An ISO 26262 Compliant Design Flow and Tool for Automotive Multicore Systems

Maria Trei[1], Salome Maro[2](✉), Jan-Philipp Steghöfer[2](✉), and Thomas Peikenkamp[1]

[1] OFFIS e.V., Eschwerweg 2, 26121 Oldenburg, Germany
{maria.trei,peikenkamp}@offis.de
[2] Chalmers | University of Gothenburg, Gothenburg, Sweden
{salome.maro,jan-philipp.steghofer}@gu.se

Abstract. Model-based design processes in the automotive industry must support standards like ISO 26262. Especially for smaller suppliers developing software for OEMs, large-scale methodologies like AUTOSAR are impractical. Instead, smaller, focused processes that still allow ISO 26262 compliance are required. In addition, the steps in the process must be well-supported by the development tool-chain, in particular when developing complex multicore systems. In this paper, we show such a process based on existing design flows and the current state of an automotive modelling tool. We structure the design flow to ensure compliance with the ISO 26262, where necessary complementing it with required steps to ensure safety. Furthermore, supporting tools extending the modelling tool are discussed. As a result, the presented design flow covers all development phases.

1 Introduction

With the development of new functions that are needed for new car generations—in particular in the context of autonomous driving functions—a massive performance increase of electric and electronic (E/E) systems is needed. The needed performance boosts are demonstrated not only by new car generations, but also several research projects—backed up by key industrial partners—that investigate how to use current multicore technologies. Projects like AMALTHEA4public and ARAMIS address concrete challenges imposed by exploiting multicore technology in model-based design processes.

These processes are very much focused on providing functional aspects of multicore systems. However, the domains in which the final system is deployed and the complexity of multicore development make it necessary to put a particular focus on safety aspects [12]. We address this need by presenting a safety-oriented design workflow, where state-of-the-art modelling and analysis concepts

The work has been partially funded by the German Ministry for Education and Research (BMBF) under the funding ID 01IS14029H (AMALTHEA4public) and ID 01IS15031H (ASSUME) and Vinnova AMALTHEA4public.

P. Abrahamsson et al. (Eds.): PROFES 2016, LNCS 10027, pp. 163–180, 2016.
DOI: 10.1007/978-3-319-49094-6_11

for the development of multicore systems are structured to support the fulfilment of ISO 26262 requirements. The underlying work was carried out in two phases that are also reflected in the structure of this paper: First, an analysis was carried out, identifying *design steps* applied in current industrial automotive processes and the *design concepts* involved in these design steps. The second phase was an analysis of the ISO 26262, identifying requirements that affect the execution of these design steps, including the relevant information to be included in the previously identified design concepts.

This paper is structured as follows: Sect. 2 introduces ISO 26262, method engineering, and related work. In Sect. 3, we discuss the AMALTHEA platform and design flow which is extended with concepts from ISO 26262 in Sect. 4. We conclude the paper with a summary and an outlook on future work.

2 Background and Related Work

The international standard ISO 26262 ("Road vehicles – Functional safety") focuses on safety issues in the development of E/E systems of passenger cars with a maximum gross vehicle mass up to 3500 kg. It describes the safety lifecycle of automotive E/E systems, including management, development, production, operation, service and decommissioning. A central element is the hazard analysis and risk assessment in the beginning of the safety lifecycle, where *Automotive Safety Integrity Levels (ASILs)* are defined. Based on this classification, requirements for avoidance of residual risk are defined and validation methods are recommended or prescribed. The procedure of how to develop the *system under development (SUD)* is given by a V-model, that divides engineering processes into phases of system, hardware, and software development based on the safety concept as shown in Fig. 1. This paper addresses the phases shown in bold, where the clauses highlighted with black boxes are already partially supported by the design flow. The organisation of requirements and their correspondence to elements of the SUD is prescribed to allow validation of the design and verification against the safety concept.

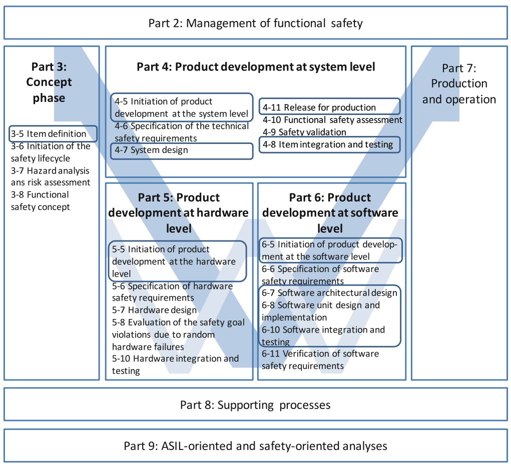
Part 2: Management of functional safety
Part 3: Concept phase
3-5 Item definition
3-6 Initiation of the safety lifecycle
3-7 Hazard analysis ans risk assessment
3-8 Functional safety concept
Part 4: Product development at system level
4-5 Initiation of product development at the system level
4-6 Specification of the technical safety requirements
4-7 System design
4-11 Release for production
4-10 Functional safety assessment
4-9 Safety validation
4-8 Item integration and testing
Part 7: Production and operation
Part 5: Product development at hardware level
5-5 Initiation of product development at the hardware level
5-6 Specification of hardware safety requirements
5-7 Hardware design
5-8 Evaluation of the safety goal violations due to random hardware failures
5-10 Hardware integration and testing
Part 6: Product development at software level
6-5 Initiation of product development at the software level
6-6 Specification of software safety requirements
6-7 Software architectural design
6-8 Software unit design and implementation
6-10 Software integration and testing
6-11 Verification of software safety requirements
Part 8: Supporting processes
Part 9: ASIL-oriented and safety-oriented analyses

Fig. 1. V-Model of ISO 26262

Since ISO 26262 mostly focuses on safety aspects of the development, additional method content must be provided for the functional part. Such adaptation can be done in a systematic fashion using approaches such as *situational method* *engineering* (SME) [9] or *process lines* [13]. The former approach uses standardised fragments to compose a process based on an analysis of the needs of a specific team and project. The latter extends SME by making the variability of the fragments explicit and using variability modelling primitives provided by process modelling tools. The need for tailoring ISO 26262 to a company's process guidelines has been acknowledged [8].

