Handling System Complexity in sCPS: Usable Design Space Exploration

Sebastian Voss
fortiss GmbH
Munich, Bavaria, Germany
voss@fortiss.org

Johannes Eder
fortiss GmbH
Munich, Bavaria, Germany
eder@fortiss.org

ABSTRACT

With a growing demand for complex features in smart cyber physical systems, the design of such system is getting increasingly complex. These features demand sound and scalable approaches to deal with the increasing design space. Consequently, standards (e.g. like ISO26262) propose methods and techniques for the systematic development of (in this case: automotive) systems. The growing amount of functionality correspondingly require more powerful electronic platforms and thus methods to deal with the integration. In this paper, we describe drivers for complexity and illustrate how formal methods, namely design space exploration techniques, can be applied to deal with this complexity. This approach is based on requirements defined by the given standards and supports the system designer by a (semi-) automatic approach to handle the complexity in system design – already in early design phases.

KEYWORDS

Model-based Systems Engineering, Design Space Exploration, Systems Engineering, Handling Complexity

ACM Reference Format:
Sebastian Voss and Johannes Eder. 2018. Handling System Complexity in sCPS: Usable Design Space Exploration. In *SEsCPS'18: SEsCPS'18:IEEE/ACM 4th International Workshop on Software Engineering for Smart Cyber-Physical Systems , May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3196478.3196489

1 INTRODUCTION

To effectively manage the ever increasing complexity in the design of smart CPS or even collaborative systems, development processes in general, and model-based approaches in particular, support the development assuming an idealized (component-based) model of computation, abstracting away from implementation issues like interference aspects of the execution platform resulting from shared computation or memory resources.

In automotive system design, for instance, standards like the ISO 26262 request that those simplifying abstraction must be met by development steps that ensure that the assumptions behind these abstraction are not violated by the properties of the implementation platform. For example, during SW-/HW-integration platform mechanisms must ensure the defined semantics to guarantee an intended system design.

A proper system design has become, due to a set of complexity drivers (compare section **??**), a manually unsolvable task and requests for (semi-) automatic techniques to support the system designer. Therefore, one of the most asked questions in industry w.r.t. the engineering of smart CPS or even collaborative (automotive) systems is also one of the most challenging one:

"What is a good or even optimized/optimal system design and how to find it - automatically?".

The definition of what is a "good" or even "optimal" system design requires a detailed (formalized) understanding of the different system parts that are concerned, their interconnect, as well as all requirements, namely the constraints and objectives, that are involved.

In this work, we illustrate how the use of model-based systems enginnering in the development of automotive systems enable a variety of front-loading methods that enable to explore the design space of smart CPS to reduce the complexitiy in systems configuration by the applicability of formal methods, namely design space exploration techquiges. This supports the system designer already in early design stages by a (semi-) automatic approach to calcualte design alternatives.

2 DRIVERS OF SMART CPS COMPLEXITY

In recent years, the market of cyber-physical systems, and automotive systems specifically, demands more and increasingly complex systems. The increasingly sophisticated functionalities of automotive vehicles – especially advanced driver assistance like adaptive cruise control or emergency breaking – is implemented by the complex interaction of highly integrated functions provided by a combination of mechanical, electric/electronic, and especially software parts to implement these functions. A substantial part of this complexity is caused by the intricate dependencies of these functions in providing an end-to-end functionality, complicated by the different requirements (e.g. timing, safety, reliability) of these functions.

Thus, these trends can be grouped into the following set of ***complexity drivers***:

(1) On *System Level* an increasing set of sophisticated functionalities starting at a sensor, controlled by (various) controllers, and provided output at (a set of) actuators introduces more and more complexity to the system design. Furthermore, this increasing set of functionality is not only highly dependent, meaning that there is dependency between these functions,

SEsCPS'18, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5728-9/18/05…$15.00
https://doi.org/10.1145/3196478.3196489

but can also typically been characterized by various *variants*. Thus, a function can be defined in different ways, namely different variant (e.g. full-fledged functionality to simple/basic functionality).

(2) On *Software Level* the set of functions are decomposed into software components, software units, etc. as well as their interfaces between them. This decomposotion is (mostly) based on decisions for the hardware level, meaning the set of computing resources (e.g. the introduction of multi-core architectures). This can – especially in early development stages – introduce some uncertanty design constraints.

(3) On *Hardware Level* there is an ongoing trend towards more and more integrated hardware-platforms (e.g. the introduction of multi-core platforms into system design). Such a introduction leds to design decisions to support separation (time and space) due to given system requirements.

