Integrated Systems Engineering and Software Requirements Engineering for Technical Systems

Jörg Holtmann, Ruslan Bernijazov,
Matthias Meyer, David Schmelter, Christian Tschirner
Project Group Mechatronic Systems Design, Fraunhofer Institute for Production Technology IPT
Zukunftsmeile 1, 33102 Paderborn, Germany
[joerg.holtmann | ruslan.bernijazov |
matthias.meyer | david.schmelter | christian.tschirner]@ipt.fraunhofer.de

ABSTRACT

The development of software-intensive technical systems (e.g., within the automotive industry) involves several engineering disciplines like mechanical, electrical, control, and software engineering. Model-based Systems Engineering (MBSE) coordinates these disciplines throughout the development by means of discipline-spanning processes and system models. Such a system model provides a common understanding of the system under development and serves as a starting point for the discipline-specific development. An integral part of MBSE is the requirements engineering on the system level. However, for the discipline-specific development to start, these requirements need to be refined, e.g., into specific requirements for the embedded software. Since existing MBSE approaches lack support for this refinement step, we conceived a systematic transition from MBSE to model-based software requirements engineering, which we present in this paper. We automated the steps of the transition where possible, in order to avoid error-prone and time-consuming manual tasks. We illustrate the approach with an example of an automotive embedded system.

Categories and Subject Descriptors

D.2.1 [**SOFTWARE ENGINEERING**]: Requirements/Specifications—*Methodologies*

Keywords

Systems Engineering, Requirements Engineering

1. INTRODUCTION

The development process of software-intensive technical systems (i.e., embedded, mechatronic, and cyber-physical systems) involves several engineering disciplines. Each of these disciplines applies its dedicated design methods and languages. This implicates the need for a holistic and interdisciplinary consideration of the overall system to obtain a common understanding of the system under development (SUD) for all roles involved in the development.

"Systems Engineering is an interdisciplinary approach and means to enable the realization of successful systems" [23] and aims at achieving this common understanding. According to [24], the current transition from document-based Systems Engineering to Model-Based Systems Engineering (MBSE) will make Systems Engineering the future development paradigm and aspires to overcome the disadvantages of the traditional "throw-it-over-the-wall" development process [35]. The abstraction of the real system to a discipline-spanning system model not only supports a holistic understanding of the SUD for a "First Time Quality Development" but also enables the traceability of all relevant information from customer requirements to the final system as well as design space exploration in early design phases based on (semi-)automatic analyses.

A high quality MBSE requirements engineering is crucial since the subsequent development relies on it, and defects in the requirements specification are hard and thus costly to fix later (e.g., [6]). These requirements need to be refined, e.g., into specific requirements for the embedded software, to start the discipline-specific development. Development process standards for technical systems like Automotive SPICE [3] explicitly distinguish between requirements engineering at system and software level.

On the one hand, the embedded software of a technical system comprises feedback controllers, e.g., electronic stability control, which continuously controls the dynamic behavior of the physical parts of the systems. This *control behavior* is specified in the discipline of control engineering by means of continuous models (based on differential equations). On the other hand, a growing part of the software is dedicated to the message-based coordination of systems (e.g., car-2-car communication). This *coordination behavior* is specified by means of discrete, state-based, and event-triggered models [33]. We regard the discipline of software engineering as the one that designs this coordination behavior. The requirements engineering for software is regarded as a branch of systems engineering [26]. More specifically, we regard software requirements engineering (SWRE) as a sub-discipline of software engineering, i.e., as requirements engineering for the coordination behavior part of the overall system.

We focus on the requirements engineering core activities *documentation* and *validation* [32]. Similar to the advantages of applying models in MBSE, applying models in requirements engineering supports the requirements validation by fostering automatic analysis techniques. A well suited no-

ICSSP'15, August 24–26, 2015, Tallinn, Estonia
ACM. 978-1-4503-3346-7/15/08...
http://dx.doi.org/10.1145/2785592.2785597

tation for the requirements documentation in model-based SWRE are scenario-based formalisms. Scenarios describe sequences of events of tasks that the SUD has to accomplish [20]. Scenario-based notations have an intuitive representation [20] and improve the comprehension of functional requirements for people experienced in modeling [1].

In previous work, we developed a formal SWRE approach based on a recent Live Sequence Chart (LSC) [12] variant compliant to the Unified Modeling Language (UML) [28], so-called *Modal Sequence Diagrams* (MSDs) [18]. This scenario-based SWRE approach allows to validate the requirements by means of simulation, i.e., the play-out algorithm originally conceived for LSCs [19]. Furthermore, the approach enables to formally verify the requirements for consistency and realizability by synthesizing global controllers from the scenarios [16]. These automatic analysis techniques enable the early detection of unintended behavior and inconsistencies between scenarios on requirements level. Our MSD approach considers assumptions on the environment [10] as well as real-time requirements [9] and is applicable to hierarchical component architectures [22], which makes it well suited in the context of technical systems.

Despite MBSE endeavors to coordinate the overall development by means of an interdisciplinary system model, the transition from MBSE to SWRE is not trivial. The system model contains much information that is only partly relevant to all involved disciplines. Thus, the software requirements engineer must carefully identify the information in the system model that is relevant to SWRE and transfer this information into a MSD specification. Despite the system model is amenable for automatisms to extract this information, this is a manual and thus time-consuming and error-prone task.

We apply the specification technique Consens (CONceptual design Specification technique for the Engineering of complex Systems) [15, Sect. 4.1] as a concrete MBSE approach. Consens has been conceived for the interdisciplinary development of complex technical systems, comprises a semi-formal modeling language and defines a method for the compilation of the system model in the concept phase.

Based on Consens, two approaches exist for the transition from MBSE to software engineering: Heinzemann et al. [21] present a systematic development process for the software comprising control and coordination behavior. This process utilizes automatisms to derive initial models. However, the approach aims at deriving and refining design models and not at the transition to discipline-specific requirements engineering. Anacker et al. [2] present a systematic process for the transition from MBSE to the software design via MSD specifications. However, this approach relies on reusable solution patterns, i.e., there must be prior projects in which these patterns have been successfully developed.

In order to facilitate the transition from MBSE with Consens to SWRE with MSDs, we introduce a systematic process that complements the transition approaches mentioned above. We identify aspects of the system model that are relevant to SWRE and describe how SWRE integrates into MBSE. In order to avoid error-prone and time-consuming manual tasks, we automated steps of the transition where possible. We illustrate the approach with an example of an automotive embedded system.

In the next section, we present the transition process. Sect. 3 investigates related work. We summarize this paper and provide an outlook on the future work in Sect. 4.