The authors of [7] model ASPICE, IEC 61508, and ISO 26262 in a process line with the aim to allow companies to derive processes conforming to one or more of these safety standards. Functional aspects are not modelled explicitly. Our work, in comparison, aims at combining a design flow for the functional aspects of the system with one for the safety aspects. We envision its use in the context of SME with specific adaptations for teams and projects.

The migration towards an ISO 26262 compliant process is the focus of [4]. Importantly, the authors identify the need to integrate the practices of the standard with the ones of existing processes at the companies. This is in line with what we have done by studying the existing process and deriving the superset of activities that are now combined with safety practices from ISO 26262.

3 The AMALTHEA Platform and Design Flow

AMALTHEA and AMALTHEA4public are ITEA funded projects running since 2011. They concentrate on software development for multi- and many core systems. One of their outcomes is the AMALTHEA platform [1,14] (APP4MC), a tool solution to aid the design of many and multi-core systems. The design flow has been developed in this context as the result of a survey of current development activities in the projects.

3.1 AMALTHEA Platform

Overview. The AMALTHEA platform is an open source tool platform developed mainly for engineering embedded systems in the automotive area. It supports many design activities as well as complex partitioning and mapping for embedded systems. Further, the platform assists engineering processes with the ability of handling product lines and executing simulation and validation tools. It is continuously extended to integrate more facets of system engineering processes such as verification, safety, and formal validation of timing requirements. Conformity to ISO 26262 is one topic to be addressed. A first step was the identification of correlations and needs between both the standard and the meta-model of the platform in form of a gap analysis [3]. The platform supports an iterative workflow to accommodate rapid prototyping and feedback from validation activities. Engineers are supported by the AMALTHEA platform in many steps ranging from component modelling to partitioning and mapping activities. The trace model, e.g., a result of the simulation of the system, provides helpful information about the execution times of specific system parts as a basis for the improvement of the existing architecture.

Data Models. APP4MC uses a top-level *System Model* that is subdivided into more fine-grained sub-models. They allow modelling based on the platform to derive the description of hard- and software used in the overall system down to the lowest level of abstraction. We will give a brief overview of the most important features.

The SUD can be represented from the system, software and hardware point of view. The *hardware model* is composed of (hardware) systems, ECUs, microcontrollers and cores. Networks and memories can be defined. Similarly, software can be subdivided into processes, tasks, runnables, and Interrupt Service Routines (ISRs) using the *software model*. The technical architecture in terms of components and their interfaces is part of the *components model*. Requirements on the behaviour of the system can be described by the *constraints model*, which consists of constraints on timing, data age and runnable sequencing or grouping, aspects particularly important in multicore development. This also includes requirements of the dynamical architecture, whereas interactions of software and hardware are covered by the *property constraints model*. The *partitioning model* describes how the software is broken down into *runnables* which are in turn aggregated in tasks [10]. The final allocation of software to hardware is given by the *mapping model*. The system model is completed by a *stimuli model*, *common model*, *event model*, *configuration model* and the *OS model*. The *trace model* contains information about the simulation/execution of the software on the hardware.

3.2 AMALTHEA Design Flow

Methodology. The design steps are the result of an extensive data collection process in which the project partners in AMALTHEA4public provided detailed information about the concrete steps they follow when developing multicore embedded software, especially in the automotive domain. Data was collected from five industrial partners and five academic partners. While the academic partners provided information mostly from their experience with different industrial partners they cooperate with in research projects, the industrial partners contributed with their concrete practical experience. The overall design steps, along with a rationale and additional information are published in [2]. The design steps are mainly concerned with development of functionality. Interestingly, safety aspects were not mentioned by any of the partners during the data collection.

Design Steps. Table 1 gives an overview of the design steps that are the result of the analysis. Note that no order in which these steps are carried out is implied since defining a concrete development process with a concrete lifecycle is generally company specific. A wide variety of lifecycles can be applied, including the V-model that is implied by ISO 26262 and AUTOSAR. While some steps are carried out sequentially, others can be done in parallel. The dependencies between the design steps at times imply an iterative approach where they are repeated or at least revisited after other work has been performed. Such an approach is common in iterative-incremental lifecycles. The identified steps cover most aspects of a traditional software development effort, starting from contract negotiation and scope identification and ending at the delivery of the software (with the exception of software maintenance). Some steps and circumstances are specific in the context of the automotive domain, e.g., the differentiation of system and software. Another re-occurring theme is product line issues and variants, even though this theme will not be regarded in detail in the context of this paper.

