Michaela Huhn, Hardi Hungar, Matthias Riebisch, Sebastian Voss (Hrsg.):
Modellbasierte Entwicklung Eingebetteter Systeme,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2018 

# Finding Inconsistencies in Design Models and Requirements by Applying the SMARDT Process

Stefan Kriebel[1] Evgeny Kusmenko[2] Bernhard Rumpe[2] Michael von Wenckstern[2] ✉

**Abstract:** The development of safety critical systems requires a highly automated integrated methodology supporting the design, verification and validation of the overall product. Such a methodology maintains the consistency and correctness of all artifacts at all development stages ideally incorporating requirements changes into the corresponding models, tests, and code. This paper shows how the SMARDT process uses formalized SysML diagrams to identify inconsistencies in architectural designs and requirements. An adaptive light system serves as illustrative running example.

## 1 Introduction

Safety critical software systems undergo a complex development process before they are introduced onto the market. The manufacturers are obliged to make their systems ISO26262 compliant and to guarantee the correctness of their implementation. Usually such systems become very large and have to fulfill hundreds of intertwined requirements developed by different teams. Therefore, an urgent need for automated means of consistency checking of requirement specifications identifying errors in early design stages has been arising.

At BMW, one of Germany's largest automotive companies, the new SMARDT process tries to tackle this problem. One of its goals is to ensure artifact consistency for the different phases of the development process by using model-based software engineering (MBSE), particularly formalized SysML diagrams.

This paper shows how the SMARDT process uses formalized SysML diagrams to **identify inconsistencies in design decisions and requirements**. Thereby, we focus on artifacts of the logical layer of the SMARDT process and demonstrate the idea on the requirement specification of a real-world Adaptive Light System (ALS).

Section 3 recaps the main ideas of the SMARDT process. Section 4 shows how Component & Connector (C&C) views reveal structural design inconsistencies derived from different

[1] BMW Group, Munich, Germany; stefan.kriebel@bmw.de
[2] RWTH Aachen University, Software Engineering, Ahornstraße 55, 52074 Aachen, Germany; {kusmenko,rumpe,vonwenckstern}@se-rwth.de

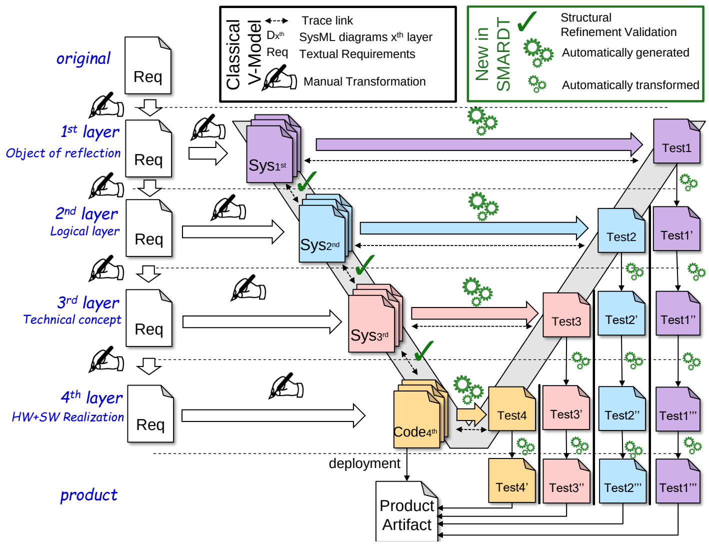


Fig. 1: Overview of the SMARDT methodology (copied from [Hi18]).

requirement specifications. Section 5 presents ideas how formalized Activity Diagrams (ADs) help to detect behavioral inconsistencies in requirements.

## 2 Running Example

To demonstrate the feasibility and the advantages of the proposed methodology we use seven real-world requirement specifications of an ALS [Be17]. The ALS model controls adaptive high and low beam, turn signals as well as cornering and ambient light. Adaptive high and low beam adjust headlamps to the traffic situation and provide optimized illumination without dazzling others. Cornering light illuminates the area to the side of a vehicle to take a look around the bend. Ambient light welcomes the driver with an indirect light. For inspection purposes all 93 original requirements of the ALS are available at our homepage[3].

