Infrastructure to Use OCL for Runtime Structural Compatibility Checks of Simulink Models

Vincent Bertram[1] , Peter Manhart[2] , Dimitri Plotnikov[1], Bernhard Rumpe[1], Christoph Schulze[1] and Michael von Wenckstern[1]

Abstract: Functional development of embedded software systems in the automotive industry is mostly done using models consisting of highly adjustable and potentially reusable components. A basic pre-requisite for reuse is structural compatibility of available component versions and variants. Since each vendor in the automotive domain uses its own toolchain with corresponding models, an unified modeling notation is needed. For this reason based on a detailed feature analysis of well-established and commonly used modeling languages, a meta-model has been derived that allows checking structural compatibility, even between heterogeneous modeling languages.

Keywords: Automotive, C&C ADLs, OCL, Meta-Model, Simulink, Structural Compatibility

1 Introduction

Nowadays, the differentiation of vehicles will take place not just in body and interior design but also in embedded software systems like adaptive cruise control or road sign detection. The growing number of Advanced Driver Assisted Systems (ADAS) and their emerging variants and versions each using individual components cause an increase in software maintenance costs making up to 60% of total software costs [Gl01]. Reusing software components reduces development and maintenance costs. A basic precondition is component compatibility. Structural compatibility serves as a first indicator as it is an important prerequisite for full compatibility, which would also enclose behavioral compatibility [Ru15], also called refinement [Ru96].

First, the methodology and infrastructure to check structural compatibility will be introduced. In Sec. 3 a meta-model is proposed that is based on a detailed feature analysis of well-established and commonly used modeling languages and their constraints. In Sec. 4 the feasibility of the proposed overall approach is demonstrated using an ADAS model developed in the industrial research project SPES_XT. This paper finishes with related work in Sec. 5 as well as a conclusion and outlook in Sec. 6.

2 Method to Check Structural Compatibility

To define structural compatibility constraints at runtime, the first order predicate logic language OCL [Ob14] is chosen to describe meta-model constraints. One of its benefits

[1] RWTH Aachen University, Chair of Software Engineering, Ahornstraße 55, 52074 Aachen
[2] Daimler Research & Development Ulm, RD/EEC, Wilhelm-Runge-Straße 11, 89081 Ulm

is that it is reasonable efficiently executable. The *Java*-based derivate OCL/Programmable (OCL/P) is used; as it is based on an easier syntax and thus easier to read and understand [Ru11, Ru12] compared to OMG's OCL.

In order to express properties of widespread Component & Connector (C&C) architectures, a generic meta-model which is encoded as a Class Diagram (CD) was developed. CDs encapsulate the state and functions of objects in form of attributes and methods. They are well suited to define the meta-model describing the data structure of C&C architectures to check compatibility on. The semantics of a CD [HR04, CGR08] is a set of valid objects. Since the meta-model describes the signature of C&C models in a notation independent manner, every C&C model can be transformed into an Object Diagram (OD) instance being conform to the meta-model [MRR11]. OCL/P constraints are then used to define structural compatibility between two of these ODs.

The overall idea of the proposed infrastructure is to check compatibility using a Satisfiability Modulo Theories (SMT) solver. Thus, the meta-model, the used C&C model, and corresponding OCL/P constraints must be mapped to solver code. In a first step two *Simulink* [DH14] model components (SLC)s are transformed to ODs using the *MATLAB-Connector* API. Next the resulting ODs, the meta-model and the defined compatibility constraints are mapped to SMT code. Furthermore, the compatibility constraints are defined and stored in separated artifacts decoupling them from the C&C models. After merging all parts to one file artifact using a PartMerger [Gr15], Microsoft's Z3 solver [MB08] is invoked with the resulted SMT file. The solver can deal with uninterpreted functions to generate counter-example witnesses to show incompatibilities.

3 Meta-Model for Component and Connector Languages

In order to express OCL constraints for heterogeneous C&C architectures in a uniform way, this section defines an expressive meta-model. Existing meta-models were not capable to express all necessary aspects that were needed to check structural compatibility. Its syntax is derived from the results of an intensive analysis of the most important Architecture Description Languages (ADLs) used in the automotive domain. This section's outline is top-down: First, the most specific meta-model classes such as `FunctionComponentElement` and `FunctionComponent` are presented. Then, general classes like `TypeReference` and `Port` are described. Finally `DataType` and `Unit` are introduced.