2. INTEGRATION OF MBSE AND SWRE

Systems Engineering has a long tradition. However, due to the various industries the systems engineers work for and their distinct tasks, its job profile is not clearly defined up to now [36]. The lowest common denominator about the job profile is: The systems engineer is the "conductor" of a project, orchestrating all technical tasks [34]. Based on this, Sheard [36] identified twelve possible roles of systems engineers.

These roles contain two views on the systems engineer: the *lifecycle view* and the *management view* [37]. In the lifecycle view, the systems engineer acts more "internally" in the product lifecycle responsible for the technical tasks, whereas the management view is oriented to the management and the external customer. Here, the following three roles of the lifecycle view are important (cf. [36]):

Requirements Owner (RO) The RO starts with the definition of technical requirements on system level (system requirements) based on given customer requirements. An example is the speed of a car: the customer demands a distinct maximum speed leading to a system requirement on the number of valves for the motor.

System Designer (SD) Based on the system requirements, the SD creates the high-level, discipline-spanning system architecture. This usually means the definition of system functions, the selection of adequate top-level components and their allocation to engineering disciplines.

System Analyst (SA) The SA ensures that the SUD meets the system requirements. Typical analyses are "system weight or throughput" for hardware and "memory usage or response times" for software. Usually, these analyses can only be executed at a very late lifecycle stage. The advent of MBSE enables early, abstract simulations on system level. This helps reducing uncertainty and risk in the early stages of the project, thus avoiding time- and cost-intensive iterations in the later stages. Since it is impossible model all relevant aspects of the system for this role, his task is rather to provide an optimal starting point for the discipline-specific issues.

Besides these lifecycle roles, the *Customer Interface (CI)* as part of the management view is essential for a successful project. CI is the "face to the customer" and responsible for delivering the right customer requirements.

Within the software engineering discipline, we focus on the role *Software Requirements Engineer* responsible to document and validate the requirements on the coordination behavior and negotiate them with the SA.

Fig. 1 depicts our idealized overall process for the systematic integration of SWRE with MSDs into MBSE with Consens. The process is specified by means of the Business Process Model and Notation (BPMN) [31]. We allocate the particular tasks to the roles mentioned before to clarify the distribution of responsibilities. Whereas the coarse-grained roles Systems Engineer and Software Engineer are specified by means of BPMN pools, the fine-grained, specialized roles introduced above are specified by by means of BPMN lanes.

The main contribution of this paper is emphasized in Fig. 1 with gray tasks and artifacts. We visualize manual steps by

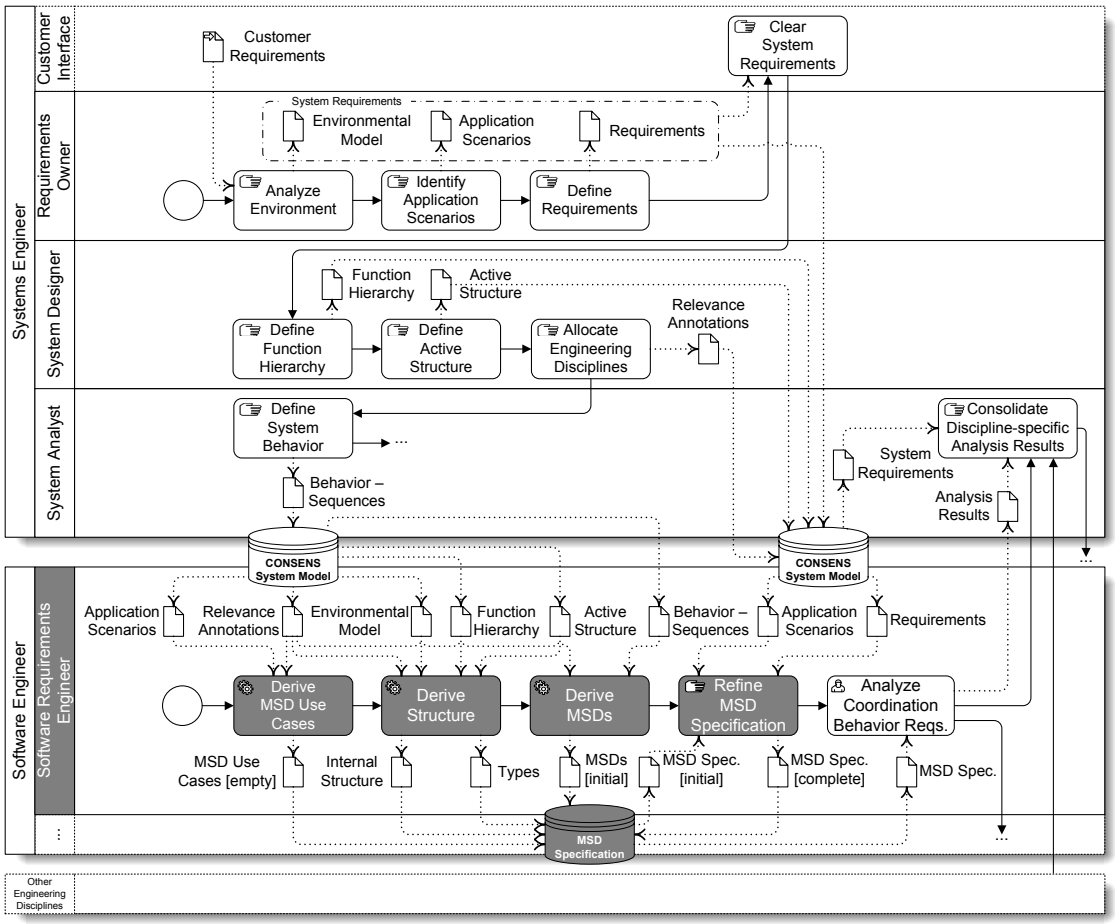

Figure 1: Idealized overall process for the integration of MBSE with CONSENS and SWRE with MSDs

means of BPMN manual tasks (hand in the upper left corner of the task). Steps that we could automate are visualized by means of BPMN service tasks (cogwheel in the upper left corner of the task). The step that is tool-supported is specified by means of a BPMN user task (person in the upper left corner of the task). Work results are specified as BPMN data objects (document icons), and persistent models that are subject to update and retrieval operations are specified as BPMN data stores (database icon). Multiple occurrences of the same data store represent a BPMN data store reference.

Note that despite the roles in Fig. 1 might indicate a strict separation of responsibilities, we encourage the involvement of all disciplines in the specification of the system requirements and design since issues resulting from a strict separation of responsibilities can cause major problems [7]. Furthermore, we assume that the Systems Engineer has a "T-shaped" competency profile [17] with deep knowledge in one discipline as well as basic knowledge in the remaining disciplines. This enables the Systems Engineer to understand and respect the needs of all discipline experts like the Software Engineer.