(4) Given the fact that *Requirements for all levels* are – by their nature – mostly contradicting, e.g. safety and performance, design alternatives have to take such contradicting requirements into consideration and provide with valid (or even paretro-efficient) solutions to support the system designer.

3 DESIGN OF (AUTOMOTIVE) SYSTEMS

To systematically address the specific challenges sketched above in the automotive industry, the ISO 26262 provides an approach for the development of automotive safety functions.

3.1 Development according to ISO26262

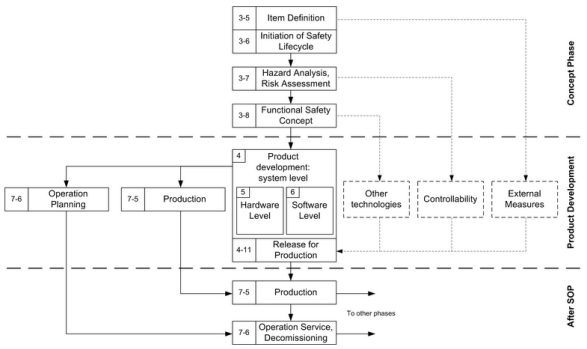

Figure 1: Overview: The ISO 26262 Safety Lifecycle

ISO 26262 [**?**] is the specific instantiation for road vehicles of the general electrical and electronics systems safety standard IEC 61508 [**?**]. We briefly sketch how methods identified there can be used in system development using integrated model-based engineering tools. The ISO 26262 development flow (compare Figure **??**) starts at abstract functional representation of the intended functionality, and leads to concrete representations of the hardware and software (sub-)systems. The following architectural levels – presented in top-down fashion – must be specified for an item – the system or function under consideration– in scope of ISO 26262.

(1) **Vehicle Level**: The *Vehicle Level* describes the context of the item including its interface, its functionality, as well as its decomposition, and specifically targets the hazard and risk assessment.

(2) **System Level**: The *System Level* describes the architectural decomposition of the item, which includes at least one sensor, one controller and one actuator. Furthermore, at this level also the allocation of the different elements to software and hardware components is defined. Additionally, a description of the interfaces between the components is provided.

(3) **Software Level**: The *Software Level* describes the decomposition of the software system into software partitions, software components, and software units, as well as the interfaces between them.

(4) **Hardware Level**: The *Hardware Level* describes the decomposition of the hardware system into hardware component and hardware parts, as well as the interfaces between the them.

In the approach presented here, we focus on the *Product development at System Level* phase, based on the safety goals identified in the *Functional Safety Concept*. This concept provides all functional safety requirements allocated to the elements of the *item* under consideration.

3.2 Component-Based Development

A common design approach for handling the increase in complexity is to decompose systems into subsystems. As part of this decomposition, ideally (more) abstract system models are refined to subsystem models. Therefore, the ISO26262 provides its own component model (compare Figure **??**). An *item* can be seen as the system or the systems under consideration. A *System* can be hierarchically structured and consist out of a set of *functions*. Each system is composed out of a set of *component*s that in turn can be hierarchically structured as well. System(s) are elements or composed of elements. Non-atomic elements are components and atomic elements are either *software parts* or *hardware units*.

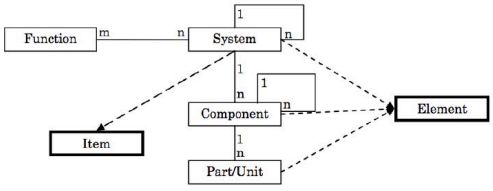

Figure 2: ISO26262: Items, Components, Parts, Units

In order to address these needs, we are able to structure given system models by applying (well known) model-based design methodologies that propose to divide a system under development into different *viewpoints* to cope with the rising complexity. A viewpoint describes the system from the perspective of a specific stakeholder. A view in turn describes the models and languages used to describe a specific viewpoint. SPES 2020 [**?**] and the EAST-ADL2 [**?**] are examples of such development methodologies. The concept of Viewpoint and View can moreover be found in the ISO-42010 standard [**?**].

One of the advantages using such an approach is that the systems functionality can be developed independently from the hardware realization. In the SPES 2020 methodology, for example, this is realized by a *logical* and *technical architecture* of the system.