FunctionComponentElement: The interface `FunctionComponentElement` is realized as composite design pattern to support hierarchical C&C models wherein the class `FunctionComponent` contains components and the class `PortConnector` associates two `Port` classes allowing directed communication from `source` to `dest` `Port` (c.f. Fig. 1a). The class `Interface` shown in Fig. 1b uses in- and out-ports for *direct* asynchronous communication. *Indirect* communication takes place over shared `GlobalStorage` elements.

TypeReference: The `DataType` and the `TypeReference` meta-model are adoptions from *EAST-ADL* [EA13] coupled with *SysML*. Properties being necessary for a concrete usage of a data type are split up for easier reuse. In order to be compatible with *Simulink*,

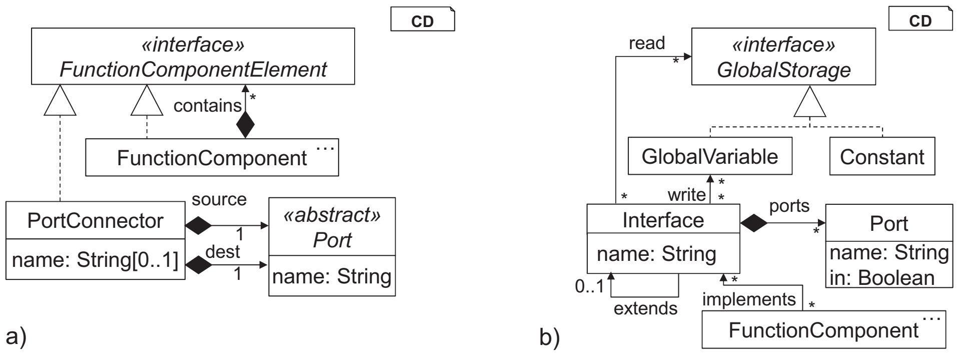


Fig. 1: Meta-Model of `FunctionComponentElement` and `FunctionComponent`

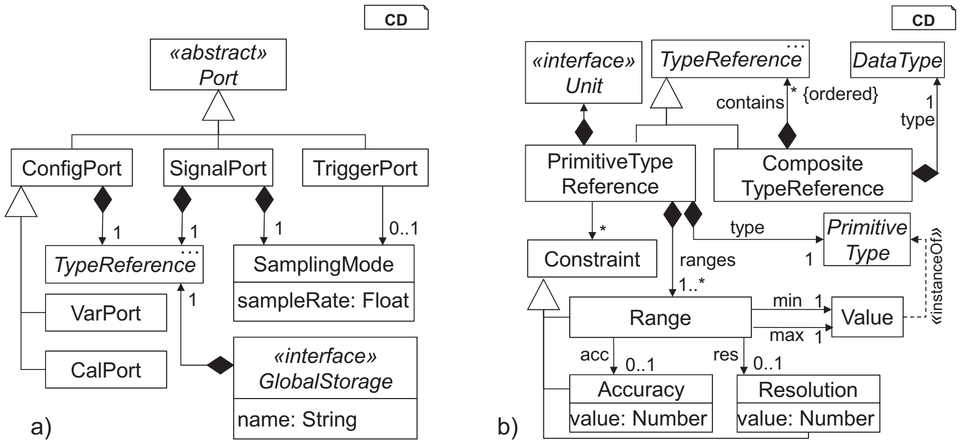


Fig. 2: Meta-Model of `Port`, `GlobalStorage` and `TypeReference`

changes were made on *EAST-ADL*'s homogeneous container having now a fixed amount of elements. A *Simulink* component that needs access to global variables is intentionally not supported by *AUTOSAR* [AU08], *EAST-ADL* and *MontiArc* [HRR12]. As a result *SysML*'s component interface model is used.

