Automotive Real-time Operating Systems: A Model-Based Configuration Approach

Georg Macher
Institute for Technical Informatics
Graz University of Technology
AVL List GmbH
Graz, AUSTRIA
georg.macher@tugraz.at

Muesluem Atas
Institute for Technical Informatics
Graz University of Technology
AVL List GmbH
Graz, AUSTRIA
muesluem.atas@avl.com

Eric Armengaud
AVL List GmbH
Hans-List-Platz 1
Graz, Austria
eric.armengaud@avl.com

Christian Kreiner
Institute for Technical Informatics
Graz University of Technology
Graz, Austria
christian.kreiner@tugraz.at

ABSTRACT

Automotive embedded systems have become very complex, are strongly integrated, and the safety-criticality and real-time constraints of these systems raise 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 automotive OSEK/ VDX standard provides an architecture for distributed real-time units in vehicles and a language aiming in specifying the configuration of real-time OSEK operating systems. The aim of this paper is to enhance a model-driven system-engineering framework with the capability of generating OS configurations from existing high level control system information. Furthermore, to enable the possibility to update stored information from OSEK Implementation Language (OIL) files and support round-trip engineering of real-time operating system (RTOS) configurations. This enables the seamless description of automotive RTOS, from system level requirements to software implementation and therefore ensures consistency and correctness of the configuration. To that aim, a bidirectional tool bridge is proposed based on OSEK OIL exchange format files.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design; D.2.3 [**Software Engineering**]: Coding Tools and Techniques

General Terms

Model-based development, traceability, embedded operating systems, OSEK OIL, ISO 26262, RTOS.

EWiLi'14, November 2014, Lisbon, Portugal.

1. INTRODUCTION

The number of embedded systems in the automotive domain has grown significantly in recent years. This trend is also strongly supported by the ongoing replacement of traditional mechanical systems with modern embedded systems. This enables the deployment of more advanced control strategies, thus providing added values for the customer and more environment friendly vehicles. At the same time, the higher degree of integration and the safety-criticality of the control application raises new challenges. Evidence of correctness of the different applications, both in the time domain and value domain, possibly running on the same computing platform, has to be guaranteed. In parallel, new computing architectures with services integrated in hardware require new software architectures and safety concepts.

Safety standards such as ISO 26262 [8] for road vehicles have been established to provide guidance during the development of safety-critical systems. These standards rely on risk identification and mitigation strategies. They target early hazard identification as well as solid counter measure specification, implementation and validation along the entire product life cycle. One challenge in this context is to provide evidence of consistency, correctness, and completeness of system specifications over different work-products along the entire product development process. This is a required basis for the development of dependable systems. Moreover, the consolidation of the system specification enables early bug finding and thus support reducing the costs for bug fixes and late re-design.

To handle these issues, model-based development supports the description of the system under development in a more structured way, enables different views for different stakeholders, different levels of abstraction, and central source of information.

The contribution of this paper is to bridge the existing gap between model-driven system engineering tools and software engineering tools for automotive real-time operating systems (RTOS). More especially, the approach makes use of existing high level control system information in SysML format to generate the configuration of automotive real-time operating

