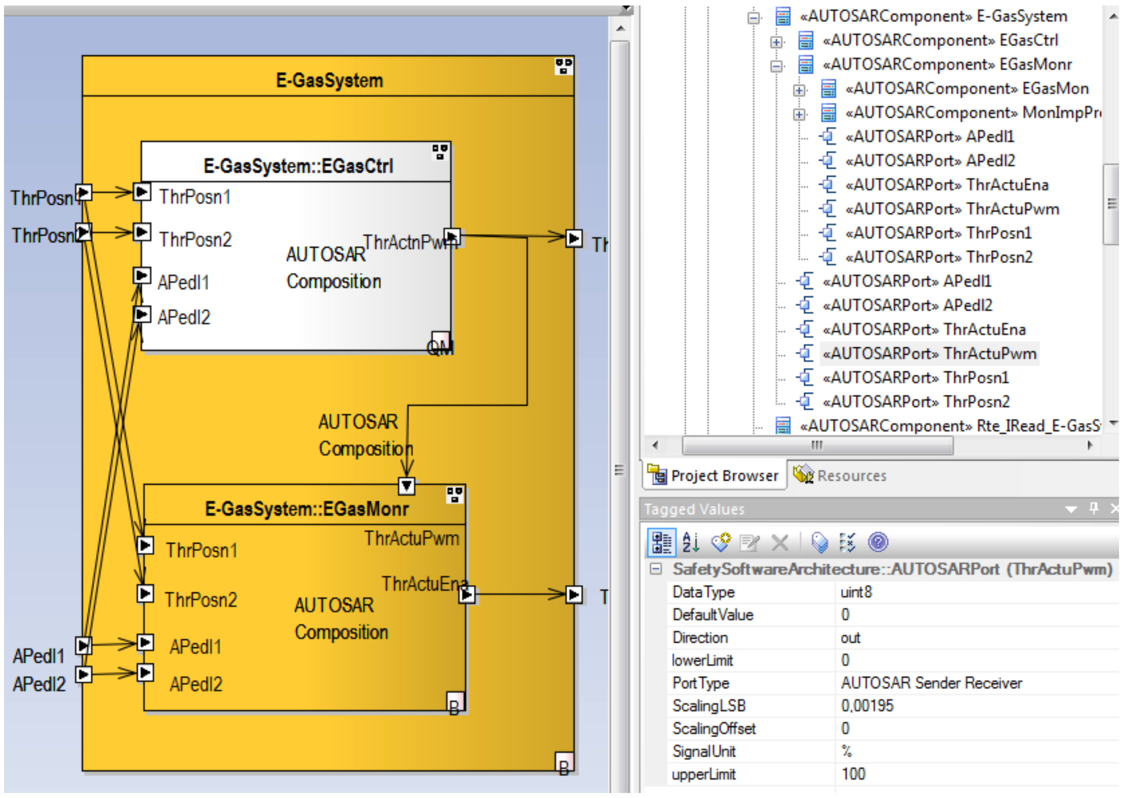
III. Basic Software Interface and Configuration Generation Approach

The underlying concept of the approach is to have a consistent information repository as a central source of information, to store all information of all engineering disciplines involved in embedded automotive system development in a structured manner [13]. The concept focuses on allowing different engineers to do their job in their own specific way, but providing traces and dependency analysis of features concerning the overall system, e.g. safety, security, or dependability. The approach stirs out of common AUTOSAR based approaches and additionally supports a non-AUTOSAR or AUTOSAR ICC1 approach, which are frequently hampered due to a lack of supporting tools. The decision of not fostering a full AUTOSAR approach is based on the one hand on focusing not only AUTOSAR based automotive software development and on the other hand, experiences we have made with our previous approach [12] confirm the problem mentioned by Rodriguez et al. [16]. Not all development tools fully support the entire AUTOSAR standard, because of its complexity, which leads to several mutual incompatibilities and interoperability problems.

Figure 2 shows an overview of the approach and highlights the main contributions. For a more detailed overview of the orchestration for the overall development tool-chain see [13].