Table 1. Overview of the identified Design Steps

DS 1: System Requirements Engineering	**DS 7: Variant Configuration**
DS 2: System Architecture Design	**DS 8: Implementation**
DS 3: Software Requirements Engineering	**DS 9: Validation and Testing**
DS 4: Derivation of Product Variants	**DS 10: System Integration**
DS 5: Definition of Software Architecture	**DS 11: Handover**
DS 6: Behaviour Modelling	

Some of the design steps we elicited, such as Requirement Engineering and Architecture Design, can be found in nearly all development processes in a similar form. However, there are specific steps steps that are only relevant in multicore development: *Partitioning, Task Creation and Target Mapping* are, e.g., part of **DS 10: System Integration**. We discuss these design steps in more detail in Sect. 4.3.

4 Analysis of Compliance Towards ISO 26262

Based on the phases and clauses of ISO 26262 as well as the identified design steps, we analyse the recommended design flow to satisfy compliance with the standard and which support APP4MC provides. Further, we will show necessary extensions and the steps to achieve them. This will be presented constructively so as to show how the elicited design flow can be extended and how the platform capabilities tie into the flow.

The resulting extended design flow as shown in Fig. 2 represents the functional design steps outside the "V" and is enriched by six design steps related to safety inside the "V". The top of the given V-model stands for the vehicle/system-part of the V-model of ISO 26262 while the bottom represents the software part. As our focus lies on the design process derived for software development, we will only briefly relate to hardware development in the following subsections. However, many design steps, e.g., partitioning or definition of property constraints, require the formulation of hardware assumptions that are needed during software development.

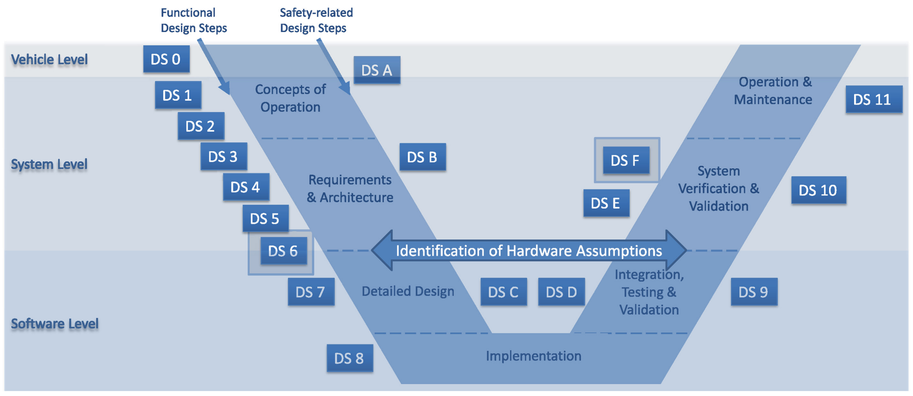

Fig. 2. Overview of the extended Design Steps

Briefly summarized, the new design flow lifts the development lifecycle up to the vehicle level, where in **DS 0** functional requirements are defined, which is important to initiate the *safety lifecycle* and therefore to support the concept phase (**DS A**) of ISO 26262. Safety requirements are introduced at system and software level (**DS B** and **DS C**). Integration and testing at software level for safety-related software elements is described by **DS D**, where the integration into the system and the item is given by **DS E**. **DS F** covers integration and validation activities at vehicle level. Since validation is against safety goals which are part of the concept phase at vehicle and item level, **DS F** can be seen as system and vehicle part. Before discussing our analysis of the different ISO 26262 phases, we first give an overview of common models that are used in all the development phases.

4.1 Generic Models Addressed in All Development Phases

ISO 26262 describes the development of an item based on the functional behaviour which is intended by each element of the item on the one hand, and the hazardous events concerning these elements which might lead to the violation of safety requirements on the other hand. As the different levels of abstraction during this development process lead to different features of safety requirements it is necessary to define the intended behaviour using concepts of certain levels of abstraction. Analogously, occurring errors of elements of the item have to be modelled at every level.

The overall functional system behaviour, as well as the specific aspects of the functional behaviour of the hardware and the software are captured in a *Behaviour Model*. The functional requirements on the vehicle level are broken down into respective parts for all levels and corresponding behaviour and thus defined on each level. Depending on the level, the specific models can take different forms such as algorithms, interaction protocols, state machines, etc. The model is captured in **DS 6: Behaviour Modelling** which is performed for each of the levels.

Further, it is necessary to introduce *modelling of errors* to analyse how faults of elements of the item affect their behaviour. On the modelling side, this requires a model that allows to perform safety activities related to the development of such a safety concept. To this end, we are using a generic error model that has been developed in the ITEA project SAFE [6] shown in Fig. 3.

Basically it captures failure behaviour, in particular how internal and external failures are propagated through the functions or components of the system. Once these propagations have been identified, *safety requirements* can then be derived and allocated to the architectural elements. Using this allocation scheme, the corresponding Automotive Safety Integrity Level (ASIL) can be derived for each of the allocated requirements. The ASIL describes the risk associated with each requirement based on the severity of a hazard, the likelihood of its occurrence, and how well the hazard can be controlled. This error model will be used at vehicle level to support the definition of the functional safety concept (ISO 26262-3:2011, 8), and at system level for definition and allocation of technical safety requirements (ISO 26262-4:2011, 6–7). Refinement of technical safety requirements to software and hardware safety requirements entails to also introduce error models at software and hardware level. Error models on system, hardware and software level have to include faults related to multicore scheduling, which are much more complex than faults occurring in a singlecore system. This is due to the fact that the scheduling does not only rely on the basis of timing and priorities of tasks, but also on the commonly shared memory available, temperature behaviour of the system and, going into the structure of the hardware a little deeper, shared buses or power supply.