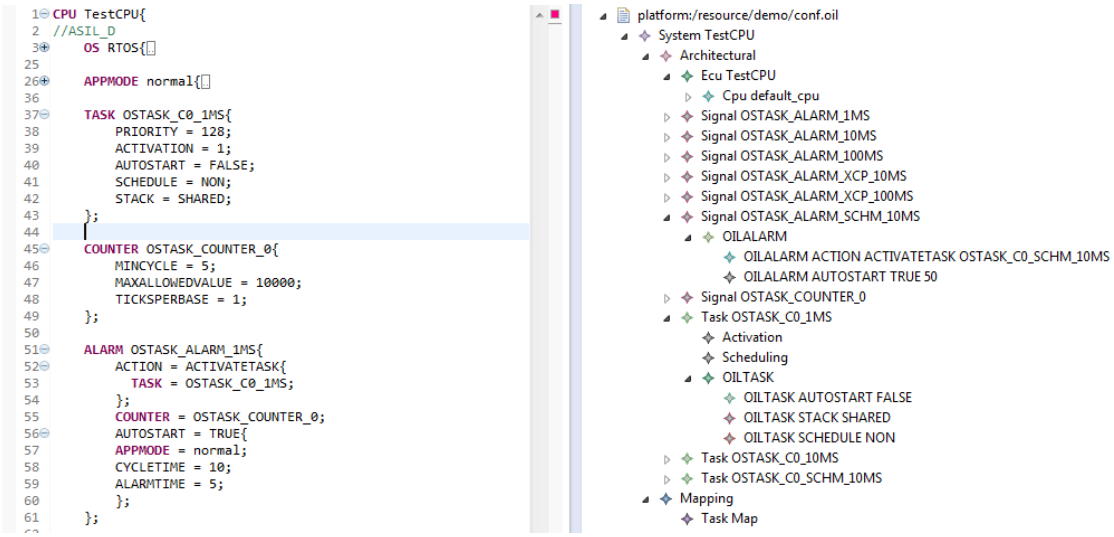

Figure 1: Comparison OIL File Normal View vs. Graphical Representation with Eclipse

systems in a standardized OSEK Implementation Language file format (OIL files) [14]. Information from the control system (such as control strategies) can thus be mapped to a configuration at software level (e.g., required interfaces to other SW components, allocation to a CPU respectively to a task). The goal is to support a consistent and traceable refinement, as required by ISO 26262 standard, from the early concept phase to individual configurations of the RTOS.

The document is organized as follows: Section 2 presents an introduction to OSEK/VDX and OSEK OIL. Then, model-based development and integrated tool chains, as well as the base tool-chain for this approach are presented in Section 3. In Section 4 a description of the proposed approach for the generation of RTOS configuration files according to OIL standard is provided. An application and evaluation of the approach is presented in Section 5. Finally, this work is concluded in Section 6 with an overview of the presented approach.

2. OSEK/VDX RTOS OVERVIEW

The German OSEK consortium (German abbreviation for open systems and their interfaces for electronics in motor vehicles) was founded in 1993 by several German automotive companies. VDX (Vehicle Distributed eXecutive) was the French pendant from the French car manufacturers' side, which regrouped the OSEK/VDX consortium in 1994.

OSEK/VDX is an open standard for specifications for embedded real-time operating systems (RTOS), designed to provide a standard software architecture for the various electronic control units (ECUs), and partially standardized in ISO 17356 [7].

The work of the OSEK/VDX consortium is today continued by the AUTOSAR consortium [1], which is based on OSEK/VDX specifications. To describe the configuration of an OSEK RTOS the OSEK implementation language files (OIL) is intended to be used. These files can be generated manually or via configuration tools. OIL files include all object containers and information required to configure the RTOS of one specific ECU.

2.1 OSEK Implementation Language

As mentioned previously, the OIL files inherit a normalized description language for OS configuration and related objects. OIL files are commonly used in the automotive domain to configure the real-time operating systems of individual ECUs. This is frequently done manually, due to the simple human readable structure of OIL files and the lack of tools supporting an automated information exchange. OIL files typically consist of implementation specific definitions, which are closely related to the hardware (ECU) in use and specify the OIL object with all their possible attribute properties.

Due to the introduction of multi-core real-time systems and the awareness of safety-criticality of such systems, tool support and automation of OIL generation becomes increasingly relevant. An example of safety-related configuration parameters contained in the OIL file are shared task resources or task priorities.

2.2 OSEK Related Tools and Publications

To our knowledge most development frameworks do not include a tool for automatic OIL file generation from prior information of previous development stages at a higher abstraction level. Nevertheless, within this work tool information for commercial tools has been omitted due to non-exhaustiveness of such an overview and the fact that this information can be found up-to-date on the respective website (e.g., Vector OIL Configurator or GOB - GUI based OIL Builder).

Most frameworks either require manual generation of OIL files by the developer, or they provide a dedicated graphical user interface for support and guidance while generating the OIL file. Figure 1 shows a comparison of the same OIL file in typical editor view and in a guided graphical representation within an Eclipse-based development framework. Many available OIL file configurators provide such a representation of the OIL information. They thus provide guidance to minimize configuration failures, but do not reduce workload or speed up the generation of OIL files. Also the import of

prior available information is very limited.

SmartOSEK's visual designer [17] makes use of a DSL (domain specific language) approach. The visual designing tool enables modeling of applications in a graphical way and automatic generation of OIL files. The main drawback of this approach is the missing availability to feedback information into the model.