The tool approach introduced in this work provides a framework for the visualization of ASW and BSW interface configuration and automated generation of these interfacing $.c$ and $.h$ files (see Figure 2). Furthermore, the available hardware- software interfacing (HSI) information can be used to generate basic software (BSW) component configurations and the HSI information import functionality can also handle HSI spreadsheet templates to ensure more versatility of the tool. More specifically, the contribution proposed in this work consists of the following parts:

- *AUTOSAR aligned UML modeling framework*:
  Enhancement of an UML profile for the definition of AUTOSAR specific artifacts, more precisely, for the definition of the components interfaces (based on the virtual function bus abstraction layer), see Figure 2 – HW and SW Modeling Framework.
- *BSW and HW module modeling framework*:
  Enhancement of an UML profile to describe BSW components and HW components. To ensure consistency of the specification and implementation for the entire control system, see Figure 2 – HW and SW Modeling Framework.
- *RTE generator*:
  Enables the generation of interface files ($.c$ and $.h$) between application-specific and hardware-specific software functions, see Figure 2 – ASW/BSW Interface Generator .



«AUTOSARComponent» E-GasSystem
«AUTOSARComponent» EGasCtrl
«AUTOSARComponent» EGasMonr
«AUTOSARComponent» EGasMon
«AUTOSARComponent» MonImpPr
«AUTOSARPort» APedl1
«AUTOSARPort» APedl2
«AUTOSARPort» ThrActuEna
«AUTOSARPort» ThrActuPwm
«AUTOSARPort» ThrPosn1
«AUTOSARPort» ThrPosn2
«AUTOSARPort» APedl1
«AUTOSARPort» APedl2
«AUTOSARPort» ThrActuEna
«AUTOSARPort» ThrActuPwm
«AUTOSARPort» ThrPosn1
«AUTOSARPort» ThrPosn2
«AUTOSARComponent» Rte_IRead_E-GasS

SafetySoftwareArchitecture::AUTOSARPort (ThrActuPwm)

DataType	uint8
DefaultValue	0
Direction	out
lowerLimit	0
PortType	AUTOSAR Sender Receiver
ScalingLSB	0,00195
ScalingOffset	0
SignalUnit	%
upperLimit	100

Fig. 3. Screenshot of the SW Architecture Representation within the System Development Tool and Representation of the Interface Information

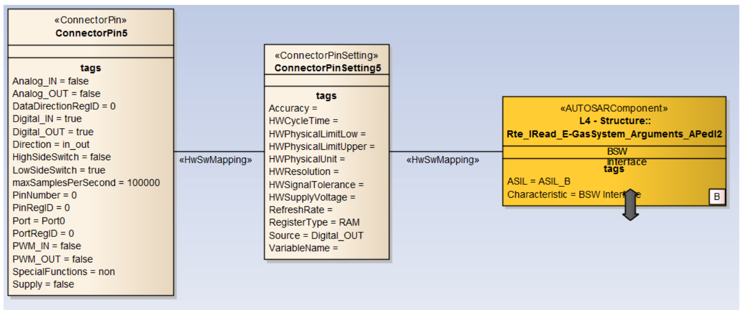
- *Basic software configuration generator*:
  Generates BSW configurations according to the specifications within the HSI definition, see Figure 2 – BSW Configurator.
- *Spreadsheet information importer*:
  Enables the import of HSI definition information done in spreadsheet format, see Figure 2 – Spreadsheet Information Importer.

This proposed approach closes the gap between system-level development of abstract UML-like representations and software-level development, also mentioned by Giese et al. [5], Holtmann et al. [6], and Sandmann and Seibt [17] by supporting consistent information transfer between system engineering tools and software engineering tools. Furthermore the approach minimizes redundant manual information exchange between tools and contributes to simplifying seamless safety argumentation according to ISO 26262 for the developed system. The benefits of this development approach are highly noticeable in terms of re-engineering cycles, tool changes, and reworking of development artifacts with alternating dependencies, as mentioned by Broy et al. [3].

The contribution proposed in this work is part of the framework presented in [13] aiming towards software development in the automotive context. The implementation of the approach is based on versatile C# class libraries (dll) and API command implementations to ensure tool and tool version in-dependence of the general-purpose UML modeling tool (such as Enterprise Architect or Artisan Studio) and other involved tools (such as spreadsheet tool and software development framework). The following sections describe those parts of the approach that make key contributions in more details.