## 3 The SMARDT Methodology

The SMARDT (*Specification Methodology Applicable to Requirements, Design, and Testing*[4]) approach [Hi18] extends the German V-Model [BD95], which is the official project management methodology of the German government. The SMARDT process

[3] `http://www.se-rwth.de/materials/cncviewscasestudy/`

[4] The original abbreviation SMArDT is related to the German term "**S**pezifikations-**M**ethode für **A**nfo**r**derung, **D**esign und **T**est"

delivers reliable specification methodologies by introducing formalized SysML diagrams [OM15] with a well-defined semantics [HR04] to specify the functionality of automotive systems. This allows to automatically ensure a permanent consistency between all abstraction layers of the V-Model. This task can become particularly tedious if maintained by hand in agile development processes as those are mostly iterative, incremental, and evolutionary [Be01]. New validations enabled by SMARDT process are: (1) backward compatibility checks [Ru15, Ri16, Be16, Ku18] for software maintenance and evolution between different diagram versions of the same layer, (2) refinement checks [Ru96, HRvW17] between diagrams of different layers allowing to detect specification inconsistencies between different layers, as well as (3) advanced structural or extra-functional property [Ma16] checks on SysML diagrams using OCL [Ma17].

The key principles of SMARDT for a formal specification of requirements, design, and testing of system engineering artifacts according to ISO 26262 are illustrated in Figure 1. The methodology is structured in four layers: (1) The object of reflection layer contains a first description of the object under consideration and shows its boundaries from a customer's point of view. (2) The logical layer containing functional specifications without details of their technical realizations. (3) The technical concept, e.g. C code or Simulink models, belongs to the third layer. Finally, the fourth layer represents the software and hardware artifacts present in the system's implementation.

In SMARDT consistency between different layers is ensured by verification and model-based testing of the final product against the requirements of all layers. More specifically, SMARDT enables structural verification as explained in Section 4 between each layer indicated by the green check marks in Figure 1. Furthermore, SMARDT enables a systematic and fully automatic derivation of test cases for each layer as discussed in [Hi18] and illustrated on the right side of Figure 1. The previous SMARDT paper [Hi18] showed how this methodology is applied to develop a self-driving racing vehicle; its main focus was the use of formalized ADs enriched by OCL constraints for automated test case derivation. In contrast, this paper focuses on detecting structural and behavioral inconsistencies between different artifacts in the logical layer of the SMARDT process.

## 4 Architecture Specification using Views

Embedded software systems are often created in Simulink as C&C models describing functional, logical or software architectures [TMD09] in terms of components executing computations and connectors effecting component interaction via typed and directed ports. An advantage of this approach is that complex components such as ALS can be hierarchically decomposed into other smaller components to be developed by different teams. Interaction between components occur only via connectors. SysML and Modelica are two other famous representatives for C&C modeling languages.

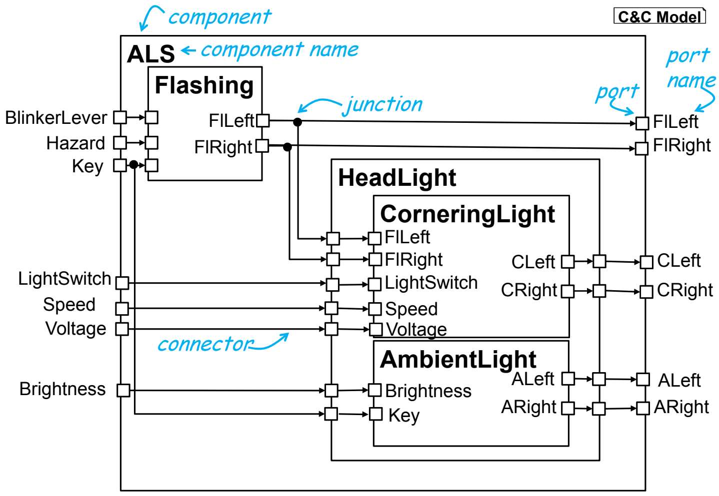


Fig. 2: Very simple example C&C model for ALS.