Most OIL configurators focus on generating OIL files at software development level, such as in the work of Koester et al. [10]. The main disadvantage here is that prior information of previous development phases cannot be used for timing analysis or have to be transferred manually.

Kim et al. [9] suggest a lightweight AUTOSAR software platform and additional extensions of OSEK OIL files. The presented approach focuses on adding extensions to OIL files, rather than supporting automated generation of OIL files.

The work of Yang et al. [16] presents a conversion of UML models into OSEK/VDX models for simulation and optimization of the system design. The authors claim that by converting UML representations into OSEK/VDX models productivity can be improved, correctness of development artifacts can be ensured more easily, and documentation can be provided with less effort and better quality.

Gu et al. [5] focus on the description of automated mechanisms for generating application codes and seamless integration of models for software development, but do not provide methods of the transformation of UML and OSEK artifacts.

3. MODEL-BASED DEVELOPMENT AND INTEGRATED TOOLCHAINS

This section provides a brief overview of model-based development (MBD) tools and related works, as well as the basic MBD framework of the presented approach. Again a tool overview of commercial tools has been omitted due to non-exhaustiveness and easy up-to-date online access of such information on the respective website (e.g., Enterprise Architect, Artisan Studio, EB studio, PREEvision).

3.1 Model-Based Development Tools and Publications

Fabbrini et al. [3] provide an overview of software engineering in the European automotive industry and present tools, techniques and countermeasures to prevent faults. This work highlights the importance of tool integration and model-based development approaches.

Broy et al. [2] mention model-based development as the best approach to manage the large amount of information and complexity of modern embedded systems. The authors also illustrate why seamless solutions have not been achieved so far, mention commonly used solutions, and problems that arise by using an inadequate toolchain (e.g. redundancy, inconsistency and lack of automation). This work presents basic ideas and concepts of MBD, but not detailed solutions.

The work of Holtmann et al. [6] also highlights process and tooling gaps between different development process steps. A model-based development process is presented which conforms with the process reference model of Automotive SPICE. With this use-case the authors highlight the lack of automation for artifact traceability and missing guidelines for model selection at varying abstraction levels. The work exposes two important gaps: First, missing links between system level tools and software development tools. Second, inconsistency and redundant information, due to various very specific tools and a lack of automated information transfer.

Giese et al. [4] also highlight the step from system design to software design as critical. System design models have to be correctly transferred to the software engineering model, and later changes must be kept consistent.

The work of Quadri and Sadovykh [15] presents approaches for novel model-driven techniques and new tools supporting design, validation, and simulation. The authors highlight the possibility of high level model analysis for schedulability and use a subset of UML and SysML for their approach. However, configuration of real-time operating systems accounting for timed resource constraints is not addressed in this work.

3.2 Basic Framework

A brief overview of the model-based development toolchain in use and the related preliminary work for the toolchain of the proposed approach is given in this section. The prototype of our toolchain, proposed by Mader [13], is a specific implementation of a tool-independent and language-independent methodology to support continuous safety analyses of system architecture development according to ISO 26262 [8].

The basic concept behind this framework is to have a consistent information repository as central source of information. The toolchain allows different engineering domains to work on one model which provides traces between different artifact types, and ensures timeliness of data. Extension for the modeling tool (Enterprise Architect ®) ensure seamless and consistent transition of information between the repository and various adequate special-purpose tools (such as OS configurators). This approach also inherits an organizational switch from document-centric development approaches to a seamless model-based development approach. For a more detailed overview of the concept and toolchain as a whole see [12].

4. OVERVIEW OF THE CONTRIBUTION

The contribution proposed in this paper is an extension of the previously mentioned framework towards RTOS configuration. The contribution (see also highlighting in Figure 3) comprises the following aspects:

- *UML modeling framework extension*: Enhancement of the software UML profile for visualization and processing of OSEK OIL files. To enable configuration of the OSEK OS by prior available constraints.
- *OIL file generator*: An extractor which automatically generates OIL files from existing information at system development level. This ensures consistency of the specification and implementation for the RTOS.
- *OIL file importer*: The importer supports round-trip engineering by re-importation of information updates from OIL files.