A. *AUTOSAR aligned UML modeling framework*

The first part of the approach is a specific UML modeling framework enabling software architecture design in AUTOSAR like representation within a state-of-the-art system development tool (in this case Enterprise Architect). A specific UML profile to limit the UML possibilities to the needs of software architecture development of safety-critical systems and enable software architecture design in AUTOSAR like representation within the system development tool (Enterprise Architect). In addition to the AUTOSAR VFB abstraction layer [2], the profile enables an explicit definition of components, component interfaces, and connections between interfaces. This provides the possibility to define software architecture and ensures proper definition of the communication between the architecture artifacts, including interface specifications (e.g. upper limits, initial values, formulas). Hence the SW architecture representation within EA can be linked to system development artifacts and traces to requirements can be easily established. This brings further benefits in terms of constraints checking, traceability of development decisions (e.g. for safety case generation), reuse and ensures the versatility to also enable AUTOSAR aligned development as proposed in [12]. Figure 3 shows an example of software architecture artifacts and interface information represented in Enterprise Architect. As can be seen in the depiction, all artifacts required to model the SW architecture are represented and inherit the required information as tagged values.



«ConnectorPin»
ConnectorPin5

tags
Analog_IN = false
Analog_OUT = false
DataDirectionRegID = 0
Digital_IN = true
Digital_OUT = true
Direction = in_out
HighSideSwitch = false
LowSideSwitch = true
maxSamplesPerSecond = 100000
PinNumber = 0
PinRegID = 0
Port = Port0
PortRegID = 0
PWM_IN = false
PWM_OUT = false
SpecialFunctions = non
Supply = false

«HwSwMapping»

«ConnectorPinSetting»
ConnectorPinSetting5

tags
Accuracy =
HWCycleTime =
HWPhysicalLimitLow =
HWPhysicalLimitUpper =
HWPhysicalUnit =
HWResolution =
HWSignalTolerance =
HWSupplyVoltage =
RefreshRate =
RegisterType = RAM
Source = Digital_OUT
VariableName =

«HwSwMapping»

«AUTOSARComponent»
L4 - Structure::
Rte_IRead_E-GasSystem_Arguments_APedl2
BSW
Interface
tags
ASIL = ASIL_B
Characteristic = BSW Interface
B

Fig. 4. Screenshot of the BSW and HW Pin Representation within the System Development Tool

B. BSW and HW Module Modeling Framework

The AUTOSAR architectural approach ensures hardware-independent development of application software modules until a very late development phase and therefore enables application software developers and basic software developers to work in parallel. The hardware profile of the approach allows a graphical representation of hardware resources (such as ADC, CAN), calculation engines (core), and connected peripherals which interact with the software. Special basic software (BSW) and hardware module representations are assigned to establish links to the underlying basic software and hardware layers. This enables an intuitive graphical means of establishing software and hardware dependencies and a hardware-software interface (HSI), as required by ISO 26262. Software signals of BSW modules can be linked to HW port pins via dedicated mappings. On the one hand this enables the modeling and mapping of HW specifics and SW signals, see Figure 4 and on the other hand this mapping establishes traceable links to port pin configurations. A third point is that this HW dependencies can be used to interlink scheduling and task allocation analysis tools for analysis and optimization of resource utilization.

C. Runtime Environment Generator

The third part of presented approach is the SW/SW interface generator. This dll- based tool generates $.c$ and $.h$ files defining SW/SW interfaces between application software signals and basic software signals based on modeled HSI artifacts. In addition, this generation eliminates the need for manual SW/SW interface generation without adequate syntax and semantic support and ensures the reproducibility and traceability of these configurations.

Figure 5 shows the conceptual overview of generated files. The $.c$ and $.h$ files on application software level are generated via a model-based software engineering tool, such as Matlab/Simulink. The files on the basic software level are usually provided by the hardware vendor. While the files referred to in the SW/SW interface layer are generated by our approach.

The generated files are designed in a two-step approach. The first step of the interfacing approach ($interface.c$ and $interface.h$) establishes the interface between ASW and BSW based on AUTOSAR RTE calls. The second step ($AVLIL\_BSWa.c$ and $AVLIL\_BSWa.h$) maps these AUTOSAR RTE based calls to the HW specific implementation

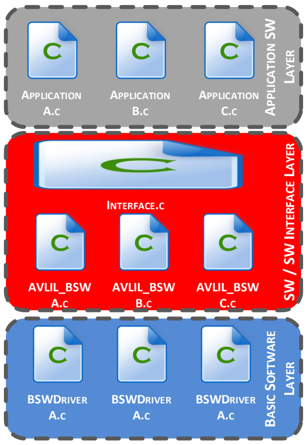


Fig. 5. Overview of Architecture Level Files Generated by the Interface Generator

of basic SW drivers. This ensures independence from implementation of the BSW drivers and also features an AUTOSAR ICC1 approach if needed.