In the following, we exemplarily perform and explain each of the process steps in Fig. 1. For illustration purposes, we use a simplified automotive car access system as a running example for the SUD. This system comprises a door lock electronic control unit (ECU) and a part of the functionality of a Body Control Module (BCM). A BCM centrally controls distributed vehicle body functions like central locking, exterior lights, anti-theft warning system, etc. We focus on the functionality of central locking and speed locking.

2.1 MBSE with CONSENS

The CONSENS modeling language is divided into eight partial models describing different aspects of a technical system: *Environmental Model*, *Application Scenarios*, *Requirements*, *Functions*, *Active Structure*, *Shape*, *System of Objectives*, and *Behavior*. Behavior can be subdivided into *Behavior – States*, *Behavior – Activities*, and *Behavior – Sequences*. We screened the CONSENS language and identified six partial models that contain information relevant to SWRE. The partial model *Shape* describing the geometrical appearance of the SUD and the other behavior models except *Behavior – Sequences* are of no interest for the addressed coordination behavior. Furthermore, in our experience *System of Objectives* is rarely applied in practice.

All aspects specified with the CONSENS partial models are strongly interconnected. These interconnections are explicitly specified as cross-references between elements of partial models, e.g., bidirectional trace links between requirements and functions. Hereby, traceability can be established and maintained to enable impact analyses.

The process step order for the specification of the particular CONSENS partial models in the systems engineering part of Fig. 1 is based on the method presented in [15, Sect. 3.2]. In the following, we exemplarily perform each of these process steps for the development of our running car access system

example. Fig. 2 depicts excerpts of all SWRE-relevant partial models specified in CONSENS.

2.1.1 Analyze Environment

The *Environmental Model* is a structural partial model and considers the SUD as a black box embedded into its environment such that relevant spheres of influence are identified.

The CI and RO analyze the environment and jointly identify all relevant elements that influence the SUD. The CI represents the market issues (e.g., external stakeholders like customers), whereas the RO represents all technical issues and is responsible for the correct usage of the method.

System and environment elements are physical elements like parts, assemblies, or modules but can also refer to non-physical elements like software components. Different kinds of flows (i.e., material flows, energy flows, and information flows) represent the relationships between the elements. The flows in the system model connect ports of the system or environment elements. These ports are specified by means of port specifications[1]. A port specification defines which particular material, energy, or information, respectively, can be transferred over the port.

The *Environmental Model* in the top left of Fig. 2 embeds the SUDs BCM and Door as part of the overall car access system into their environment. In our example, this environment encompasses other ECUs. Since these ECUs communicate by means of signals, there are only information flows on this abstraction level. For example, the BCM can receive central locking requests via the information flow central(Un)LockingReq from the RemoteKey and the current velocity via v_current from SpeedSensor.

2.1.2 Identify Application Scenarios

Application scenarios are initial assumptions of the system's behavior. They describe the most common operation modes of the system and the corresponding behavior in a rough manner. Every application scenario describes a specific technical situation and the required behavior of the system.

Again, the RO is responsible for this task, but is supported by other stakeholders, e.g., from manufacturing and especially validation and verification. He has to consolidate all application scenarios and evaluate their significance.

The application scenario Remote-based Locking (top right in Fig. 2) describes the trigger situation as well as the intended behavior for the overall remote locking functionality including sketches. For example, it specifies that after a successful door locking operation the turn signals have to be activated to provide feedback to the passengers. A second application scenario Speed-based Locking is indicated, which describes the behavior that all doors have to be locked when a certain speed threshold is exceeded after car start.

The environment elements have affects trace links to application scenarios in which they are involved. For example, the environment element SpeedSensor is not involved in the application scenario Remote-based Locking.

2.1.3 Define Requirements

Based on the partial models *Environmental Model* and *Application Scenarios*, the RO defines, specifies, and manages the requirements. The partial model *Requirements* contains an organized collection of requirements that need to be fulfilled by the SUD. Requirements allow to expose what is expected from the future system. They form a milestone for the validation and verification in the subsequent development phases.

The *Requirements* model (center left in Fig. 2) refines and breaks down the application scenarios into particular, individually testable requirements. For example, requirement 5.6 specifies that the turn signal feedback operation shall be activated within 200 milliseconds.

All requirements with ID 5.x have refines trace links to the application scenario Remote-based Locking, and all requirements with ID 6.x have links to Speed-based Locking (not depicted in Fig. 2). These trace links enable that all requirements refining an application scenario can be identified from it and vice versa.

2.1.4 Clear System Requirements

We regard the *Environmental Model*, *Application Scenarios*, and *Requirements* altogether as System Requirements (cf. BPMN group in Fig. 1), which propose a technically oriented solution of the SUD functionality that is described by the Customer Requirements (cf. BPMN data input in Fig. 1). These system requirements have to be cleared by the CI with the customer before the actual system design starts.

2.1.5 Define Function Hierarchy

A function is the general and required coherence between input and output parameters, aiming at fulfilling a task. Functions are realized by solution patterns and their concretizations. Starting with the overall function, a break-down into sub functions takes place until useful technical solutions can be found for the functions.

In contrast to [36], we expect the SD to be responsible for defining the function hierarchy. The SD knows best the technologies available at the company or at suppliers for realizing the functions. Furthermore, he performs the function hierarchy definition together with the RO.

In our example, the function hierarchy (center right in Fig. 2) decomposes the overall locking functionality Door Locking into the separate functions Remote Locking and Speed Locking.

The traceability between application scenarios and functions is established via induces trace links. For example, the function Remote Locking is induced from the application scenario Remote-based Locking, and Speed Locking is induced by Speed-based Locking.

2.1.6 Define Active Structure

By means of the *Active Structure*, the SUD is considered as a white box. The *Active Structure* defines the internal structure and the operational modes of the system: system elements, their attributes as well as the relationships between system elements. As in the *Environmental Model* (cf. Sect. 2.1.1), the relationships are specified by means of different kinds of flows. We call the combination of *Environmental Model* and *Active Structure* together system structure.

The *Active Structure* (bottom left in Fig. 2) considers the SUDs BCM and Door as white box and modularizes them into subsystems. The BCM comprises the subsystems CentralLocking and ExteriorLights, which are connected to the environment elements from the *Environmental Model* via delegation connectors. For instance, CentralLocking receives

[1]Ports and port specifications are hidden in Fig. 2 to increase the readability.