This proposed extension is a constituent of the proposed toolchain in [12] to close the gap between system-level development at abstract UML-like representations and RTOS configuration at software-level. This bridging extends the work presented in [11] on basic software level, guarantees

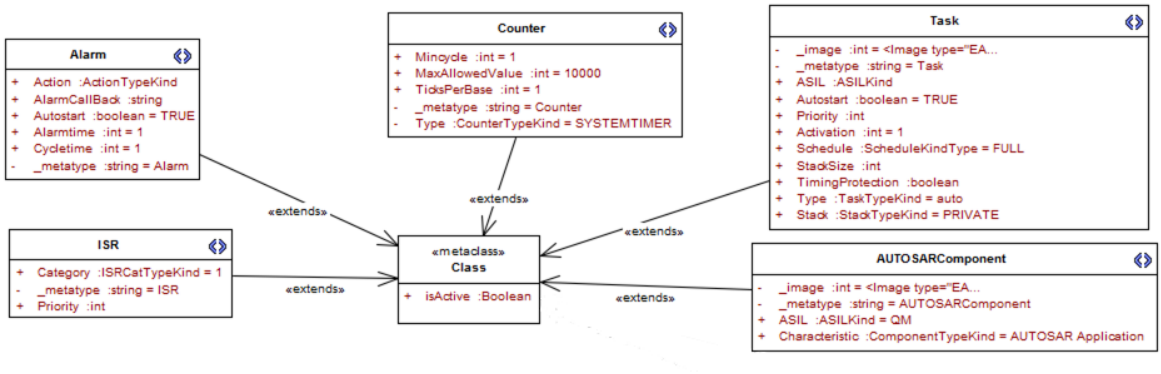

Figure 2: UML Profile Addons for OIL Objects

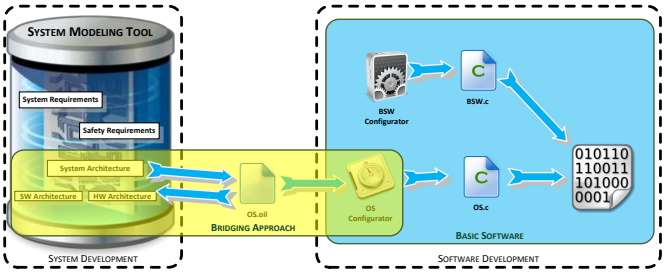

Figure 3: Representation of the Bridging Approach to Ensure Boundless Information Exchange

consistency of information due to the single source of information principle, and shares information more precisely and accurately. The approach minimizes redundant manual information exchange between tools and also takes IS0 26262 requirements (especially traceability) and constraints into account.

Figure 3 shows the conceptual overview of the tool-chain and highlights the OS configuration part. As can be seen from the figure, a lack of tool support for information transfer between system development tools and basic software development tools exist, which has been reduced by the proposed approach. The tight linking of the independent system development and OS configuration tools, to a seamless model-based development toolchain interacting via OIL files, further allows the inclusion of additional tools, such as scheduling analysis tools, seamlessly into the toolchain.

4.1 UML Modeling Framework Extension

The UML profile add-on allows a graphical visualization and processing of OSEK OIL objects. Figure 2 shows a cutout of the profile. This additional information enables the mapping of tasks to a specific core and clear arrangement of dependencies and shared resources in terms of multi-core development. Furthermore, the profile offers an intuitive graphical way of generating OSEK OS configurations and highlighting functionality for safety-related software tasks and resources. This enables the possibility of a traceable automatic OIL file configuration generation, which inherits increasing significance in terms of safety-critical system development according ISO 26262 and traceability. Note that this profile has been integrated in the existing framework described in Section 3.2. Consequently, the system description can be refined down to the operating system, thus improving architecture consistency over skills boundaries (such as systems and software engineering).

4.2 OIL File Generator

The second part of the approach is an exporter capable of exporting the RTOS configuration available from the SysML model to an OIL file. The exporter generates OIL files enriched with the available system and safety development artifact traces (such as required ASIL of task implementation). Most state-of-the-art software development frameworks are able to configure the RTOS according to the specifications within such an OIL file. Consequently the use of this exporter additionally improves communication of the (safety) context to the software experts into their native development tools, thus improving the consistency of the product development. Furthermore, the toolchain is capable of multi-core or multi-system development, therefore the generation of OIL file is selectable for individual cores.

