

A Model-Based Configuration Approach for Automotive Real-Time Operating Systems

Georg Macher
Graz University of Technology

Muesluem Atas and Eric Armengaud
AVL List GmbH

Christian Kreiner
Graz University of Technology

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. The OSEK/VDX standard provides an open-ended architecture for distributed real-time capable units in vehicles. This is supported by the OSEK Implementation Language (OIL), a language aiming at specifying the configuration of these real-time operating systems. The challenge, however, is to ensure consistency of the concept constraints and configurations along the entire product development.

The contribution of this paper is to bridge the existing gap between model-driven systems engineering and software engineering for automotive real-time operating systems (RTOS). For this purpose a bidirectional tool bridge has been established based on OSEK OIL exchange format files. The approach makes use of existing high-level control system information in SysML format to generate the configuration of automotive real-time operating systems. The implemented tool extension transfers artifacts from system engineering domain to software development frameworks for RTOS configuration, thereby creating traceable links across domain and tool boundaries, and relying on standardized OSEK OIL exchange files.

CITATION: Macher, G., Atas, M., Armengaud, E., and Kreiner, C., "A Model-Based Configuration Approach for Automotive Real-Time Operating Systems," *SAE Int. J. Passeng. Cars – Electron. Electr. Syst.* 8(2):2015, doi:10.4271/2015-01-0183.

INTRODUCTION

The number of embedded systems in the automotive domain has grown significantly in recent years. Current premium cars implement more than 90 electronic control units (ECU) with close to 1 Gigabyte software code [1]. In 2018 30% of the overall vehicle costs are predicted to stem from vehicle electronics [2]. 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 new benefits for the customer and environment, such as reduced fuel consumption and better driveability. At the same time, the higher degree of integration and the safety-criticality of the control application raise new challenges. Hence, the 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 enable the development of new software architectures and safety concepts.

Safety standards such as ISO 26262 [3] 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, further supported by appropriate quality management. They target early hazard identification as well as solid counter measure specification, implementation and validation along the entire product life cycle.

The challenge in this context is to provide evidence of consistency during product development among the different work-products. To handle upcoming issues with modern real-time systems in relation to ISO 26262, model-based development supports the description of the system under development in a more structured way. Model-based development approaches enable different views for different stakeholders, different levels of abstraction, and central storage for information. This improves the consistency, correctness, and completeness of the system specification and thus supports the demands of time-to-market (first time right).

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 relies on the generation of standardized OSEK implementation language files (OIL files) [4] for automotive RTOS based on existing high-level control system information in SysML format. The goal is to support a consistent and traceable refinement from the early concept phase to individual configurations of the RTOS.

The document is organized as follows:

The following section presents an introduction to OSEK/VDX and OSEK OIL as well as model-based development and integrated tool chains. In the third section, a description of the proposed approach for the generation of RTOS configuration files according to the OIL standard is provided. An application and evaluation of the approach is presented in the fourth section. Finally, this work is concluded with an overview of the presented approach.

OSEK/VDX AND MODEL-BASED DEVELOPMENT OVERVIEW

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

OSEK/VDX is an open standard for specifications for embedded real-time operating systems (RTOS) designed to provide standard software architecture for the various electronic control units (ECUs) and partially standardized in ISO 17356. The work of the OSEK/VDX consortium is now continued by the AUTOSAR consortium [5], which is based on OSEK/VDX specifications. Therefore, several publications deal with OSEK/VDX approaches and different tool vendors provide OSEK/VDX tools, which are supporting at different development stages.

The OSEK implementation language files (OIL) are intended to be used to describe OSEK RTOS. These files may be generated manually or via configuration tools. In both cases, the OIL files include all object containers and information required to configure the RTOS of one specific ECU. This is possible due to the fact that automotive real-time systems are typically statically configured. In accordance with this, features of an OSEK implementation are configured at compile-time and changes at run-time are prohibited. Only static priorities are allowed for tasks and FIFO scheduling is used for tasks with equal priority. Deadlocks and priority inversion are prevented by a priority ceiling approach.

The following paragraph provides a brief overview of OIL files and included objects to configure automotive RTOS. The next paragraph briefly describes both OSEK/VDX supporting tools and publications related to OSEK OIL. Nevertheless, this is not intended to be a full description or complete overview; the main focus is set on RTOS configuration with OIL files. The third paragraph of this section

provides an overview of publications dealing with model-based development approaches and integrated tool chains for real-time systems. The final paragraph highlights the basic framework for the proposed approach.

OSEK Implementation Language Files

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 and all possible attribute properties. Secondly, OIL files consist of application specific definitions that are, as the name implies, specific to one application and specify the attribute of the OIL object.