Port: The meta-model distinguishes three different kinds of `Port` classes (see Fig. 2a) based on the signal's purpose. This is similar to *AUTOSAR*. The class `TypeReference` of a `Port` defines the kind of signal content. Each `TypeReference` has at least one `Range` specifying the operative minimum and maximum (see Fig. 2b). If a signal has one range with higher accuracy (e.g. at low speed) than in another range (e.g. at high speed), there exists the possibility to define as many `Ranges` with its own `Accuracies` and `Resolutions` as necessary. The difference between `Resolution` and `Accuracy` is that `Resolution` represents the maximum delta how measured data can be stored and `Accuracy` represents the maximum error on stored data. Each `SignalPort` has additionally a `SamplingMode` describing the sample rate of a physical signal sampled into a digital one. If `TriggerPort` has a `SamplingMode`, an event can only arise at a specific time interval.

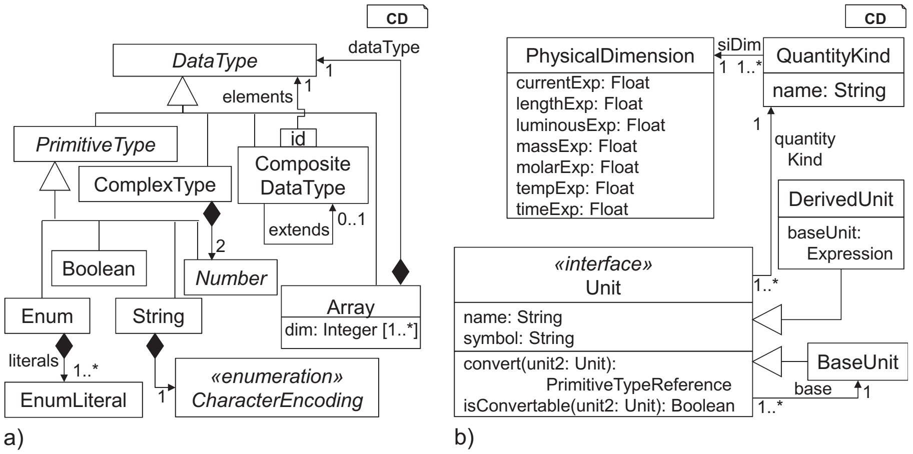


Fig. 3: Meta-Model of `DataType` and `Unit`

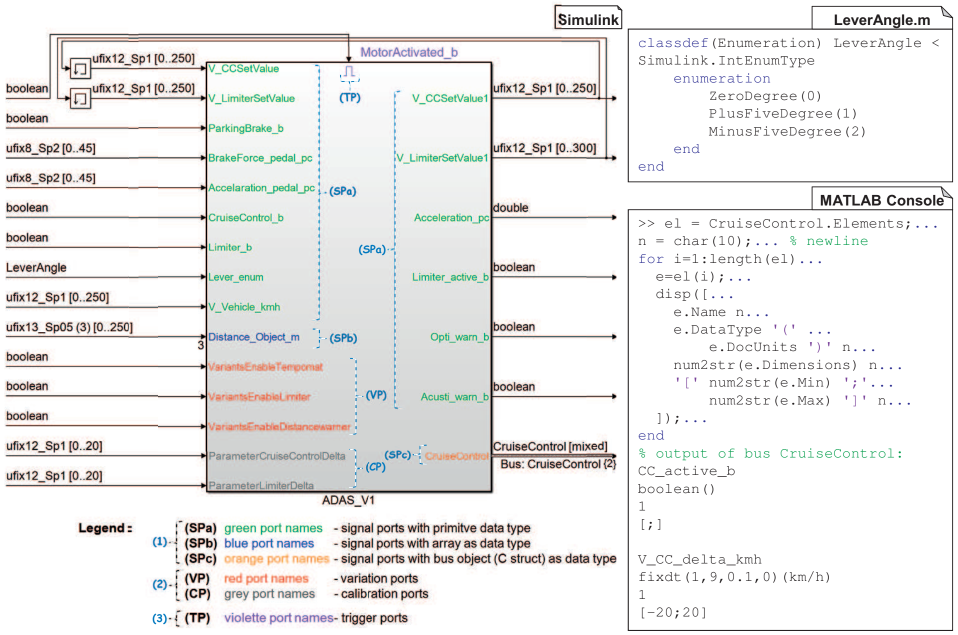
DataType: The class `DataType` shown in Fig. 3a is based on the type systems used by *Java* and *Simulink* where `ComplexType` represents complex numbers like $5+3i$, a `Composite-Type` is a heterogeneous container accessing its children by an unique `id` and an `Array` is a homogeneous container of a specified length. An `Enum` consists of finite set of `EnumLiteral` elements. Each `String` has a specific `CharacterEncoding`. Nested ports that are available in *SysML* are similar to ports having a `CompositeType` as `DataType`.