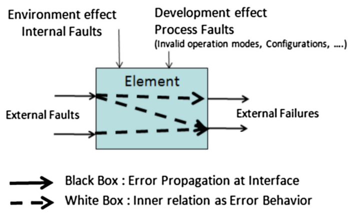
Environment effect
Internal Faults
Development effect
Process Faults
(Invalid operation modes, Configurations, ...)
Element
External Faults
External Failures
Black Box : Error Propagation at Interface
White Box : Inner relation as Error Behavior

Fig. 3. Error model from SAFE project

4.2 Concept Phase (ISO 26262 Part 3)

Processes provided by ISO 26262. The safety lifecycle given by the international standard ISO 26262 starts with the concept phase in ISO 26262-3:2011, which consists of the clauses given in the first column of Table 2.

Description of the SUD, called *item* in terms of ISO 26262, takes place at the vehicle level and includes dependencies on, and interaction with, other items and the environment of the item. Functional and non-functional requirements need to

Table 2. Concept phase

Clause	Supporting DS	Support by APP4MC
5: Item definition	**DS 0**, **DS A**, DS 6	(Property) Constraint Model, Component Model
6: Initiation of the safety lifecycle	**DS A**	ProR, Papyrus, Yakindu statecharts
7: Hazard analysis and risk assessment	**DS A**	Papyrus, Yakindu statecharts
8: Functional safety concept	**DS A**	ProR

be defined, as well as constraints given by the environment and other items. Note that at this point not only safety-related requirements are considered. Operating modes, interfaces and boundaries on them have to be specified to enable hazard analysis and risk assessment on the one hand and the development of the item with its intended behaviour on the other hand. Based on the item definition, the safety lifecycle is initiated in ISO 26262-3:2011, 6, which means that the development category of all parts of the item is analysed.

Processes during the hazard analysis and risk assessment of ISO 26262-3:2011, 7, systematically describe possible *hazardous events* and their consequences. They need to be classified using metrics for the *severity* of potential harm, the *probability* of exposure of operational situations, and the *controllability* of each hazardous event. As a result the *ASIL* can be calculated for each hazardous event. A *safety goal* is derived for each hazardous event, inheriting its ASIL. These safety goals serve as top-level safety requirements, from which further functional safety requirements will be derived, in particular those characterizing the *Functional safety concept*.

The definition of a functional safety concept requires to analyse how component faults can contribute to the identified risks. On the modelling side, this requires a model that allows to perform safety activities related to the development of such a safety concept (ISO 26262-3:2011, 8). To this end, we are using the generic error model introduced in the previous section and shown in Fig. 3. Basically it is able to capture failure behaviour, in particular how internal and external failures are propagated through the components of the system. Once these propagations have been identified, *functional safety requirements* can then be derived and allocated to the architecture according to ISO 26262-3:2011, 8.4.2 and 8.4.3. Using this allocation scheme, the corresponding ASIL can be derived for each of the allocated requirements.

Activities in the Design Flow. We extend the elicited design steps by introducing **DS 0: Functional Requirements Engineering**. In this step the functional requirements of the item at vehicle level will be collected. They provide a basis for the functional design of the system, software and hardware on the one hand and the initiation of the safety lifecycle on the other hand. As our design flow does not include the *development* of hardware, but has to represent all important information about the hardware available, if appropriate, it is recommended to define hardware properties already at this early stage of development. This can be, e.g., the intended number of available cores, the size of memory, and the interaction with other systems. The higher complexity of multicore systems compared to singlecore architectures may affect the exposure of certain hardware failures, which has to be considered both at system and at software level again. Further, the activities given by clauses 6–8 and discussed above need to be represented. For this, we introduce **DS A: Derivation of the Functional safety concept**, where the hazard analysis and the risk assessment take place to define the corresponding safety goals as top-level safety requirements.

Support by APP4MC. As shown in Table 2, APP4MC can only support the Item definition clause by allowing the definition of the system through the Component Model and constraints through the Property Constraint Model. Other open source eclipse-based requirements management tools such as ProR[1] can be used in the Item definition clause to describe the requirements of the item as text. Papyrus[2] and Yakindu statechart tools[3] can be used to describe the relationship between the item and the environment through component models. Description of the safety lifecycle, hazard analysis and risk assessment and functional safety concept can be done by text in ProR. However one needs to add safety related attributes such as *implementation categories, safety goals* and *hazardous events* to the requirements.

4.3 Product Development at the System Level (ISO 26262 Part 4)

Processes provided by ISO 26262. Development of the product on the system level defines how both hardware and software capabilities form the overall system. The division into seven clauses according to ISO 26262 is shown in Table 3. In clauses 5–7, requirements for the system to be developed as well as its overall structure are defined. Then, hardware and software development takes place (cf. Sects. 4.4 and 4.5), before clauses 8–11 manage the integration of hardware and software components into the overall system, further verification activities and, later on, the release for production.

Central work products of the former steps are the *Technical safety requirements specification* (ISO 26262-4:2011, 6.4) and the *Technical safety concept* (ISO 26262-4:2011, 7.4.) Both contain the set of safety-related requirements that are used to implement functional safety requirements. This hierarchical structure of requirements must be traceable back to the top-level safety goals and to the corresponding system elements. Basic information about the intended use of particular software and hardware resources shall be considered during these steps. Safety mechanisms qualified to prevent parts of the system from failing need to be described in terms of the underlying system architecture. Technical safety requirements inherit the ASIL of the corresponding functional safety requirements following the rules for ASIL decomposition given in ISO 26262-9:2011.