4.3 OIL File Importer

The third part of the approach is the import functionality add-on for the system development tool. This functionality enables bidirectional update of representation in the system development tool and the software development tool. This ensures consistency between system development artifacts and changes done in the software development tool. The importer also implies an overview of changes between database and re-imported OIL file. This offers the possibility of selective database updates and supports impact analysis (as part of the change management process). Finally, the importer enables reuse of available RTOS configurations, guarantees consistency of information, and thereby shares information more precisely and less ambiguously. Figure 4 shows a Screenshot of the import, selective update, and difference highlighting functionality.

5. APPLICATION OF THE APPROACH

This section demonstrates the application of the introduced approach. The application of this approach inherits a tool change for the configuration of the RTOS, from text editor or software development framework to a graphical

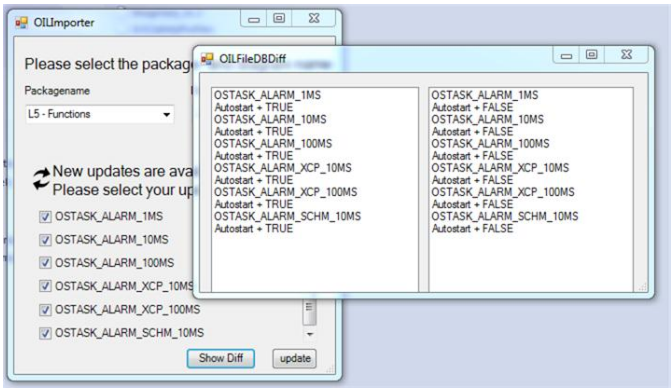

Figure 4: Screenshot of the OIL File Importer

Table 1: OIL Objects of the Evaluation Use-Case

OIL Object	Element-count	Configurable Attributes per Element
CPU	4	2
OS	1	15
APPMODE	2	1
TASKS	6	9
COUNTER	1	5
ALARMS	6	6

representation within the system development tool. Nevertheless, this tool change does not affect the work of the basic software developer in negative ways, but offers a significant benefit for development of safety-critical software in terms of traceability and replicability of development decisions.

With the improvements presented in this paper the extra facility of mapping SW task to dedicated ECU cores and their required resources enables the possibility to unambiguously visualize dependencies and analyze scheduling variants at early development phases. Furthermore, safety-related software artifacts can be explicitly highlighted and dependencies linked in an graphical way.

To provide a comparison of the improvements of our approach we selected a simplified multi-core use-case solely consisting of tasks, alarms, counters, OS, CPU, and application modes. Other OIL objects have been omitted because of the variable multiplicity of these objects (such as resources of a task). An overview of OIL objects within our use-case is given in Table 1.

This amounts to a total of 20 OIL objects and 46 relations between the elements. This small example already indicates that relations between the elements increase quickly and become confusing. To overcome this issue the model-based development approach offers the possibility to hide specific relations.

It might be argued that this approach does not reduce the workload or speed up the generation of OIL files significantly, due to the high number of relations that need to be established. However, the approach provides guidance to minimize configuration failures. Additionally, it supports round-trip engineering features, which split workloads among different development phases and thus simplifies reuse. Table 2 compares the proposed solution with other approaches presented in Section 2.2 and discusses different improvement indicators. The labels for categorizations are:

\+ supported or positive effects
\- not supported or negative effects
o possible or no effects

In terms of safety-critical development and reuse the presented approach supports crucial additional features, such as round-trip engineering by tool-supported information transfer between separated tools and links to supporting safety-relevant information. Furthermore, the approach eliminates need of manual generation of OIL files without adequate syntax and semantic checking support, ensuring reproducibility, and traceability argumentation.

6. CONCLUSION

An important challenge for the development of safety-critical real-time automotive systems is to ensure consistency of the safety relevant artifacts (e.g., safety concepts, requirements and configurations) over the development cycle. This is especially challenging due to the large number of skills, tools, teams and institutions involved in the development. This work presents an approach to bridge tool gaps between an existing model-driven system and safety engineering framework and RTOS configuration tools, based on domain standard OSEK. The implemented tool extension transfers artifacts from system development tools to software development frameworks for RTOS configuration, thereby creating traceable links across tool boundaries, and relying on standardized OSEK OIL exchange files. The main benefits of this enhancement are: improved consistency and traceability from the initial design at the system level down to the single CPU configuration, as well as a reduction of error-prone manual work. Further improvements of the approach include the progress in terms of reproducibility and traceability of safety-critical arguments, configurations for software development, and support of multi-core system development.