Unit: The interface `Unit` in Fig. 3b is based on the unit type system of *SysML* [Ha06] and *Modelica* [Mo12]. `Unit` has a `QuantiyKind` such as acceleration, energy or speed. Each `QuantiyKind` has a physical dimension defined in terms of basic units. Two different `QuantityKind` objects can have the same `PhysicalDimension`, e.g. *torque* (*Nm*) and *energy* (*J*) are different units, but have the same physical dimension $\frac{kg \cdot m^2}{s^2}$. The class `DerivedUnit` specifies how units with different magnitudes within the same `QuantiyKind` are converted by stating a conversion formula. Among other earlier mentioned ADLs, the proposed meta-model allows to create all existent units, supporting either the metrical (km/h) or empirical (mph) measurement system.

4 ADAS Component Demonstrating Methodology

This section starts with an overview of the evaluated *ADAS_V1* component and continues with the translation of structural information of SLCs into the proposed meta-model.

The provided ADAS model has four different levels of evolution and is realized as *Simulink* model which is a special case of an C&C architecture software mainly but not only used in the automotive domain. This section depicts the structure of the *ADAS_V1*'s top-level SLC only. The running example is simplified and does not represent a 100% real world model, but it provides enough structural information without being overloaded with unimportant



LeverAngle.m

```
classdef(Enumeration) LeverAngle <
Simulink.IntEnumType
    enumeration
        ZeroDegree(0)
        PlusFiveDegree(1)
        MinusFiveDegree(2)
    end
end
```

MATLAB Console

```
>> el = CruiseControl.Elements;...
n = char(10);... % newline
for i=1:length(el)...
  e=el(i);...
  disp([...
    e.Name n...
    e.DataType '(' ...
        e.DocUnits ')' n...
    num2str(e.Dimensions) n...
    '[' num2str(e.Min) ';'...
        num2str(e.Max) ']' n...
  ]);...
end
% output of bus CruiseControl:
CC_active_b
boolean()
1
[;]

V_CC_delta_kmh
fixdt(1,9,0.1,0)(km/h)
1
[-20;20]
```

Fig. 4: Syntactic Interface Description of *ADAS_V1* developed in the project SPES_XT

details. The whole SLC interface is defined as the set of all in-, out-ports as well as used variables being shared outside this component.

The SLC interface of the *ADAS_V1* as shown in Fig. 4 is clearly evident, because it is completely covered by all visible in- and out-ports. This example consists of three different port types: (1) signal, (2) configuration, and (3) trigger ports. Signal ports can be grouped using arrays or non-virtual buses and have been divided into three subtypes: (SPa) for one single primitive type, (SPb) for grouped signals having the same data type, and (SPc) for grouped signals of different data types. Due to different purposes for using a configuration port, it is divided into two subtypes: (VP) for software variation (enabling or disabling features) and (CP) for calibrating subcomponents with parameters. The trigger port (TP) has only one kind. All port types (CP, SP, TP, VP) extend the port concept defined in *AUTOSAR*. The ports' data types with its ranges together are displayed directly on the signals connected to them; `ufix12_Sp1 [0..250]` stands for unsigned fixed point data type of word length 12 and having a slope (resolution) of 0.1 and a value range between 0 and 250.

In the example SLC of *ADAS_V1*, the port `Lever_enum` is an enumeration of `LeverAngle` and contains three values representing the deflecting angle of the lever: `ZeroDegree`, `PlusFiveDegree`, and `MinusFiveDegree` (c.f `LeverAngle.m` in Fig. 4). Non-virtual data types represent a semantic union. The non-virtual data type `CruiseControl` groups the boolean signal `CC_active_b` and the fix-point number signal `V_CC_delta_kmh` having a $[-20;20]$ range together to a non-virtual bus (c.f. `MATLAB Console` in Fig. 4).