The *System design specification* and the *Hardware-software interface specification* (ISO 26262-4:2011, 7.4) consist of the overall system architecture and a first view on the separation into hardware and software. The system design shall be based on the functional concept, in particular not just the safety-related parts and the technical safety concept. If technical safety requirements are allocated to certain parts of the system design, their ASIL shall be attached to these parts respecting amongst others the criteria for coexistence given in ISO 26262-9:2011. Note that the hardware-software interface specification is going to be refined during hardware and software development.

[1] http://eclipse.org/rmf/pror/.
[2] https://eclipse.org/papyrus/.
[3] https://marketplace.eclipse.org/content/yakindu-statechart-tools.

Table 3. Phase 4: product development at the system level

Clause	Supporting DS	Supported by APP4MC
5: Initiation of product development at the system level	**DS B**	✗
6: Specification of the technical safety requirements	DS 1, **DS B**	(Property) Constraint Model, Component Model
7: System design	DS 2, DS 6	Component Model
8: Item integration and testing	DS 10	Mapping Model, (Property) Constraints Model, OS Model, Trace Model, Stimuli Model, Event Model, Configuration Model, Hardware Model
9: Safety validation	**DS E**	✗
10: Functional safety assessment	**DS F**	✗
11: Release for production	DS 11	✗

Other work products concerned with *planning* integration, verification and validation activities must also be considered. The relevance of verification even this early on during system development must be noted.

Execution of integration, verification and validation is part of ISO 26262-4:2011, 8–11. Integration takes place on different levels, where, roughly described, first hardware and software is integrated to systems, second systems are integrated to the item, and third the item is integrated to the vehicle. The compliance of the ASILs of different objects must be analysed, as well as the correctness of the functionality of the system design. Together, these steps form the work products *Integration testing specifications* and *Integration testing reports*. Clauses 9–10 describe in more detail which methods are recommended for certain verification and validation processes.

Activities in the Design Flow. The activities in the design flow on the system level are concerned with requirements engineering, overall system design, and system integration. We combine these aspects concerning the functional system with safety-specific aspects by introducing new, safety-focused activities.

DS 1: System Requirements Engineering is concerned with the elicitation and definition of system requirements as well as of platform and product requirements in case a product line approach is used. The system requirements can be captured in different ways and describe the system as a black-box, focusing mainly on what the system does (goals and scenarios), who the users are and what other systems will it interact with [5]. Model-based systems engineering methods provide a set of partial models to capture system requirements. Examples of the partial models are environment models which describe the SUD in its context, application scenarios which capture the different use cases of the SUD and requirements models which capture additional functional and non-functional requirements. The relevant artefacts created in this step are the *System Model (preliminary)* and the *System Requirements Specification Document*.

There is a strong correspondence between the described artefacts of the design flow and the *Technical Safety Requirements Specification* and the *Technical Safety Concept* of ISO 26262, even if safety is not yet regarded. System requirements with an impact on safety questions of the SUD should be treated separately from system requirements as system *safety* requirements with necessary attributes to correlate to technical safety requirements of ISO 26262. This includes amongst others the introduction of safety mechanisms and ASILs for these requirements. We therefore introduce **DS B: System Safety Requirements Engineering** which consists of at least the following sub-steps:

1. planning verification/validation activities at system level;
2. definition of system safety requirements based on results of **DS 1** and **DS A**, including safety-related assumptions caused by the multicore structure of the SUD.

DS 2: System Architecture Design addresses the design of the overall system architecture according to the elicited system requirements. This system architecture consists of several partial models that describe the system, sub-systems, their respective structure and behaviour, and the relations and interactions of the system with the environment. The relevant outcome of this step is the *System Architecture* that contains the architecture of the entire SUD. Along with the functional behaviour on the system level defined in **DS 6: Behaviour Modelling**, this complies in many points with the *system design specification* of ISO 26262. Accordingly, traceability of different system elements and underlying (safety) requirements should be enabled in this design step to ensure that safety-related sub-systems with their corresponding ASIL can be identified. Safety-related interference must be represented to allow validating if criteria for coexistence are met by objects defined during the system design. This includes, for instance, the problem of space and time partitioning, which describes the concurrency of resources and is therefore related to hardware and software development, but has to be regarded at system level. Therefore the behaviour modelling also relates to ISO 26262's *hardware/ software interface specification* as part of the system, software, and hardware design.

After the hardware and software development has been performed, the next relevant step is **DS 10: System Integration** in which executable tasks from the system are created, partitioned and mapped to the target hardware. This is where multicore development differs from singlecore development: multiple tasks can run at the same time and efficiency and consistency must be guaranteed. Therefore creation, partitioning and mapping of tasks as discussed below are essential activities.

Task Creation. In this step, tasks that contain a set of Runnables are created from the software model. Task creation also takes into account the constraint model in order to decide which Runnables can be grouped together. The results of this step are stored in an augmented *Software Model*. At this point, the rate at which the tasks are activated, e.g., periodic, single, or sporadic activation is determined and stored in a *Stimulation model*.

Partitioning. In this step, tasks are identified to derive possible partitions that can be executed in parallel. Partitioning includes possibilities to group Runnables by their activation reference and group independent sets of Runnables to come up with graph structures that have the most efficient potential when running tasks in parallel. This step leads to a *Partitioned Software Mode*l and a *Constraint Model*.