C&C views, as presented in [MRR13], are developed to focus on important view points (excerpts) of large C&C models without being required to model all the other (for the view point unimportant) information. For example a view can only show how a component is decomposed into subcomponents without showing any ports and connections of the subcomponents. The main aim of C&C views concept is to have many but therefore small, precise and good-readable view points of one large C&C model. For this reason C&C views introduce four major abstractions: hierarchy, connectivity, data flow, and interfaces of C&C models. The hierarchy of components in C&C views is not necessarily direct (intermediate components may be skipped); abstract connectors can cross-cut component boundaries and they can also connect components directly (if the concrete connected port is not important for this view); abstract effectors describe data flow abstracting over chains of components and connectors, and C&C views do not require complete interfaces with port names, types and array sizes. Intuitively, a C&C model satisfies a C&C view iff all elements and relations shown by the view have a satisfying concretization in the model. The formal definitions of C&C model, C&C view, and their satisfaction are available in [MRR13] and from supporting materials website of paper [Be17].

**C&C View Verification and Witnesses for Tracing** Figure 2 shows the example C&C model of an adaptive light system (*ALS*). This simplified model controls the flash LEDs for turning as well as the left and right light bulbs for cornering and ambient lights. The `ALS` consists of the two subcomponents `Flashing` and `HeadLight`. Component `HeadLight` is hierarchically decomposed into two further components: `CorneringLight`, and `AmbientLight`. The C&C view *ALS1* shown in Figure 3 describes the `CorneringLight` component illuminating on intersections the road where the driver wants to turn into. Thus, the input port `BlinkerLever`

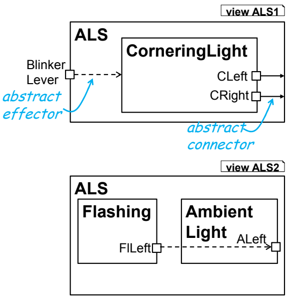


Fig. 3: Two C&C views *ALS1* (top) and *ALS2* (bottom)

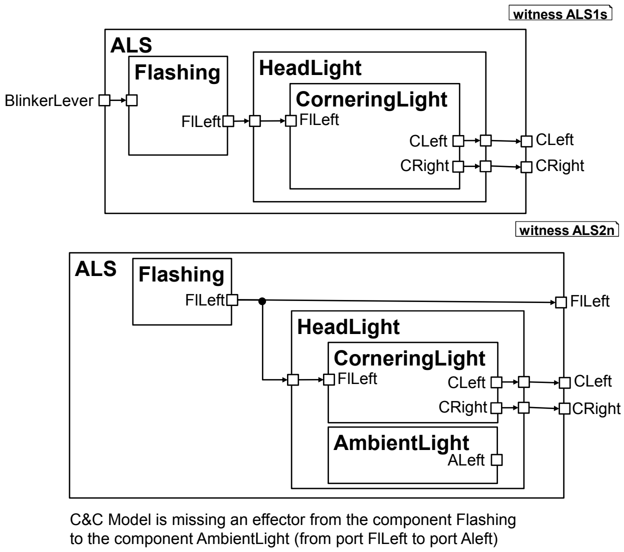


Fig. 4: Witness for satisfaction *ALS1s* (top) and witness for non-satisfaction *ALS2n* (bottom)

of the `ALS` component has effect (modeled by an abstract effector) to at least one input port of the `CorneringLight` component. The calculated light values (`CLeft`, and `CRight`) of the `CorneringLight` component are passed directly (without being modified anymore) to the output ports of the `ALS` component (modeled by two abstract connectors). The C&C view *ALS2* is about the relation between the `Flasing` and the `AmbientLight` component. It specifies that the `FlLeft` (flashing left) output of components `Flashing` effects the ambient left light (port `ALeft` of component `AmbientLight`); e.g. brighter left ambient light when the left parking mode is activated. The model *ALS* satisfies the view *ALS1*.