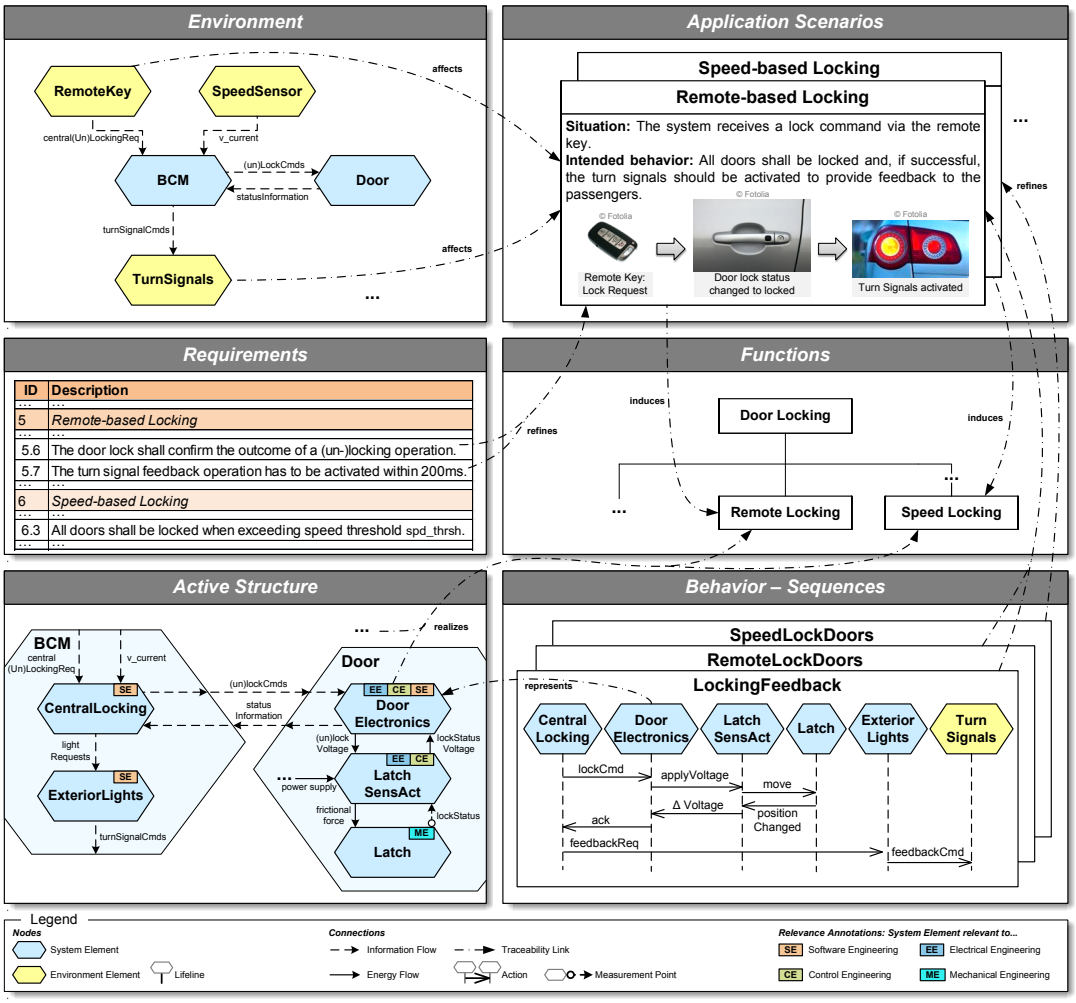

Figure 2: CONSENS system model of the BCM with partial models relevant to SWRE

signals via the connector central(Un)LockingReq from RemoteKey as well as via v_current from SpeedSensor. The Door contains a DoorElectronics subsystem that activates the combined sensor/actuator LatchSensAct by means of the energy flow (un)lockVoltage. LatchSensAct can cause frictional force on the actual Latch and senses its lockStatus.

The system elements have realizes trace links to the functions of the partial model *Functions* if they contribute to the realization of a function. In our example, almost all depicted system elements of the active structure realize both functions (only indicated in Fig. 2). The only exception is ExteriorLights, which has no trace link to Speed Locking.

2.1.7 Allocate Engineering Disciplines

While conceiving the *Active Structure*, the SD determines during the discussion with the discipline-specific experts (e.g., the Software Engineer in Fig. 1) which system elements are to be concretized in which disciplines. As in [21], we use so-called *relevance annotations* to specify the relevance of a system element to a certain discipline in the active structure. We exploit this information in the transition to SWRE to enable the automatic derivation of a MSD specification (cf. Sect. 2.2). We use the relevance annotation "SE" to specify the relevance of a system element to the discipline of software engineering. We call such system elements *SE-relevant*. Since we regard SWRE as sub-discipline of software engineering (cf. Fig. 1), we also regard all SE-relevant system elements as relevant to SWRE.

For instance, the BCM in the *Active Structure* (cf. Fig. 2) contains only software components, whereas the Door contains system elements that are relevant to different disciplines. The DoorElectronics is an electronic component with deployed software. Experts from the disciplines electrical, control, and software engineering jointly design this system element. This is visualized by the relevance annotations "EE", "CE", and "SE", respectively. The electrical engineer designs the electronic parts, e.g., the energy flows from/to LatchSensAct. The software is jointly designed by a software (coordination behavior) and a control engineer (control behavior).

2.1.8 Define System Behavior

Mechatronic systems are characterized by different kinds of behavior. This is reflected in CONSENS by a group of different behavior models: *Behavior – States*, *Behavior – Activities*, and *Behavior – Sequences*. The usage of the models depends on the underlying development task. In this paper, especially the *Behavior – Sequences* are important. Similar to other scenario-based notations, they specify partially ordered, discipline-spanning sequences of *actions* executed on system/environment elements, which are represented by

lifelines similar to UML Sequence Diagrams [28].

Beyond the specification of *Behavior – Sequences* and other behavioral partial models, the SA continuously consolidates discipline-specific work products to ensure a capable system design. As the SA is not able to master all discipline-specific analysis techniques, it is of high importance to initiate an effective integration of all work products and to balance the needs of the disciplines.

The partial model *Behavior – Sequences* (bottom right in Fig. 2) refines the application scenarios and requirements. The partial model of our example contains three scenarios; we focus on the scenario LockingFeedback. This scenario describes the intended behavior of the system elements w.r.t. the activation of the turn signal feedback operation. For example, after a lockCmd has been sent to the DoorElectronics, this system element has to activate the motor of the LatchSensAct by means of the action applyVoltage.

Each application scenario has a refines trace link to a corresponding application scenario as depicted in Fig. 2. Furthermore, each lifeline has a represents trace link to a system or environment element in the active structure and environment, respectively. In Fig. 2, we indicate this correspondence between lifelines and system elements by name equality and one exemplary trace link.

2.2 The Transition to SWRE

The semi-formal CONSENS system model is well suited to facilitate communication between stakeholders like customers, different systems engineering roles, and discipline experts. However, a formal requirements notation is needed to enable automatic techniques for the requirements analysis. As motivated in the introduction, we apply MSDs for this purpose in the discipline of SWRE.