This idealized model allows abstracting from resource-specific implementation issues, enabling frontloading activities, such as the validation of possible integration variants, namely mappings (deployments) of software parts to hardware units (as requested by the SW- / HW- Integration part of the ISO 26262), balancing the limitations of the available hardware platform against functional requirements.

This is a usual design step, however, finding such valid (or even optimized) mappings which satisfy a multitude of (extra-) functional requirements (e.g. safety, timing, memory consumption, ...) in an industrial sized use case is a manually unsolvable task and require formal methods support, namely *Design Space Exploration (DSE)* techniques.

3.3 Industrial Use Case

In order to get a feeling of a concrete subsystem in automotive industry, we describen in In [**?**] a collaboration with Continental applying DSE to Continentals Traffic Jam Assist (henceforth referred to as TJA). The TJA integrates longitudinal control of a vehicle with lateral control into one seamless full speed range function. In essence, it combines the functionality of a full speed range Adaptive Cruise Control system with Auto-Go capability capable of driving the vehicle at a preset speed while maintaining a preset distance to the preceding vehicle and speeding up or slowing down the vehicle as needed all the way down to a complete standstill, and then re-launch the vehicle together with a full speed Lane Keep Assist system, allowing the vehicle to automatically handle highway road curvatures while keeping it centered in the lane. TJA is an advanced driver assistance system (ADAS) complying to Level 2 automation classification as defined in the SAE J3016 standard for the classification of automotive automation systems.

The logical architecture is composed of components (SysML blocks) which communicate with each other using ports and communication channels (flows). The data type information is annotated on the ports. In total the overall number of model artifacts in the TJA logical architecture is:

- 31 blocks,
- 458 ports,
- 327 information flows and
- 36 data types.

The technical architecture is composed of four types of components (blocks): processing unit, gateway, bus and power supply. The processing units communicate through ports and buses and/or gateways, with energy supplied by power supplies. Software components are deployed and executed on processing units. Buses connect processing units allowing data exchange while gateways connect different buses but do not support software execution (aside from such software as specific for a gateway, which is not part of the current study).

In total the overall number of modeling artifacts in the TJA technical architecture is:

- 22 blocks,
- 63 ports and
- 39 information flows.

3.4 SW- / HW- Integration

In order to build an ISO-conform system, assumptions of the hardware and software system model (e.g. ASIL conformance, timing, memory, cost constraints) may not be violated by the actual implementation platform.

For a (valid) SW- / HW- Integration – a *deployment* – the ISO allows for co-existence of different criteria levels, namely ASILs, within the same element it needs to be shown that a lower-level sub-element does not interfere with any other sub-element assigned a higher ASIL (see ISO26262, part 9, clause 6).

Therefore, time and space separation in HW-/SW-integration need to be guaranteed by a safety-oriented deployment of software tasks to hardware resources, to achieve independence and to avoid propagation of failures. Depending on the assurance level, this independence can be achieved either by hardware separation, or by using middleware (software) mechanisms, e.g. SW-partitions.

4 (USABLE) DESIGN SPACE EXPLORATION

In a process for software-intensive systems as defined by ISO 26262, development is executed by a sequence of steps, incrementally refining the design [**?**]. Exploring the design space in terms of valid and optimized deployments (mappings) from a logical view to a technical view of a system, is such a refinement step. Each step involves decisions made by the system designer. These decisions are restricted by design constraints, that limit the set of possible solutions in the design space and reflect different development objectives. Many of these design constraints – like platform limitations – can be objectified and consequently also formalized. All of these constraints alter the system design, while the requirements of the system still have to be met, while limiting the set of possible solutions. Therefore, to support the system designer, DSE techniques are applied to support in such refinement steps.

In [**?**], we introduced a domain specific modelling language (DSML) which specifies the set of given requirements (e.g. constraints and objectives). This DSML-based requirement specification in combination with the introduced system models, enable the applicability of our (semi-) automatic design space exploration techniques, namely the description of this problem as a satisfiability problem with linear arithmetics. This enables to use state-of-the-arte SMT solvers to support in these refinement steps.

AutoFOCUS3 (`http://af3.fortiss.org`) is a research CASE tool that allows modeling and validating concurrent, reactive, distributed, timed systems on the basis of a formal semantics [**?**]. It provides a graphical user interface supporting the specification of embedded systems in different layers of abstraction while supporting different views on the system model (e.g. from the model-based requirements view down to the hardware-related platform view). The approach is implemented in the framework to form the desired integration models (SW- / HW- integration) as requeste by the ISO26262.