C&C view verification, as presented in [MRR14], gets as input a C&C model and a C&C view. Besides a Boolean answer whether the C&C model satisfies the C&C view, the verification algorithm produces a local minimal satisfaction or one or more local non-satisfaction witnesses. As an example, a witness for satisfaction *ALS1s* is shown in Figure 4 and demonstrates how the C&C model satisfies *ALS1*. The SMARDT approach uses the generated witnesses to automatically generate traceability information between SysML artifacts of layer 2 (C&C views) against the large SysML model of layer 3 (C&C models). The witness is itself a well-formed model. The witness contains all view's components (here `ALS`, and `CornerningLight`) as well as their parent components to show the complete hierarchy between two components specified in the view. Therefore, the witness contains also the `HeadLight` component. The *positive satisfaction witness* also contains all ports corresponding to a view, therefore the witness contains `BlinkerLever` port of `ALS` as well as `CLeft`, and `CRight` ports of `CorneringLight`. Additionally the witness contains all model connectors and all data-flow paths. The abstract connector from `CorneringLight` (port `CLeft`) to `ALS` (port *unknown*) introduces the following elements in the witness: (1) port `CLeft` of component `HeadLight`; (2) connector of ports `CLeft` from component `CorneringLight` to component `HeadLight`; and (3) connector of ports `CLeft` from component `HeadLight` to component `ALS`. For the abstract effector from `ALS` (port `BlinkerLever`) to `CorneringLight` the following elements in the chain serve as witness: (1) component `Flashing`; (2) ports `BlinkerLever` and `FlLeft` of `Flashing`; (3) connector of ports `BlinkerLever` from `ALS` to `Flashing`; (4) connector of ports `FlLeft` from `Flashing` to `HeadLight`; and (5) connector of ports `FlLeft` from `HeadLight` to `CorneringLight`.

The model *ALS* does not satisfy the view *ALS2*. Every *negative non-satisfaction witness* contains a minimal subset of the C&C model and a natural-language text, which together explain the reason for non-satisfaction. These witnesses are divided into five categories: MissingComponent, HierarchyMismatch, InterfaceMismatch, MissingConnection, Missing-Effector (see [MRR14]). A witness for non-satisfaction *ALS2n* (case MissingEffector) is shown in Figure 4. It shows all outgoing connector-effector chains starting at port `FlLeft` of component `Flashing` as well as the abstract effector's target port, `AmbientLight`'s `ALeft`, which is not reachable. Removing the effectors in the view *ALS2* would cause the model to satisfy this modified view even though `Flashing` and `AmbientLight` are direct siblings in the C&C view and are not direct siblings in the C&C model; C&C views allow to abstract away the intermediate component `HeadLight`.

**Identifying Design Inconsistencies with C&C Views** The previous paragraphs showed how C&C views verification can be used to check structural consistencies and how to generate tracing witnesses between artifacts of layer B (view models) against artifact of layer C (concrete logical C&C models). This part focuses on identifying structural design inconsistencies between different artifact models of layer B. Figure 5 shows two requirements about the *cornering light*. Since the last requirement **AL-139** is a safety feature for armored vehicles, a special team is responsible for it. Thus, the architectural designs (`view AL-122`, and `view AL-139`) for the two requirements are developed by two different teams. The top design shows the `CorneringLight` with two modes (subcomponents `Cornering_Active`, `Cornering_Deactive`, and *MultiSwitch*), whereby the mode `Cornering_Deactive` is selected if the voltage is too low. In the bottom design model the `min` block in combination with the `Switch` one deactivates indirectly the cornering light by propagating 0% as light value to `OvervoltageProtection`'s input ports when the `DarknessSwitch` port has the value true. C&C view synthesis [MRR13] is the process in deriving one valid C&C model which satisfies all given C&C views. The first part of this algorithm checks whether views contain design contradictions. When all views as in our example are positive views (only expressing what should be present; and do not contain negations such as component A should not contain port B), then the contradiction check can be done in polynomial time and scales very well to many hundreds views.