However, it is expensive to develop a complete MSD specification that can be simulated or checked for consistency. This is exaggerated as the system model contains much information that is only partly relevant to all involved disciplines. In Sect. 2.1 we already determined which CONSENS partial models are relevant to SWRE. But as one can see on the system model in Fig. 2, even only parts of the particular, selected partial models are relevant to SWRE. For example, the software requirements engineer is not interested in the electrical details of LatchSensAct or in the operating principles of the mechanical Latch.

Despite being semi-formal, the CONSENS specification provides much information in a model-based and structured way. In order to exploit this information, we present an approach to semi-automatically derive MSD specifications from a CONSENS system model in this section. A key aspect of our transition technique is the identification of information in the system model that is relevant to SWRE. The technique exploits the relevance annotations to identify the SE-relevant system elements within the active structure and the traceability links between the different partial models. Thereby, information that is not relevant to SWRE is filtered.

The target model of our transition technique is a hierarchical MSD specification [22]. Such a MSD specification bases on a subset of modeling constructs of both the UML [28] and the Systems Modeling Language (SysML) [30] and extends these constructs by means of the so-called modal profile. This profile provides additional MSD-specific modeling constructs for UML Sequence Diagrams, which we present in Sect. 2.2.5. Fig. 3 shows the initial MSD specification that

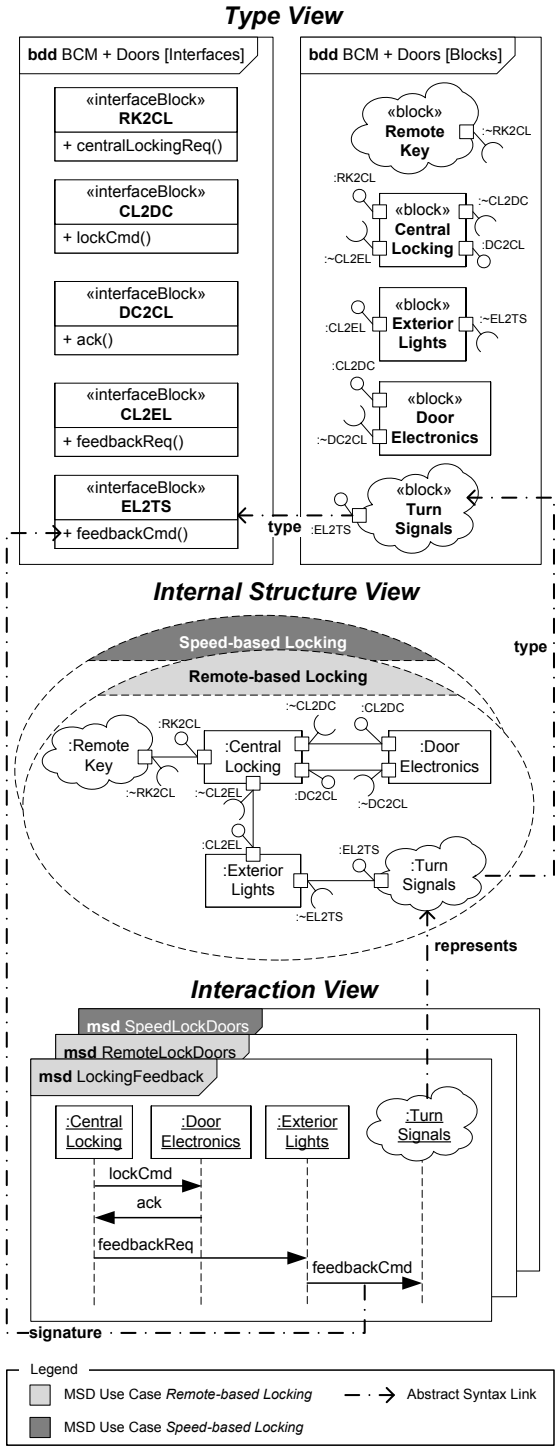

Figure 3: Initially generated MSD specification

is automatically derived from the CONSENS model in Fig. 2. It is structured in three different views.

The Type View provides reusable types (top of Fig. 3). On the one hand, it contains interfaces with operations. These are depicted in the Block Definition Diagram (BDD) named BCM + Doors [Interfaces]. On the other hand, this view encompasses structural blocks owning ports that are typed by the interfaces (cf. BDD named BCM + Doors [Blocks]).

The Internal Structure View contains *MSD use cases*, which

encapsulate the requirements on the coordination behavior of the system or system parts in a specific situation (cf. center of Fig. 3). We use UML collaborations to specify MSD use cases for hierarchical software component architectures. These collaborations define the participants of a MSD use case by means of roles typed by the blocks in the Type View. The communication links between these roles are specified by means of ports (defined by the role types) and connectors within the collaboration.

Each use case contains a set of MSDs, which are specified in the Interaction View (cf. bottom of Fig. 3). The MSDs formally specify requirements on the coordination behavior of the use case participants. Each lifeline in the MSD represents a role within the collaboration of a specific use case. The messages correspond to the operations of the interfaces in the Type View.

We documented the transition technique and described as well as implemented the particular mapping rules from a CONSENS system model to a MSD specification in [5]. We implemented the mappings by means of QVT Operational (QVT-O) [27] model transformations. The source model is a system model specified in a CONSENS language variant based on the SysML profile SYSML4CONSENS [25] in the UML/SysML modeling tool Papyrus[2]. The target model is a MSD specification for hierarchical component architectures [22] based on UML/SysML in the SCENARIOTOOLS[3] tool suite. Since both the source and target model make use of the UML/SysML profiling extension mechanism, the approach can also be implemented in other UML/SysML tools.

In the following, we explain each particular step of our transition as depicted in Fig. 1 based on the running example.

2.2.1 Derive MSD Use Cases

The partial model *Application Scenarios* (cf. Sect. 2.1.2) in the system model provides a system-level structuring of the desired SUD functionality that is similar to the structuring by means of use cases in the MSD specification.

In order to reuse this structuring, we create an empty use case for each SE-relevant application scenario in the system model. We call an application scenario SE-relevant, if at least one SE-relevant system element realizes one of the functions that are induced by this scenario. We use the bidirectional trace links in the system model to determine the SE-relevance of an application scenario and to exactly identify the system elements that are used to realize the functionality specified in this scenario. Furthermore, we use the annotations of these elements: If one is SE-relevant, then also the application scenario is SE-relevant.

For example, the application scenario Remote-based Locking in Fig. 2 is SE-relevant because the system element DoorElectronics is annotated with "SE" and realizes the function RemoteLocking induced by this application scenario. Thus, we add the empty use case Remote-based Locking to the MSD specification, which is depicted by the collaboration Remote-based Locking in Fig. 3. The same principle holds for the MSD use case Speed-based Locking.