D. Basic Software Configuration Generator

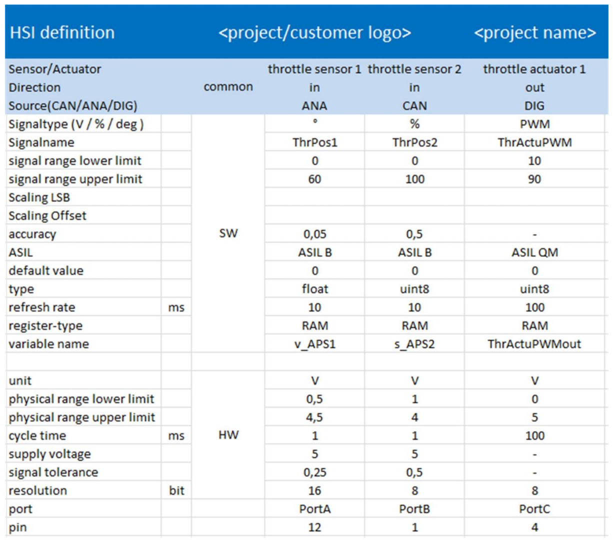
The basic software configuration generator is also part of the dll- based tool, which generates BSW driver specific $*\_cfg.c$ files. These files configure the vendor specific low-level driver (basic software driver) of the HW device according to the HSI specifications. The mapping of HSI specifications to low-level driver configuration is hardware and low-level driver implementation specific and needs to be done once per HW device and supported low-level driver package.

E. HSI Spreadsheet Information Importer

The HSI definition requires mutual domain knowledge of hardware and software and is to be a work product of a collective workshop of hardware, software, and system experts and will act as the linkage between different levels of development. Consistency and evidence of correct implementation of HSI can be challenging in case of concurrent HW and SW development and cross-dependencies of asynchronous update intervals. Therefore, this approach enables a practicable and intuitive way of engineering HSI definitions in a spreadsheet tool (Excel) and transforms them to a reusable and versionable representation in the MDB tool (Enterprise Architect). The spreadsheet template defines the structure of the data representation in a project-specific customizable way. On the one hand this enables a practicable and intuitive means of engineering HSI definitions with spreadsheet tools, while their machine- and human-readable notation ensures a cost- and time-saving alternative to the usually complex special-purpose tools, while on the other hand it enables the generation of SW/SW interface files and BSW configurations without the need for a model-based development toolchain in place. Figure 6 depicts the project-independent template structure for HSI definition data preparation.

IV. Application of the Proposed Approach

This section demonstrates the benefits of the introduced approach for development of automotive embedded systems.



HSI definition		<project/customer logo>			<project name>
Sensor/Actuator			throttle sensor 1	throttle sensor 2	throttle actuator 1
Direction		common	in	in	out
Source(CAN/ANA/DIG)			ANA	CAN	DIG
Signaltype (V / % / deg)		SW	°	%	PWM
Signalname			ThrPos1	ThrPos2	ThrActuPWM
signal range lower limit			0	0	10
signal range upper limit			60	100	90
Scaling LSB					
Scaling Offset					
accuracy			0,05	0,5	-
ASIL			ASIL B	ASIL B	ASIL QM
default value			0	0	0
type			float	uint8	uint8
refresh rate	ms		10	10	100
register-type			RAM	RAM	RAM
variable name			v_APS1	s_APS2	ThrActuPWMout
unit		HW	V	V	V
physical range lower limit			0,5	1	0
physical range upper limit			4,5	4	5
cycle time	ms		1	1	100
supply voltage			5	5	-
signal tolerance			0,25	0,5	-
resolution	bit		16	8	8
port			PortA	PortB	PortC
pin			12	1	4

Fig. 6. Example of a project-independent spreadsheet template structure for HSI definition

TABLE I. OVERVIEW OF THE EVALUATION USE-CASE SW ARCHITECTURE

Object type	**Element-count**	**Configurable Attributes per Element**
ASW Modules	10	3
BSW Modules	7	3
ASW Module Inputs	54	10
ASW Module Outputs	32	10
ASW/ASW Interfaces	48	-
ASW/BSW Interfaces	19	-
HW/SW Interfaces	19	13

We used an automotive battery management system (BMS) as the use-case for the evaluation of the approach. This use-case is an illustrative material, reduced for internal training purposes and is not intended to be either exhaustive in scope or to represent leading-edge technology.

The definition of the software architecture is usually done by a software system architect within the software development tool (Matlab/Simulink). With our approach this work package is included in the system development tool (as shown in Figure 3). This does not hamper the work of the software architect but enables the possibility to also link existing HSI mapping information to the SW architecture (as shown in Figure 4).