The contradiction check for the both views `view AL-122` and `view AL-139` would result in an

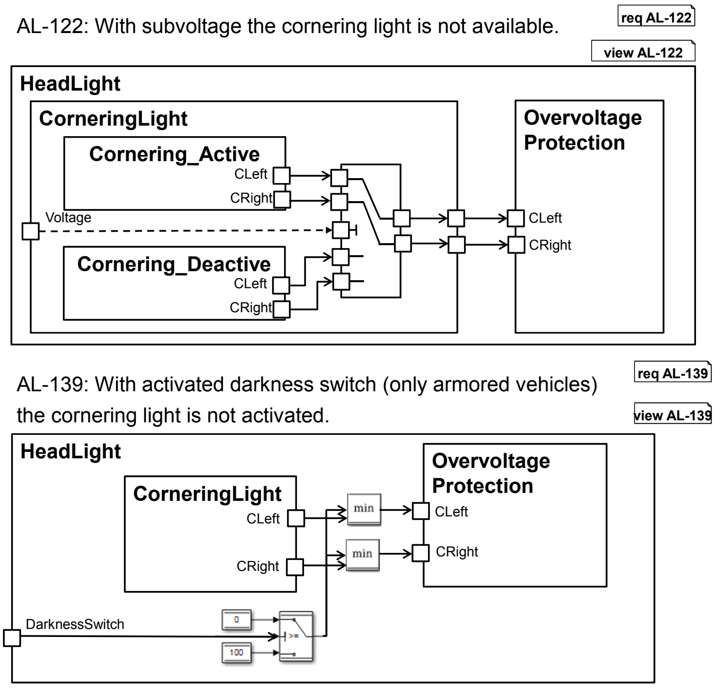


Fig. 5: Design Inconsistency of two C&C views

error as in the top view the port `CLeft` of component `CorneringLight` is directly connected (without modifying the value) to the component `OvervoltageProtection` whereas in the bottom view the value from `CorneringLight`'s `CLeft` is manipulated by the `min` component before it goes to the `OvervoltageProtection`'s `CLeft` port. Similar to the C&C view verification process presented above, the contradiction algorithm generates an intuitive witness to highlight incompatible parts of two views. The formalized C&C view verification problem with its derived contradiction problem enables early analysis of structural design models in the SMARDT process to detect as early as possible inconsistencies between different artifacts created by different persons or teams, to avoid problems when integrating at a later time step different software modules developed on inconsistent designs.

## 5 Formalized Activity Diagrams

In [Hi18] we showed how formalized ADs can be used in combination with OCL constraints in order to generate test cases. In this paper, we demonstrate how formalized ADs can help finding inconsistencies in requirement specification. Consider the Figures 6 and 7 illustrating two ADs describing the steering column stalk and the hazard lights behavior based on the requirements *AL-40* and *AL-41* from [Be17], respectively. In Figure 6 the

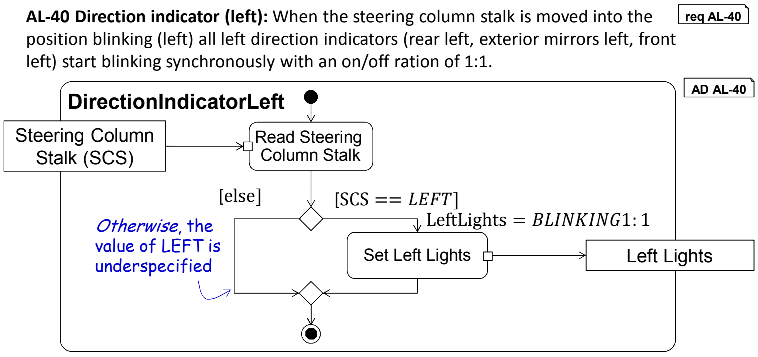


Fig. 6: Activity diagram for AL-40 describing the steering column stick behavior.