2.2.2 Derive Structure

The partial models *Active Structure* (cf. Sect. 2.1.6) and *Environmental Model* (cf. Sect. 2.1.1) in the system model specify the SUD's system architecture and its environment in a discipline-spanning manner, thereby providing a basis for the software architecture of a MSD use case.

In order to derive this software architecture, we apply the same principle as described in Sect. 2.2.1 to determine the SE-relevant system elements realizing an application scenario, i.e. we use the trace links within the system model. We derive a role for the collaboration representing the use case and a corresponding block in the Type View in the MSD specification for each SE-relevant element associated with the corresponding application scenario. For example, the system elements CentralLocking and DoorElectronics realize the application scenario Remote-based Locking and are SE-relevant (cf. Fig. 2). Thus, we derive the roles :CentralLocking and :DoorElectronics within the collaboration Remote-based Locking as well as their typing blocks (cf. Fig. 3).

Additionally, we consider information flows between SE-relevant elements in the structural partial models in CONSENS as SE-relevant flows. For example, the system element DoorElectronics (cf. Fig. 2) has in- and outgoing information as well as energy flows. As motivated before, the software requirements engineer is mainly interested in the coordination behavior, so we only consider the information flows (un)LockCmds and statusInformation in our transition technique. We use the information flows and their ports to derive ports and connectors for the Internal Structure View, e.g., the ports and connectors of :DoorElectronics in the use case Remote-based Locking (cf. Fig. 3). We also use the port specifications of the ports in the system model to derive ports for the blocks and interfaces in the Type View of the MSD specification, i.e., the ports of the block **DoorElectronics** typed by the interface blocks **CL2DC** and **DC2CL**.

As in CONSENS, a MSD specification distinguishes between environment and system elements. These are visualized by cloud symbols and rectangles, respectively. We consider all CONSENS environment elements that are connected via information flows with SE-relevant system elements as SE-relevant. In our example, this encompasses all environment elements. Furthermore, we use the affects trace links to determine the environment elements involved in an application scenario to derive the environment roles of the corresponding use case. For example, we only derive the environment roles :Remote Key and :TurnSignals for the use case Remote-based Locking and neglect the environment element SpeedSensor in the *Environmental Model*. Besides deriving the environment roles within the collaborations, we generate all corresponding MSD specification elements encompassing interfaces, blocks, ports and connectors analogous to system elements.

2.2.3 Derive MSDs

Similar to MSDs and scenario-based formalisms in general, the partial model *Behavior – Sequences* specifies partially ordered sequences of actions. Therefore, we reuse the information contained in this partial model to derive initial MSDs for the MSD specification.

We create one MSD in the MSD specification for each Behavior – Sequence in the system model (cf. MSDs in Fig. 3 stemming from Behavior – Sequences in Fig. 2). As it is the case for the structural partial models, the partial model *Behavior – Sequences* contains discipline-spanning information that can be irrelevant to SWRE. For example, the Behavior – Sequence LockingFeedback specifies actions like changing voltages between DoorElectronics and LatchSensAct as well as physical and sensor actions on Latch. Since such actions are

[2] http://www.eclipse.org/papyrus/
[3] http://scenariotools.org/

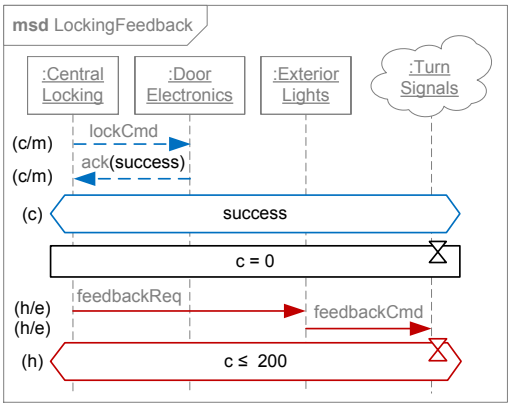

Figure 4: Manually refined MSD LockingFeedback

not part of the message-based coordination behavior between software components, we only reuse Behavior – Sequence lifelines that represent SE-relevant system/environment elements and the actions between them. For this purpose, we exploit the represents links from a Behavior – Sequence lifeline to system/environment elements (cf. Fig. 2). We consider a Behavior – Sequence lifeline as SE-relevant if it represents a SE-relevant system or environment element (cf. Sect. 2.2.2). Furthermore, we map an initially derived MSD to a MSD use case if the source Behavior – Sequence has a refines trace link to the corresponding application scenario (cf. shades of gray for MSDs and use cases in Fig. 3). The Behavior – Sequence lifelines are mapped to lifelines in the corresponding MSD and the actions are mapped to messages between the corresponding MSD lifelines. SE-relevant lifelines representing environment elements are only mapped to MSD lifelines if the environment element is involved in the application scenario. As a result, we do not derive lifelines and incident messages for LatchSensAct and Latch into the MSD LockingFeedback (cf. Fig. 3), for example.

Our experience shows that—in contrast to the partial models *Environmental Model*, *Application Scenarios*, and *Active Structure*—the partial model *Behavior – Sequences* is often specified incompletely or not at all. Thus, this transition step is optional and does not completely replace manual work of the software requirements engineer to specify MSDs.

2.2.4 Refine MSD Specification

The initially derived MSDs have no MSD-specific modeling constructs. Thus, the initially derived MSD specification has to be refined. CONSENS provides information relevant to this refinement within the partial models *Application Scenarios* and *Requirements*. Since the contained information is informal text, we cannot automate this step. Thus, the software requirements engineer has to perform it manually.

We illustrate the refinement exemplarily on the MSD LockingFeedback. Fig. 4 depicts this MSD after the refinement step has been performed. The information that was initially derived (cf. MSD LockingFeedback in Fig. 3) is visualized in gray, and information that is added during the refinement step is emphasized by full colors.

First, the software requirements engineer has to analyze in which situation the visual feedback shall be given. The first two messages `lockCmd` and `ack` represent the context of the MSD. For this reason, he sets the *temperature* of these messages to cold (depicted by blue message arrows and indicated by `c` on the left side of Fig. 4), meaning that other messages specified in the MSD can occur earlier or later. If this is the case, this "aborts" the current progress of the MSD. Furthermore, he sets their *execution kind* to monitored (depicted by dashed message arrows and indicated by `m` on the left side of Fig. 4), meaning that this message may or may not occur.

Second, the application scenario Remote-based Locking (top right of Fig. 2) specifies that the visual feedback is only given after a successful lock command. Thus, the software requirements engineer adds a boolean parameter `success` to the message `ack` (and extends the corresponding operation in the interface **DC2CL**, which is not depicted in Fig. 4). Furthermore, he adds a cold *condition* (depicted by the blue hexagon in Fig. 4) with the expression `success`, which evaluates to a Boolean value depending on the value assignment of `success`. If the condition evaluates to `false`, the MSD is discarded. Otherwise, it progresses beyond the condition.