The use-case consists of 10 ASW modules and 7 BSW modules with 19 interface definitions between ASW and BSW and makes use of the 3 fundamental low-level HW functions (digital input/output, analog input/outputs, and PWM outputs). A more complete overview of use-case is given in Table I.

The definition of the 19 HW/SW interfaces with 10 parameters for each SW signal and 13 parameters for each HW pin sums up to 437 parameter configurations within the HSI spreadsheet template or in the MDB tool, which can be used to generate ASW/BSW interfaces and BSW configurations.

This results in the file architecture depicted in Figure 7. With the use of the approach 8 additional interfacing files with 481 lines of code (LoC) source and 288 LoC configuration have been generated.

In terms of getting started with AUTOSAR aligned development or supporting non-AUTOSAR SW development our approach features a smooth first step approach of the ICC1 AUTOSAR and generates an interface layer (similar to AUTOSAR RTE) without relying on full AUTOSAR tooling support. In terms of safety-critical development the approach presented supports traceability links between BSW configurations to HSI information and eliminates the need of manual interface source code rework, which further surmounts the main drawbacks of the ICC1 AUTOSAR approach.

V. CONCLUSION

An important challenge for the development of embedded automotive systems is to ensure consistency of the design decisions, SW implementations, and driver configurations, especially in the context of safety-related development. This work presents an approach which seamlessly describes safety-critical software, from requirements at the system level down to software component implementation in a traceable manner. The available hardware- software interfacing (HSI) information can thus be used to generate basic software (BSW) component configurations, as well as automatic software interface layer generation (interface between application software and basic software). With this aim in mind a framework consisting of a basic software configuration generator and a software interface generator producing $.c$ and $.h$ files for linking ASW and BSW has been presented, which can also be used in combination with a spreadsheet based HSI definition. The main benefits of this enhancement are: improved consistency and traceability from the initial design at the system level down to the single CPU driver configuration, together with a reduction of cumbersome and error-prone manual work along the system development path. Further improvements of the approach include the progress in terms of reproducibility and traceability of configurations for software development (such as driver configurations and SW-SW interfaces).

The application of the presented approach has been demonstrated utilizing an automotive BMS use-case, which is intended to be used for training purposes for students and engineers and does not represent either an exhaustive or a commercial sensitive project. While the authors do not claim completeness of the analysis (due to confidentiality issues), the benefits of the approach are already evident.

ACKNOWLEDGMENTS

This work is partially supported by the EMC^2 and the $MEMCONS$ projects.

The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement nr 621429 (project EMC^2) and financial support of the ”COMET K2 - Competence Centers for Excellent Technologies Programme” of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Economy, Family and Youth (BMWFJ),