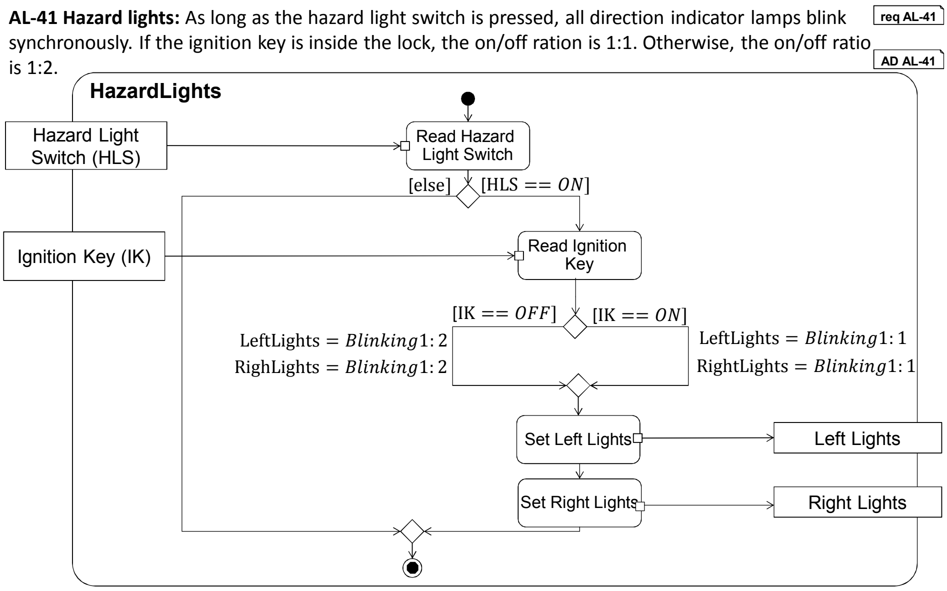


Fig. 7: Activity diagram for AL-41 describing the emergency lights behavior.

model would receive the state of the steering column stalk through the corresponding input port. If the position of the steering column stalk is set to left, the left lights of the car are set to the blinking mode with an on/off ratio of 1:1, denoted as `BLINKING1:1`. This value is written to the output port of the AD. The *else* branch is underspecified, i.e., nothing is said about the output for the cases if the steering column stalk is set to right or to straight. Note, that this is in accordance with the underlying requirement.

In the AD of Figure 7, the hazard light switch is read in order to decide whether to turn the lights on or not. However, to determine the concrete blinking mode, the position of the ignition key is required. If the key is inserted, the 1:1 blinking mode is activated, otherwise the 1:2 mode is used to save battery power. The *else* branch is underspecified again. Note, that according to the requirement the same physical lights are used for hazard blinking as for direction indication. This leads to a contradiction which becomes obvious in the two ADs: if the key is not inside the lock and the hazard light switch is pressed, according to **AL-41**, the vehicle's indicators lamps should blink with a ratio of *1:2*. Now imagine that at the same time the steering column stalk is set to *blinking (left)*. According to **AL-40** the blinking ratio of the left direction indicators should be *1:1*. Considering that a faster blinking drains the battery in a shorter amount of time, this specification error might even become critical for human lives in cases of emergency. The error is discovered automatically by creating a table mapping the three input signals to the specified lamp light modes and filling it with all *specified* combinations. For Figure 6 we would obtain one single combination, namely: $(SCS = LEFT, HLS = DC, IK = DC) \rightarrow (LeftLights = BLINKING1 : 1, RightLights = DC)$ where $DC$ stands for "don't care". Figure 7 produces two combinations one of which is $(SCS = DC, HLS = ON, IK = OFF) \rightarrow (LeftLights = BLINKING1 : 2, RightLights = BLINKING1 : 2)$. Expanding the $DC$ fields to all possible values reveals that the specification requires two contradictory outputs for the same input.

## 6 Conclusion

In this paper we discussed the need for automated consistency checks for requirement and design artifacts of safety critical systems. Therefore, we developed a methodology supporting the SMARDT process - a formalized version of the widely used V-Model - by identifying wrong or contradicting requirements at early development stages using C&C views and activity diagrams. The methodology was demonstrated on a real-world ALS requirement specification. Using the proposed concepts we were able to detect structural and behavioral inconsistencies on the logical layer of the SMARDT process, i.e., long before the actual implementation would be created. Future work comprises analysis of further formalized diagrams and their respective combinations as well as detailed case studies of the complete SMARDT process and the proposed tooling.