Third, the software requirements engineer analyzes the mandatory behavior of the application scenario. Since the intended behavior is to send the messages `feedbackReq` and `feedbackCmd` to activate the turn signal feedback operation, he sets their temperature to hot (depicted by red message arrows and indicated by `h` on the left side of Fig. 4). This means that other messages specified by the scenario must not occur at this point in time. Furthermore, he sets their execution kind to monitored (depicted by solid message arrows and indicated by `e` on the left side of Fig. 4), which means that the messages must eventually occur.

Finally, requirement 5.7 in the CONSENS requirements specifies that the turn signal feedback operation has to be performed within 200ms. To formalize this requirement, the software requirements engineer for one thing specifies a *clock reset* (depicted by the black rectangle with an hour-glass icon in Fig. 4) for the clock variable `c`, which sets `c` to zero. For another thing, he adds a hot *clock condition* (depicted by the red hexagon with an hour-glass icon in Fig. 4) at the end of the MSD. If its expression $c \leq 200$ evaluates to `false`, i.e., the sending of the messages `feedbackReq` and `feedbackCmd` lasts longer than 200ms, this is a safety violation.

Of course, the manual refinement of a MSD specification is not restricted to the aspects mentioned above. The software requirements engineer can freely add details to and change all aspects of a MSD specification.

2.2.5 Analyze Coordination Behavior Requirements

The formal semantics of MSDs enable different techniques for the tool-supported analysis of the requirements on the coordination behavior of the SUD. Fig. 5 shows a situation identifiable in Play-out, which operationalizes the MSD semantics and is a means for simulative validation. The software requirements engineer can observe that the MSD RemoteLockDoors, responsible for the actual locking operation, correctly activates the MSD LockingFeedback triggering the turn signal feedback operation. However, he also reveals that the MSD SpeedLockDoors activates this MSD, if he simulates both use cases together. That is, the coordination behavior requirements specify that the speed locking activates the turn signals during driving, which can cause safety-critical confusion in the traffic.

Using these (semi-)automatic analysis techniques, the software requirements engineer is able to identify such unintended behavior and scenario inconsistencies on requirements level.

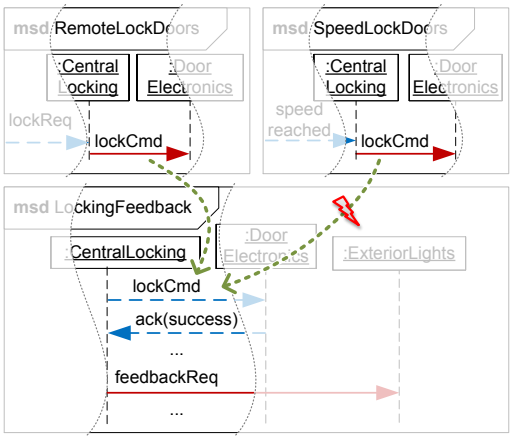

Figure 5: Identifying unintended behavior by means of simulative validation

2.2.6 *Consolidate Discip.-specific Analysis Results*

The software requirements engineer as well as the requirements engineers from the other engineering disciplines hand over their respective analysis results to the SA. He consolidates the different discipline-specific analysis results and decides how to proceed.

3. RELATED WORK

Similar to our work, Böhm et al. [8] present a model-based methodology for the transition from systems to software engineering compliant to established process models. They also address that these process models demand a software requirements analysis phase prior to the software design: A system element to be concretized in software engineering is input to the context model of the SPES requirements viewpoint [13]. This context model is comparable to our *Environmental Model* on system level. However, they only consider structural but no behavioral models, which are crucial for the specification of functional requirements. Furthermore, they do not provide automatisms to support the transition.

Barbieri et al. [4] present a systems engineering approach based on the SysML-extension SysML4*Mechatronics*. They describe a coarse-grained design process and introduce stereotypes for system elements to be concretized in their respective engineering disciplines, which is very similar to our relevance annotations. However, the approach aims for analyzing discipline-spanning change influences and not on the transition to discipline-specific models. Furthermore, the discipline-specific information in the system model is not exploited to facilitate the automation of such a transition.

The SysML-based MBSE methods Object-Oriented Systems Engineering Method (OOSEM) [14] and SYSMOD [38] serve the same purpose as Consens. OOSEM only distinguishes between software and hardware, and SYSMOD promotes a more software-focused approach since the hardware elements are often considered as environment elements. In contrast, Consens promotes a multidisciplinary approach facilitating the communication between all engineering disciplines at eye level. Furthermore, OOSEM and SYSMOD only propose to refine the system requirements within a typical software engineering process, but the transition to this process is out of their scope. Nevertheless, the basic principle of our transition from MBSE to SWRE could also be transferred to OOSEM and SYSMOD.

Several approaches exist to systematize and automate the transition from MBSE to other discipline-specific design and analysis models, e.g., in the area of control engineering. The OMG developed a bidirectional transformation from SysML to Modelica [29]. Similarly, Cao et al. present an integration between SysML models with MATLAB/Simulink based on a bidirectional model transformation [11]. Such approaches focus on the derivation of discipline-specific design and analysis models. In contrast to our work, the derivation or refinement of discipline-specific requirements from system requirements is not addressed.

4. CONCLUSIONS AND OUTLOOK

In this paper, we introduced a systematic process to integrate SWRE into MBSE based on Consens and MSDs. We assigned the particular process steps to different roles involved in the integrated development process. We screened the Consens language and identified six partial models in Consens that contain information relevant to SWRE. The presented technique enables to filter the SWRE-relevant information from these interdisciplinary partial models and automates process steps by means of model transformations where possible.

The transition process guides the software requirements engineer to perform the particular steps in a systematic way. Thereby, the task assignment to roles clarifies the distribution of responsibilities. Due to our screening, the software requirements engineer is aware of the partial models that he has to analyze. Our process automation increases his efficiency by reducing manual work, which is time-consuming and error-prone. Furthermore, the automatically derived parts of a MSD specification assures that he does not overlook SWRE-relevant information.

Future work is diverse. First, we want to combine our approach with the existing approaches for the transition from MBSE with Consens to software engineering. That is, we want to combine it, on the one hand, with the transition to software design models [21] to facilitate the subsequent software development. On the other hand, we want to combine it with the transition to software engineering using solution patterns [2] to enable the reuse of parts of MSD specifications and software design models. Second, we focused on requirements on the coordination behavior part of the overall system. In the future, we also want to consider requirements for the control behavior and other disciplines to enable an interdisciplinary requirements engineering. Finally, we want to extend our approach with model synchronization techniques to provide additional process support by means of the following features: on the one hand, enable an incremental transformation from Consens to the MSD specification and, on the other hand, propagate changes in the MSD specification influencing the system model back to Consens.