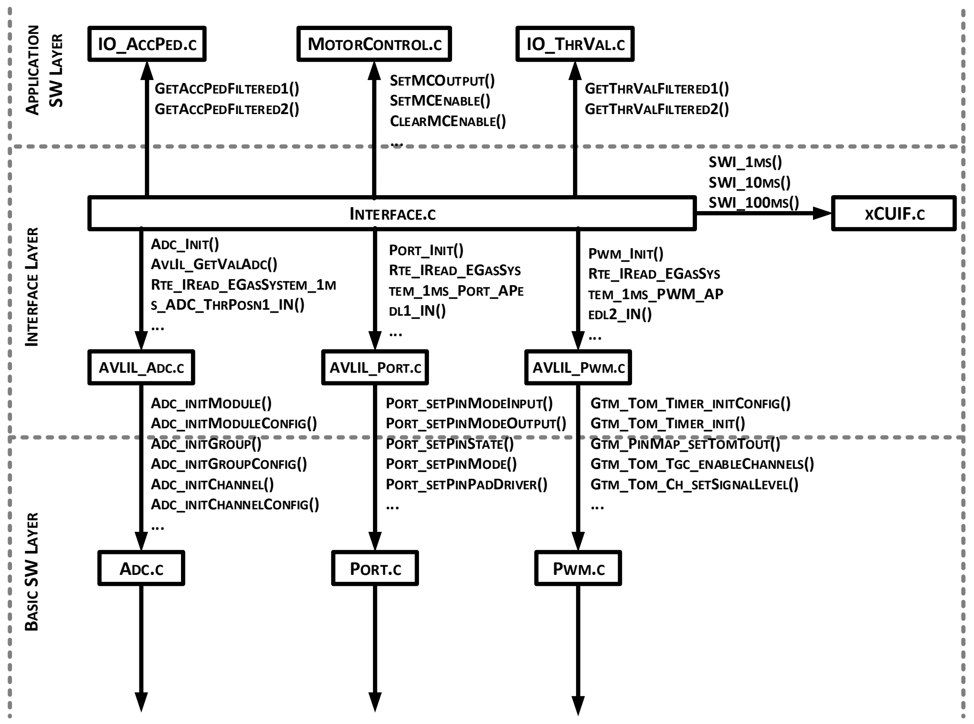


Fig. 7. Excerpt of Generated Files for the BMS Use-Case

the Austrian Research Promotion Agency (FFG), the Province of Styria, and the Styrian Business Promotion Agency (SFG).

Furthermore, we would like to express our thanks to our supporting project partners, AVL List GmbH, Virtual Vehicle Research Center, and Graz University of Technology.

References

[1] AUTOSAR development cooperation. AUTOSAR AUTomotive Open System ARchitecture, 2009.

[2] AUTOSAR Development Cooperation. Virtual Functional Bus. online, 2013.

[3] M. Broy, M. Feilkas, M. Herrmannsdoerfer, S. Merenda, and D. Ratiu. Seamless Model-based Development: from Isolated Tool to Integrated Model Engineering Environments. *IEEE Magazin*, 2008.

[4] C. Ebert and C. Jones. Embedded Software: Facts, Figures, and Future. *IEEE Computer Society*, 0018-9162/09:42–52, 2009.

[5] H. Giese, S. Hildebrandt, and S. Neumann. Model Synchronization at Work: Keeping SysML and AUTOSAR Models Consistent. *LNCS 5765*, pages pp. 555 –579, 2010.

[6] J. Holtmann, J. Meyer, and M. Meyer. A Seamless Model-Based Development Process for Automotive Systems, 2011.

[7] J. Hyun Chul, P. Shiquan, C. Sung Rae, and J. Woo Young. RTE Template Structure for AUTOSAR based Embedded Software Platform. In *Basic Research Program of the Ministry of Education, Science and Technology*, pages 233–237, 2008.

[8] ISO - International Organization for Standardization. ISO 26262 Road vehicles Functional Safety Part 1-10, 2011.

[9] H. C. Jo, S. Piao, and W. Y. Jung. Design of a Vehicular code generator for Distributed Automotive Systems. In *Seventh International Conference on Information Technology*. DGIST, 2010.

[10] D. Kum, G.-M. Park, S. Lee, and W. Jung. AUTOSAR Migration from Existing Automotive Software. In *International Conference on Control, Automation and Systems*, COEX, Seoul, Korea, 2008. DGIST.

[11] J.-C. Lee and T.-M. Han. ECU Configuration Framework based on AUTOSAR ECU Configuration Metamodel. 2009.

[12] G. Macher, E. Armengaud, and C. Kreiner. Automated Generation of AUTOSAR Description File for Safety-Critical Software Architectures. In *12. Workshop Automotive Software Engineering (ASE)*, Lecture Notes in Informatics, pages 2145–2156, 2014.

[13] G. Macher, E. Armengaud, and C. Kreiner. Bridging Automotive Systems, Safety and Software Engineering by a Seamless Tool Chain. In *7th European Congress Embedded Real Time Software and Systems Proceedings*, pages 256 –263, 2014.

[14] N. Naumann. Autosar runtime environment and virtual function bus. Department for System Analysis and Modeling.

[15] S. Piao, H. Jo, S. Jin, and W. Jung. Design and Implementation of RTE Generator for Automotive Embedded Software. In *Seventh ACIS International Conference on Software Engineering Research, Management and Applications*. DGIST, 2009.

[16] E. Rodriguez-Priego, F. Garcia-Izquierdo, and A. Rubio. Modeling Issues: A Survival Guide for a Non-expert Modeler. *Models2010*, 2:361–375, 2010.

[17] G. Sandmann and M. Seibt. AUTOSAR-Compliant Development Workflows: From Architecture to Implementation - Tool Interoperability for Round-Trip Engineering and Verification & Validation. *SAE World Congress & Exhibition 2012*, (SAE 2012-01-0962), 2012.

[18] G. Scuro. Automotive industry: Innovation driven by electronics. http://embedded-computing.com/articles/automotive-industry-innovation-driven-electronics/, 2012.

[19] S. Voget. AUTOSAR and the Automotive Tool Chain. In *DATE10*, 2010.