**Acknowledgements** This research was supported by a Grant from the GIF, the German-Israeli Foundation for Scientific Research and Development, and by the Grant SPP1835 from DFG, the German Research Foundation.

## References

[BD95] Bröhl, Adolf-Peter; Dröschel, Wolfgang: Das V-Modell. München, Wien: Oldenburg-Verlag, 1995.

[Be01] Beck, Kent; Beedle, Mike; Van Bennekum, Arie; Cockburn, Alistair; Cunningham, Ward; Fowler, Martin; Grenning, James; Highsmith, Jim; Hunt, Andrew; Jeffries, Ron et al.: Manifesto for agile software development. 2001.

[Be16] Bertram, Vincent; Roth, Alexander; Rumpe, Bernhard; von Wenckstern, Michael: Extendable Toolchain for Automatic Compatibility Checks. In: OCL'16. 2016.

[Be17] Bertram, Vincent; Maoz, Shahar; Ringert, Jan Oliver; Rumpe, Bernhard; von Wenckstern, Michael: Case Study on Structural Views for Component and Connector Models. In: MODELS. 2017.

[Hi18] Hillemacher, Steffen; Kriebel, Stefan; Kusmenko, Evgeny; Lorang, Mike; Rumpe, Bernhard; Sema, Albi; Strobl, Georg; von Wenckstern, Michael: Model-Based Development of Self-Adaptive Autonomous Vehicles using the SMARDT Methodology. In: MODELSWARD'18. 2018.

[HR04] Harel, David; Rumpe, Bernhard: Meaningful Modeling: What's the Semantics of "Semantics"? IEEE Computer, 37(10), 2004.

[HRvW17] Heithoff, Malte; Rumpe, Bernhard; von Wenckstern, Michael: Anforderungsverikation von Komponenten- und Konnektormodellen am Beispiel Autonom Fahrender Autos. GI Softwaretechnik-Trends, 37(2), 2017.

[Ku18] Kusmenko, Evgeny; Shumeiko, Igor; Rumpe, Bernhard; von Wenckstern, Michael: Fast Simulation Preorder Algorithm. In: MODELSWARD. 2018.

[Ma16] Maoz, Shahar; Ringert, Jan Oliver; Rumpe, Bernhard; Wenckstern, Michael von: Consistent Extra-Functional Properties Tagging for Component and Connector Models. In: ModComp. 2016.

[Ma17] Maoz, Shahar; Mehlan, Ferdinand; Ringert, Jan Oliver; Rumpe, Bernhard; von Wenckstern, Michael: OCL Framework to Verify Extra-Functional Properties in Component and Connector Models. In: ModComp. 2017.

[MRR13] Maoz, Shahar; Ringert, Jan Oliver; Rumpe, Bernhard: Synthesis of component and connector models from crosscutting structural views. In: FSE. 2013.

[MRR14] Maoz, Shahar; Ringert, Jan Oliver; Rumpe, Bernhard: Verifying component and connector models against crosscutting structural views. In: ICSE. 2014.

[OM15] OMG: OMG Systems Modeling Language (OMG SysML). Technical Report Version 1.4, 2015.

[Ri16] Richenhagen, Johannes; Rumpe, Bernhard; Schloßer, Axel; Schulze, Christoph; Thissen, Kevin; von Wenckstern, Michael: Test-driven Semantical Similarity Analysis for Software Product Line Extraction. In: SPLC. 2016.

[Ru96] Rumpe, Bernhard: Formale Methodik des Entwurfs verteilter objektorientierter Systeme. Herbert Utz Verlag Wissenschaft, 1996.

[Ru15] Rumpe, Bernhard; Schulze, Christoph; von Wenckstern, Michael; Ringert, Jan Oliver; Manhart, Peter: Behavioral Compatibility of Simulink Models for Product Line Maintenance and Evolution. In: SPLC. 2015.

[TMD09] Taylor, Richard N.; Medvidovic, Nenad; Dashofy, Eric: Software Architecture: Foundations, Theory, and Practice. Wiley, 2009.