5. ACKNOWLEDGMENTS

This research and development project is funded by the German Federal Ministry of Education and Research (BMBF) within the Leading-Edge Cluster “Intelligent Technical Systems OstWestfalenLippe” (it’s OWL), managed by the Project Management Agency Karlsruhe (PTKA) as well as by the ITEA 2 AMALTHEA4public project (No. 01IS14029I), managed by the Project Management Agency of the German Aerospace Center (PT-DLR).

6. REFERENCES

[1] S. Abrahão, C. Gravino, E. Insfran, G. Scanniello, and G. Tortora. Assessing the effectiveness of sequence diagrams in the comprehension of functional requirements: Results from a family of five experiments. *IEEE Transactions on Software Engineering*, 39(3):327–342, 2013.

[2] H. Anacker, J. Gausemeier, R. Dumitrescu, S. Dziwok, and W. Schäfer. Solution patterns of software engineering for the system design of advanced mechatronic systems. In *MECATRONICS REM*, pages 101–108. IEEE, 2012.

[3] Automotive Special Interest Group (SIG). Automotive SPICE: Process reference model, release v4.5, 2010.

[4] G. Barbieri, K. Kernschmidt, C. Fantuzzi, and B. Vogel-Heuser. A SysML based design pattern for the high-level development of mechatronic systems to enhance re-usability. In *IFAC World Congress*, pages 3431–3437. IFAC, 2014.

[5] R. Bernijazov. *Systems and Software Requirements Engineering for Cyber-Physical Systems*. Bachelor's thesis, University of Paderborn, Paderborn, 2015.

[6] B. W. Boehm. Seven basic principles of software engineering. *J. Syst. Software*, 3(1):3–24, 1983.

[7] B. W. Boehm. Unifying software engineering and systems engineering. *Computer*, 33(3):114–116, 2000.

[8] W. Böhm, S. Henkler, F. Houdek, A. Vogelsang, and T. Weyer. Bridging the gap between systems and software engineering by using the SPES modeling framework as a general systems engineering philosophy. In *Systems Engineering Research*, pages 187–194, 2014.

[9] C. Brenner, J. Greenyer, J. Holtmann, G. Liebel, G. Stieglbauer, and M. Tichy. ScenarioTools real-time play-out for test sequence validation in an automotive case study. In *GT-VMT*, 2014.

[10] C. Brenner, J. Greenyer, and V. Panzica La Manna. The ScenarioTools play-out of modal sequence diagram specifications with environment assumptions. In *GT-VMT*, 2013.

[11] Y. Cao, Y. Liu, and C. J. Paredis. System-level model integration of design and simulation for mechatronic systems based on SysML. *Mechatronics*, 21(6):1063 – 1075, 2011.

[12] W. Damm and D. Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19:45–80, 2001.

[13] M. Daun, B. Tenbergen, and T. Weyer. Requirements viewpoint. In *Model-Based Engineering of Embedded Systems*, pages 51–68. Springer, 2012.

[14] S. Friedenthal, A. Moore, and R. Steiner. *A Practical Guide to SysML*. MK/OMG Press, 3rd edition, 2014.

[15] J. Gausemeier, F.-J. Rammig, and W. Schäfer. *Design Methodology for Intelligent Technical Systems*. Lecture Notes in Mechanical Engineering. Springer, 2014.

[16] J. Greenyer, C. Brenner, M. Cordy, P. Heymans, and E. Gressi. Incrementally synthesizing controllers from scenario-based product line specifications. In *Proc. ESEC/FSE*, pages 433–443. ACM, 2013.

[17] D. Guest. The hunt is on for the renaissance man of computing. *The Independent (London)*, 17, 1991.

[18] D. Harel and S. Maoz. Assert and negate revisited: Modal semantics for UML sequence diagrams. *Software & Systems Modeling*, 7:237–252, 2008.

[19] D. Harel and R. Marelly. *Come, let's play: Scenario-based programming using LSCs and the play-engine*. Springer, 2003.

[20] J. Hassine, J. Rilling, and R. Dssouli. An evaluation of timed scenario notations. *J. Syst. Software*, 83(2):326–350, 2010.

[21] C. Heinzemann, O. Sudmann, W. Schäfer, and M. Tichy. A discipline-spanning development process for self-adaptive mechatronic systems. In *ICSSP*, pages 36–45. ACM, 2013.

[22] J. Holtmann and M. Meyer. Play-out for hierarchical component architectures. In *Automotive Software Engineering*, volume P-220 of *LNI*, pages 2458–2472. Bonner Köllen Verlag, 2013.

[23] INCOSE. Systems engineering handbook: A guide for system lifecycle processes and activities (version 3.2.2), 2011.

[24] INCOSE. *A World in Motion: Systems Engineering Vision 2025*, 2014.

[25] L. Kaiser, R. Dumitrescu, J. Holtmann, and M. Meyer. Automatic verification of modeling rules in systems engineering for mechatronic systems. In *ASME IDETC/CIE*. ASME, 2013.

[26] B. Nuseibeh and S. Easterbrook. Requirements engineering: A roadmap. In *Future of Software Engineering*, pages 35–46. ACM, 2000.

[27] OMG. Meta Object Facility (MOF) 2.0 Query/View/Transformation specification: Version 1.1, 2011.

[28] OMG. OMG Unified Modeling Language (OMG UML) superstructure: Version 2.4.1, 2011.

[29] OMG. OMG SysML Modelica Transformation: Version 1.0, 2012.

[30] OMG. OMG Systems Modeling Language (OMG SysML): Version 1.3, 2012.

[31] OMG. Business Process Model and Notation (BPMN): Version 2.0.2, 2013.

[32] K. Pohl and C. Rupp. *Requirements Engineering Fundamentals*. Rocky Nook, 2011.

[33] U. Pohlmann, J. Holtmann, M. Meyer, and C. Gerking. Generating Modelica models from software specifications for the simulation of cyber-physical systems. In *SEAA*, pages 191–198. IEEE, 2014.

[34] M. Ryschkewitsch, D. Schaible, and W. Larson. The art and science of systems engineering. *Systems Research Forum*, 3(2):81–100, 2009.

[35] W. Schäfer and H. Wehrheim. The challenges of building advanced mechatronic systems. In *Future of Software Engineering*, pages 72–84. IEEE, 2007.

[36] S. A. Sheard. Twelve systems engineering roles. *INCOSE Int. Symposium*, 6(1):478–485, 1996.

[37] C. Tschirner, L. Kaiser, R. Dumitrescu, and J. Gausemeier. Collaboration in model-based systems engineering based on application scenarios. In *NordDesign*, 2014.

[38] T. Weilkiens. *Systems Engineering with SysML/UML: Modeling, Analysis, Design*. MK/OMG Press, 2007.