Acknowledgments

The authors would like to acknowledge the 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), 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.

Table 2: Improvement Indicators

Improvement Indicators	Proposed Approach	Visual DSL Approach	SW Development Level Configurators	Manual Approach
OIL syntax and semantic checks	+	+	+	o
Reuse	+	+	o	o
Speed-up	o	o	o	o
Distribution of configuration activities	+	o	o	-
Automated configuration from available information	+	+	-	-
Additional (safety) constraints (such as ASIL indicator, requirements)	+	-	-	o
Consistency, correctness, and completeness checks	+	o	o	-
Round-trip engineering support and bi-directional update features	+	o	o	o
Traceability of decision making process	+	+	-	-
Multi-core systems	+	o	o	-

7. REFERENCES

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

[2] Manfred Broy, Martin Feilkas, Markus Herrmannsdoerfer, Stefano Merenda, and Daniel Ratiu. Seamless Model-based Development: from Isolated Tool to Integrated Model Engineering Environments. *IEEE Magazin*, 2008.

[3] Fabrizio Fabbrini, Mario Fusani, Giuseppe Lami, and Edoardo Sivera. Software Engineering in the European Automotive Industry: Achievements and Challenges. In *COMPSAC*, pages 1039–1044. IEEE Computer Society, 2008.

[4] Holger Giese, Stephan Hildebrandt, and Stefan Neumann. Model Synchronization at Work: Keeping SysML and AUTOSAR Models Consistent. *LNCS 5765*, pages pp. 555 –579, 2010.

[5] Zonghua Gu, Shige Wang, and Kang Shin. Issues in Mapping from UML Real-Time Profile to OSEK. In *Proc SVERTS: Workshop on Specification and Validation of UML models for Real Time and Embedded Systems, Workshop at UML 2003, October*, page 7, 2003.

[6] Joerg Holtmann, Jan Meyer, and Matthias Meyer. A Seamless Model-Based Development Process for Automotive Systems, 2011.

[7] ISO - International Organization for Standardization. ISO 17356 Road vehicles – Open interface for embedded automotive applications, 2005.

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

[9] JaeYoung Kim, JungWook Lee, Jeongho Son, Kee-Koo Kwon, and Gwangsu Kim. Lightweight AUTOSAR Software Platform for Automotive. In *IEEE International Conference on Consumer Electronics*, pages 307 – 308, 2012.

[10] Lutz Koester, Thomas Thomsen, and Ralf Stracke. Connecting Simulink to OSEK: Automatic Code Generation for Real-Time Operating Systems with TargetLink. *Embedded Intelligence 2001*, 2001.

[11] Georg Macher, Eric Armengaud, and Christian Kreiner. Automated Generation of AUTOSAR Description File for Safety-Critical Software Architectures. In *Lecture Notes in Informatics*, 2014.

[12] Georg Macher, Eric Armengaud, and Christian 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.

[13] Roland Mader. *Computer-Aided Model-Based Safety Engineering of Automotive Systems*. PhD thesis, Graz University of Technology, 2012.

[14] OSEK/VDX Steering Committee. OSEK/VDX System Generation OIL: OSEK Implementation Language. http://portal.osek-vdx.org/files/pdf/specs/oil25.pdf, 2004.

[15] Imran Rafiq Quadri and Andrey Sadovykh. MADES: A SysML/MARTE high level methodology for real-time and embedded systems, 2011.

[16] Guoqing Yang, Minde Zhao, Lei Wang, and Zhaohui Wu. Model-based Design and Verification of Automotive Electronics Compliant with OSEK/VDX. In *Proceedings of the Second International Conference on Embedded Software and Systems*, ICESS '05, pages 237–245, Washington, DC, USA, 2005. IEEE Computer Society.

[17] Mingde Zhao, Zhaohui Wu, Guoqing Yang, Lei Wang 0023, and Wei Chen 0005. SmartOSEK: A Real-Time Operating System for Automotive Electronics. In Zhaohui Wu, Chun Chen, Minyi Guo, and Jiajun Bu, editors, *ICESS*, volume 3605 of *Lecture Notes in Computer Science*, pages 437–442. Springer, 2004.