Target Mapping. The aim of this step is to find a valid and optimal distribution of software elements to hardware components. Data from *Software and Hardware Models*, as well as the tasks activation from the *Stimulation Mode*l are used to calculate such a distribution. Additionally, a *Property Constraints Model* may be included during the mapping process which is used to narrow down the solution space, e.g., some tasks may require the target platform to have a certain amount of memory. The results of this step are stored in a *Mapping Model*. Moreover, a preliminary *OS Model* is generated, which contains a scheduler for each of the cores of the hardware platform.

The artefacts of **DS 10** correspond to inputs of the *hardware-software interface specification* (HSI) of the ISO 26262 system level, which is already supported by **DS B**. But the elicited design steps do not contain explicit support for verification of safety system requirements after the integration. This will be corrected by introduction of **DS E: Safety Validation** and **DS F: Functional Safety Assessment**. **DS E** consists of testing activities to verify that the safety requirements are satisfied and **DS F** assesses the functional safety concept defined in **DS A** and its implementation.

Finally, **Design Step 11: Handover** handles acceptance testing, delivery of the product and sign-off. All acceptance tests must pass before the product can be delivered to the customer. Including safety-related tests, this corresponds in many points to the *Release for production* given by ISO 26262-4:2011, 11.

Support by APP4MC. APP4MC provides a *Hardware Model* which is dedicated model that supports the mapping process. The model allows for definition of the available hardware by specifying the number of cores, the speed, memory and other hardware related properties [11]. This model can then be used together with the software model and the constraint model to create an optimal distribution of the software to a specific hardware platform. The property constraints model also offers the possibility to define safety-related constraints, e.g., to separate safety-critical from non-safety-critical software running on different hardware components. This could be necessary in a development process to guarantee freedom of interference of certain artefacts. Another use case of separation related to safety is the principle of redundancy, which means that some safety-critical parts of a system are implemented twice to prevent the single-point-failure for a certain part.

4.4 Product Development at the Hardware Level (ISO 26262 Part 5)

Processes provided by ISO 26262. Due to space constraints, we restrict development activities on the hardware level to the derivation of hardware safety requirements (ISO 26262-5:2011, 6) as shown in Table 4 and do not dive into the hardware design or its evaluation (ISO 26262-5:2011, 7–9). We cannot drop this clause since hardware safety requirements have to be consistent with the technical safety concept and the system design specification (ISO 26262-5:2011, 6.1), imposing non-trivial relationships between the hardware and system level on one side, and between hardware and software level on the other side. The hardware elements relevant for the hardware-software interface have already been identified as shown in Sect. 4.3.

Activities in the Design Flow. Description of required hardware and its components can be captured in the *Variant Model* and the *Hardware Model* derived during **DS 4: Derivation of Product Variants** and refined in **DS 6: Behaviour Model**. An important aspects to be considered when designing a multicore systems is the distribution of safety-related software to certain hardware elements such as ECUs, cores, or memory. Depending on such a classification, hardware can then also be categorised as safety-related or not which affects to what extent testing activities are required. This again emphasizes why traceability of all kinds of information is very important in the development of safety-critical systems. The elicited design flow did not contain an explicit step for definition of hardware safety requirements. Therefore we suggest that this is addressed as early as in **DS B: System Safety Requirements Engineering**. When system safety requirements are defined, hardware safety requirements should be defined as well. These can later be refined when the actual design and development of the hardware takes place.

Support by APP4MC. Table 4 shows the tools available in APP4MC, that can be used to support clause (ISO 26262-5:2011, 6). All necessary elements for a description of the hardware-software interface are available in the Hardware model, including descriptions for *ECU*, *Microcontoller*, and *Core* and their associated memory access and communication characteristics. Since hardware safety requirements have to address the effectiveness of safety mechanisms (ISO 26262-5:2011, 6.4.2), an error model used to represent hardware failures is required for a precise description of these. Although it has the same role as the error model on concept level shown in Fig. 3, this model has to be rich enough to allow for the (later) evaluation, integration and testing activities (ISO 26262-5:2011, 7–10). Due to this detail in the hardware safety requirements it is possible to evaluate the impact of hardware failures on system level (ISO 26262-5:2011, 7.4.4) even before the hardware is available. The support by APP4MC for the definition of hardware safety requirements is given by the usage of ProR on the one hand and the Property Constraints Model on the other hand, where requirements and constraints on hardware can be defined.

4.5 Product Development at the Software Level (ISO 26262 Part 6)

Processes provided by ISO 26262. ISO 26262 provides the clauses shown in Table 5. *Software safety requirements* based on the technical safety concept specified at system level need to be defined. The HSI is updated to accord with these

Table 4. Product development at the hardware level

Clause	Supporting DS	Support by APP4MC
6: Specification of hardware safety requirements	**DS B**	Hardware Model, (Property) Constraint Model, ProR

requirements before regarding a first version of the overall software architecture in ISO 26262-6:2011, 7. The *Software architectural design specification* describes *software components* and their interactions, i.e., their hierarchical structure and interfaces, and properties that influence their implementation such as scheduling properties (ISO 26262-6:2011, 7.4). Based on this, the *Software unit design specification* is derived (ISO 26262-6:2011, 8.4). The next step is to generate source code, which is followed by verification and testing activities (ISO 26262-6:2011, 8–9). The goal of these activities is to ensure that the implementation satisfies requirements on the software units. These requirements are not limited to safety-related parts of the software, even if testing activities in ISO 26262 only focus on them. Integration of the software elements to the embedded software in ISO 26262-6:2011, 10, consists of regarding the implementation results with respect to the software architectural design specification, whereas satisfaction of software safety requirements is part of ISO 26262-6:2011, 11. Relations concerning the hardware-software interface must be considered.