Design Space Exploration in AF3 as a general approach is based on our proposal of a jointly generation of deployments and schedules. In [**?**] we demonstrate how such a joint generation of deployments

and schedules can principally been done for shared-memory multi-core architectures. In [**?**] we apply an extended DSML on the use case described in section **??**. The proposed approach has proven to perform in a scalable fashion for practical sizes, as it relies on a symbolic formalization encoding the deployment synthesis as a satisfiability problem over Boolean formulas and linear arithmetic constraints.

5 LESSONS LEARNED

One of the usual questions, that arise, when thinking about starting with a model-based Systems Engineering approach is: *How much will it cost?* and *What to model (first) ?*. Different companies have different goals and reasons why and where they want to use it. Therefore, in order to make assessments and define the strategy of MBSE implementation, one needs to think about additional benefits compared to a standard development process that are (simply just) possible because of the use of dedicated models and provide an reasonable argumentation for the upfront invest. Frontloading activities and especially Design Space exploration are such arguments.Thus, in order to make use of design space exploration techniques, the project has proven that there is a need for a clear and unambiguous syntax and semantics of the input models. A DSML can easily be defined in order to properly formalize constraints and objectives of the system.

Furthermore, precise system models in terms of a logical and technical architecture are required to serve as a proper system description. We identified and illustrated a multitude of quick-wins using such an integrated model-based approach, e.g. invalidity of the input model elements of the Rhapsody model, while importing these models into our framework. Such precise and unambiguous models (for requirements as well as the system architecture) has been successfully used as a basis for further frontloading activities.

The applicability of design space exploration techniques, namely the synthesis of mappings that are valid (and optimized) allocations of software parts to hardware units, could be successfully demonstrated (compare [**?**]). The usage of a state-of-theart satisfiability modulo theory (SMT), integrated into our DSE framework leads to highly appreciated results, w.r.t. calculation time.

One of the next steps is to apply our (DSE) approach to new problems in MbSE Systems engineering, that are connected to the presented *deployment synthesis* problem, namely the synthesis of technical architectures, answering questions like: *How many cores are needed for my next multi-core platform?* or *Which kind of communication bus variants is needed for my system design?*.

REFERENCES

[] 2011. *ISO 26262 - Road vehicles âĂŤ Functional safety.* Geneva, Switzerland.

[] Philippe Cuenot, Patrick Frey, Rolf Johansson, Henrik Lönn, Yiannis Papadopoulos, Mark-Oliver Reiser, Anders Sandberg, David Servat, Ramin Tavakoli Kolagari, Martin Törngren, and Matthias Weber. 2010. The EAST-ADL Architecture Description Language for Automotive Embedded Software. In *Proceedings of the 2007 International Dagstuhl Conference on Model-based Engineering of Embedded Real-time Systems (MBEERTS'07)*. Springer-Verlag, Berlin, Heidelberg, 297–307. http://dl.acm.org/citation.cfm?id=1927558.1927574

[] J. Eder, S. Zverlov, S. Voss, M. Khalil, and A. Ipatiov. 2017. Bringing DSE to Life: Exploring the Design Space of an Industrial Automotive Use Case. In *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, Vol. 00. 270–280. https://doi.org/10.1109/MODELS.2017.36

[] International Electrotechnical Commission 2010. *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems.* International Electrotechnical Commission, Geneva, Switzerland.

[] ISO/IEC/IEEE. 2011. Systems and software engineering – Architecture description. *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)* (1 2011), 1 –46. https://doi.org/10.1109/IEEESTD.2011.6129467

[] Klaus Pohl, Harald Hnninger, Reinhold Achatz, and Manfred Broy. 2012. *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology.* Springer Publishing Company, Incorporated.

[] Bernhard Schätz, Florian Hölzl, and Torbjoern Lundkvist. 2010. Design-Space Exploration through Constraint-Based Model-Transformations. In *Proceedings of the 2010 17th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*. 173–192.

[] Sebastian Voss, Vincent Aravantinos, Sabine Teufl, Florian ölzl, and Bernhard Schätz. 2015. AutoFOCUS 3: Tooling Concepts for Seamless, Model-based Development of Embedded Systems. In *8th International Workshop on Model-based Architecting of Cyber-Physical and Embedded Systems*.

[] Sebastian Voss and Bernhard Schätz. 2013. Deployment and Scheduling Synthesis for Mixed-Criticality Shared Memory Systems. In *IEEE ECBS International Conference*. IEEE.