OIL objects and their descriptions:

- **CPU** - specification of the core which runs the application
- **OS** - specification of the RTOS of the specific core
- **APPMODE** - definition of different application modes to control task features or support reduced functionalities
- **ISR** - specification of interrupt service routines
- **RESOURCE** - specification of different resources of the ECU
- **TASK** - software functionality handled by the RTOS
- **COUNTER** - a SW/HW resource for alarms
- **EVENT** - synchronization mechanism for tasks
- **ALARM** - notification mechanism
- **COM** - definition of the OSEK communication subsystem
- **MESSAGE** - definition of data exchange between tasks
- **NETWORK MESSAGE** - definition of data exchange between different CPUs
- **NM** - network management subsystem configurations

As mentioned previously, the creation of OIL files is done for each CPU individually and very often manually, due to the project-specific nature of OIL files. Nevertheless, due to the introduction of multi-core real-time systems and current awareness of safety-criticality of such system configurations, tool support and automation of OIL generation becomes increasingly relevant.

OSEK/VDX Related Publications and Tools

To our knowledge, most development frameworks do not include a tool for automatic OIL file generation of previous development stages at higher abstraction level, e.g. systems engineering. 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. Such representations do not support the automatic generation of OIL files from available information of previous development steps, nor do

they significantly speedup the configuration process of RTOS. Many available OIL file configurators provide such a representation of the OIL information. They thereby provide guidance to minimize configuration failures, but do not significantly reduce workload or speed up the generation of OIL files. Also the import of prior available information is very limited. Vector's OIL Configurator and GOB - GUI based OIL Builder are mentioned as representatives of such commercial and non-commercial tool implementations. This group of tools will be referenced as SW Development Level Configurators in the course of this document.

A DSL (domain specific language) approach was applied with SmartOSEK's visual designer [6], a part of the SmartOSEK platform. The visual design tool enables modeling of applications in a graphical way and automatically generates the designed OIL file. The graphic design helps developers devote their minds to the modeling of the application rather than applying correct OIL syntax and semantics. Drawbacks of this approach are the limited usability, solely for SmartOSEK platform, and the missing availability of feedback information into the model.

Most OIL configurators, like the previously mentioned Vector OIL Configurator, GOB, and Freescale's OSEK Builder Tool focus on generation of OIL files at a software development level. The work of Koester et al. [7] is another example of achieving OSEK-compliant code after software development.

This implies the drawback that information of prior development phases cannot be used for timing analysis or has to be transferred manually to the corresponding OIL files.

The publication of Kim et al. [8] suggests a lightweight AUTOSAR software platform and additional extensions of OSEK OIL files. This approach offers the possibility of adding system protection functions, real-time scheduling functionalities, and additional timing attributes to the standard OIL files. The platform is improved for development of time-critical application software. Nevertheless, the presented approach focuses on adding extensions to OIL files, rather than supporting automated generation of OIL files or the origin of such information.

Yang et al. [9] base their work on the SmartOSEK platform mentioned earlier in this section and present the conversion of UML models into OSEK/VDX models for simulation and optimization of the system design. The authors claim that by converting UML into OSEK/VDX models productivity can be improved, correctness of development artifacts can be, more easily ensured and documentation can be provided with less effort and better quality.

Gu et al. [10] focus on the mapping of RT-UML (UML real-time profile) to OSEK APIs. The authors focus on the description of automated mechanisms for generation of application code and seamless integration of models for software development, but do not provide methods of the transformation of UML and OSEK artifacts.

In contrast to the last two publications, our approach focuses on the safety-critical development of multi-core systems. The two approaches have solely the ability to provide a graphical representation of an OSEK OIL file and thereby improve productivity. Our approach, in contrast, enables an automated information exchange and generation also for safety and multi-core constraints (non-functional requirements which are hard to trace and verify). This supports several factors required for safety-critical system development (e.g. tracing of automotive safety integrity levels (ASIL) and ASIL decompositions, proof of freedom from interference, proof of separation, support of safety-case generation).

Model-Based Development and Integrated Tool Chains

Broy et al. [11] mention concepts and theories for the model-based development of embedded software systems. The authors also claim model-based development to be the best approach to manage the large amount of information and complexity of modern embedded systems with safety constraints. The paper illustrates why seamless solutions have not been achieved so far, they mention commonly used solutions, and problems that arise through the use of an inadequate tool chain (e.g. redundancy, inconsistency and lack of automation). The focus of their work is to present basic ideas and concepts, although no detailed solutions for the automotive domain are presented.

The work of Quadri and Sadovykh [12] presents a real-time embedded system for avionics and a model-driven engineering approach aiming to develop novel model-driven techniques and new tools supporting design, validation, and simulation. They defined profiles using a subset of UML and SysML for their approach and mentioned the usage of effective design tools and methodologies as crucial to be capable of managing complex real-time embedded systems. Furthermore, they highlighted the possibility of high-level model analysis for schedulability.

The work of Holtmann et al. [13] highlights process and tooling gaps between different modeling aspects of a model-based development process. Often, different specialized models for specific aspects are used at different development stages with varying abstraction levels. Traceability between these different models is commonly established via manual linking. The authors claim that there is a lack of automation for those linking tasks and no guidance which model is to be used at which specific development stage. A model-based development process is presented that conforms to the process reference model of Automotive SPICE. The proposed tool chain mentions two important gaps: First, missing links between system level tools and software development tools. Second, several very specific and non-interacting tools that require manual synchronization and which are therefore often inconsistent, rely on redundant information and, due to a lack of automation, require redundant manual work.