Table 5. Phase 6: product development at the software level

Clause	Supporting DS	Supported by APP4MC
5: Initiation of product development at the software level	**DS C**	✗
6: Specification of software safety requirements	DS 3, DS 4, **DS C**	Software Model, (Property) Constraint Model
7: Software architectural design specification	DS 5, DS 6, DS 7	Software Model, Component Model
8: Software unit design and implementation	DS 8	Software Model
9: Software unit testing	DS 9	Trace Model
10: Software integration and testing	DS 9	✗
11: Verification of software safety requirements	**DS D**	✗

Activities in the Design Flow. The activities in the functional design flow cover all aspects of the software development lifecycle, starting from requirements, to analysis and design, to coding, and validation. **DS 3: Software Requirements Engineering** addresses the elicitation of requirements pertaining to the software part of the system. Sources for requirements are system requirements and customers or potential users of the software. Software units shall be described so that a partitioning can be executed to prepare the software for mapping to the hardware. For multicore development, it is very important to describe any kind of requirements on relations between software units, such as sequencing and dependent deadlines. This step is usually done in several iterations. The collected requirements are recorded in the *Software Requirements Specification Document*. *Acceptance Tests* are defined based on the requirements to validate that the agreed-upon system is being built.

As for the system level, requirements engineering at the software level needs to be extended with respect to safety to comply with ISO 26262. We introduce **DS C: Software Safety Requirements Engineering** which, similar to **DS B**, consists of

1. planning verification/ validation activities at software level,
2. the definition of software safety requirements based on **DS B** (and the system design), and the
3. validation of the software safety requirements against the software requirements (i.e., the part that is not safety-related).

Also, it must be ensured that the software safety requirements are correct, complete, and consistent with respect to the safety goals and the system design.

DS 5: Definition of Software Architecture can produce design artefacts such as component models which contain all the software components and their dependencies and interaction models to describe communication between these components. The architecture of a software can be described using more than one model in order to capture different perspectives of the software or to further refine it into a lower abstraction level. All models are captured in the *Software Architecture Document*. The development of software is further supported by **DS 6: Behaviour Modelling**, where the behaviour of software components can be specified, such as the communication between components. An iterative review process ensures consistency and completeness of the architecture.

In **Design Step 8: Implementation**, the required code is produced, tests are developed and executed, the software is integrated and the code is reviewed. The main resulting artefacts are *Source Code* and different sets of *Tests* (unit, component, integration) as well as the *Integrated Software and its Documentation*.

The final activity in this phase is **Design Step 9: Validation and Testing**. It involves testing of software components to validate if they are working as desired, i.e., according to the specified requirements. For software components that will interact with hardware components, simulations are run in order to fix as much defects as possible before the component can be tested on the actual hardware. *Deployable Control Software* is a packaged integrated software that is ready to be deployed on a specific hardware.

In parallel testing methods given by ISO 26262 should be introduced to support the verification of the embedded software against the software safety requirements. Depending on the ASIL of each safety goal, different technologies and testing environments are feasible, as e.g. fault injection tests or interface tests. We define **DS D: Verification of software safety requirements** to be the design step supporting these activities.

Support by APP4MC. The platform provides the Software Model that can be used to model the runnables, tasks and processes that make up the software. For software architecture specification, the Component Model can be used, as well as the Property Constraint Model to define software constraints. Additionally, the platform provides a Trace model to assist software testing. The trace model gives details on time consumed by tasks to allow refinement of the model to get the most efficient one.

4.6 Tool Support

As can be seen in the tables that show tool support for the design steps in APP4MC, mainly parts introduced by the clauses related to safety assessment and verification in ISO 26262:2011 are missing natively. Therefore, it is necessary to accompany APP4MC with one or more additional tools that can fill those gaps. Where open source eclipse tools are available(e.g., ProR and Papyrus), we have proposed their use in combination with APP4MC. However, not all phased can be supported by open source tools, for instance for functional safety activities a commercial tool like medini analyse[4] which supports Item Definition (ISO 26262-3:2011, 5; **DS 0, DS A, DS 6**), Initiation of the Safety Lifecycle (ISO 26262-3:2011, 6; **DS A**), Hazard Analysis and Risk Assessment (ISO 26262-3:2011, 7; **DS A**), and Functional Safety Concept (ISO 26262-3:2011, 8; **DS A**) may be used. This covers the entire concept phase. Validation and assessment capabilities, supporting Safety Validation (ISO 26262-4:2011, 9; **DS E**), Functional Safety Assessment (ISO 26262-4:2011, 10; **DS F**), and Verification of Software Safety Requirements (ISO 26262-6:2011, 11; **DS D**), can be provided by, e.g., BTC EmbeddedValidator[5].

In addition, APP4MC interfaces with a number of requirements management and modelling tools that are commonly used in the industry. Some like IBM Rational DOORS are integrated through the use of OSLC adapters or other means, while many such as the different modelling environments integrate

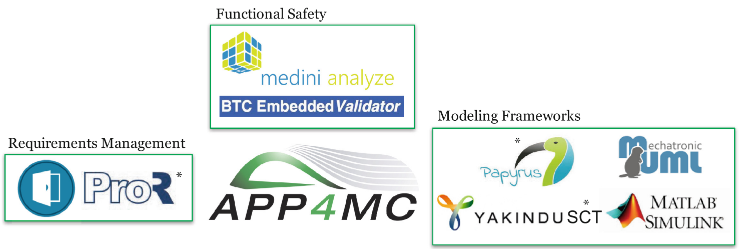

Fig. 4. Tools providing support for clauses not covered directly by APP4MC. The star indicates which tools are shipped with the distribution of the platform.

[4] http://www.kpit.com/engineering/products/medini-functional-safety-tool.
[5] www.btc-es.de/index.php?idcatside=40&lang=2.

seamlessly into the Eclipse environment provided by APP4MC. If a direct integration or the use of standards like OSLC is not feasible, the export and import capabilities of APP4MC and the tools must be used. This potentially introduces synchronisation issues, however. For design steps that are not repeated very often and occur towards the end of the development cycle—such as **DS D: Verification of software safety requirements**—this problem is negligible since exported artefacts do not need to be synchronised with the tools where they were originally created. An overview of the tools that are currently in use to support development with APP4MC is shown in Fig. 4.

5 Summary

An efficient way of working in an ISO 26262 compliant fashion with effective tool support is vital to maintain the relationship between OEMs and their suppliers. In this paper we have shown how a design flow elicited from actual development practices at such companies can be extended for ISO 26262 compliance and how it is supported by APP4MC. Where necessary (e.g., concept phase), missing modelling concepts (e.g., error models) have been identified. Other tools (both open source and commercial) that can be used in combination with the APP4MC to support safety activities in the ISO 26262 have also been suggested.

Future work will include the deployment and validation of the design flow in the companies that are part of AMALTHEA4public. In addition, we aim to strengthen the interface between systems and software engineering by refining the design steps that regard the exchange of information between these levels. Issues of traceability and cross-company information exchange will also be regarded.

We proposed extensions of the platform either through provision of own tools or recommendation of tools compatible with APP4MC to support the additional design steps. However, the benefit of using other external tools needs to be analysed, especially concerning our safety extensions for which commercial tools exist that allow ISO 26262 compliant design in some of the defined steps. But even commercial tools do not allow to follow our design flow in all aspects, so we have to investigate which design steps are supported by external tools, and how they can get integrated with APP4MC. Even if some tools offer interfaces via OSLC, there is still the need for rich traceability between different tools to support the exchange of information between different companies as well as change management. This leads to the implementation of tool adapters and traceability tools across the whole AMALTHEA toolchain.

References

1. Amalthea Project. http://www.amalthea-project.org/. Online; Accessed 16 Mar 2007
2. Amalthea4Public Project. D1.1: Analysis of Necessary Design Steps. Technical report, ITEA (2015). https://itea3.org/project/workpackage/document/download/2347/13017-AMALTHEA4public-WP-1-D11:AnalysisofNecessaryDesignSteps.pdf
3. Amalthea4Public Project. D4.1: Gap analysis against ISO 26262. Technical report, ITEA (2015). https://itea3.org/project/workpackage/document/download/2232/13017-AMALTHEA4public-WP-4-13017-AMALTHEA4public-WP-4-d41GapanalysisagainstISO26262.pdf
4. Born, M., Favaro, J., Kath, O., Application of ISO DIS 26262 in practice. In: 1st Workshop on Critical Automotive Applications: Robustness & Safety, pp. 3–6. ACM (2010)
5. Braun, P., Broy, M., Houdek, F., Kirchmayr, M., Müuller, M., Penzenstadler, B., Pohl, K., Weyer, T.: Guiding requirements engineering for software-intensive embedded systems in the automotive industry. Comput. Sci. Res. Dev. **9**(1), 21–43 (2014)
6. Cuenot, P., Peikenkamp, T., Wenzel, T., Khalil, M., Rudolph, A., Lucas, J., Voget, S., Ross, H., Eckel, A., Biendl, E., Adler, N., Otten, S., Buch, S.: Methodology and application rules documentation. Technical report, ITEA (2014). https://itea3.org/project/workpackage/document/download/1629/10039-SAFE-WP-6-SAFED6b.pdf
7. Gallina, B., Kashiyarandi, S., Martin, H., Bramberger, R.: Modeling a safety- and automotive-oriented process line to enable reuse and flexible process derivation. In: COMPSACW, pp. 504–509, July 2014
8. Hamann, R., Sauler, J., Kriso, S., Grote, W., Mössinger, J.: Application of ISO 26262 in distributed development ISO 26262 in reality. Technical report, SAE Technical Paper (2009)
9. Henderson-Sellers, B., Ralyté, J.: Situational method engineering: State-of-the-art review. J. Univ. Comput. Sci. **16**(3), 424–478 (2010)
10. Höttger, R., Krawczyk, L., Igel, B.: Model-based automotive partitioning and mapping for embedded multicore systems. Int. J. Comput. Control, Quantum Inf. Eng. **9**(1), 268–274 (2015)
11. Krawczyk, L., Kamsties, E.: Hardware models for automated partitioning and mapping in multi-core systems using mathematical algorithms. Int. J. Comput. **12**(4), 340–347 (2014)
12. Parkinson, P.: Safety, security and multicore. In: Dale, C., Anderson, T. (eds.) Advances in Systems Safety, pp. 215–232. Springer, London (2011)
13. Ternité, T.: Process lines: a product line approach designed for process model development. In: SEAA 2009, pp. 173–180. IEEE (2009)
14. Wolff, C., Krawczyk, L., et al.: Amalthea - tailoring tools to projects in automotive software development. In: IDAACS, vol. 2, pp. 515–520, September 2015