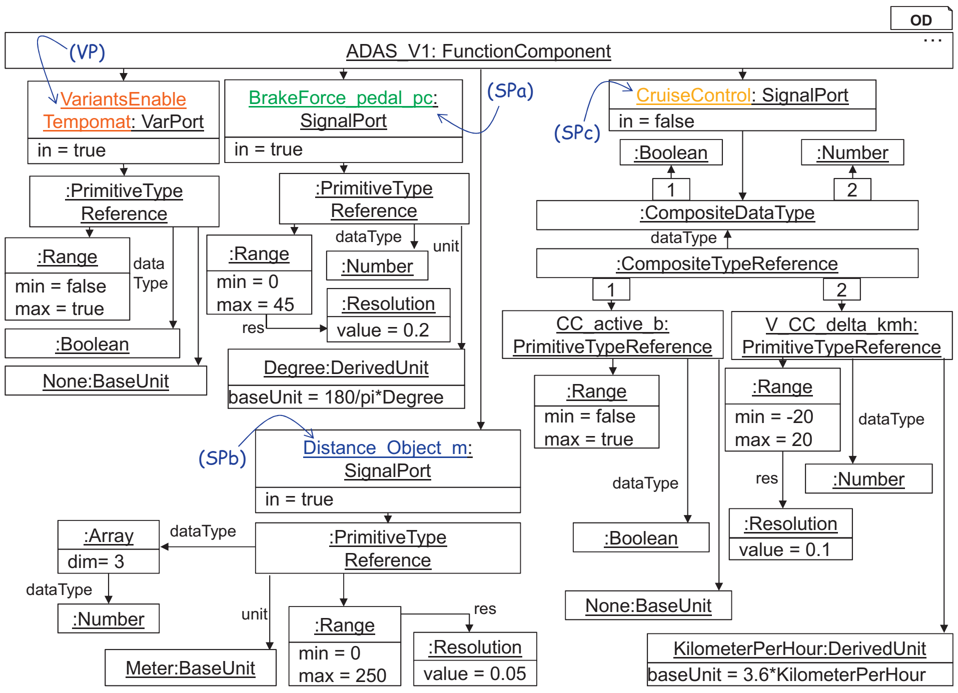


Fig. 5: Example meta-model instantiation of *ADAS_V1* Limiter component

OCL/P

```
def Boolean infix (Number v) isin (Range r) is:
  result =
    v >= r.min &&
    v <= r.max &&
    (~r.res ||
      (v - range.min) % range.res == 0)
```

(Range *r* has no optional association *res* to Resolution)

Fig. 6: OCL code snippet and SMT function defining whether `Number` v belongs to a given `Range` r

To show an example instantiation of the meta-model, a subset of *ADAS_V1* is chosen and shown as an OD in Fig. 5. The excerpt uses 3 of 15 in-ports (`VariantsEnableTempomat` (VP), `BrakeForce_pedal_pc` (SPa), `Distance_Object_m` (SPb)) and 1 of 7 out-ports (`CruiseControl` (SPc)) containing all port types already shown in Fig. 2. The classes `CompositeDataType` and `CompositeTypeRefence` use numbers instead of names as identifiers as this allow an easier iteration over the elements in the used OCL constraints.

The code shown in Fig. 6 defines the semantics for the overloaded `isin` operator in OCL. This operator returns the `Boolean` value `true` when the `Number v` can be found in the given `Range r`, which is specified by a minimum (`min`), maximum (`max`) and a resolution (`res`). As an example a range with the properties `min=1.0, max=4.1, res=0.5` would return `true` only for the values `1.0, 1.5, 2.0, 2.5, 3.0, 3.5` and `4.0`, which is defined in the lines 4, 5 and 7. Line 6 is only necessary because the `res` association is optional. If it is not defined, the range will contain all values between `min` and `max`.

5 Related Work

MontiArc [HRR12] is a C&C ADL where asynchronous message based communication is done over directed connectors between typed component ports. It uses context conditions for structural consistency checks such as data types and input/output directions of ports. Context conditions extend the expressiveness of grammar based approaches and enable checking of constraints, e.g. type systems, which are not expressible through grammars.

The approach presented in [Bh11] tackles the problem of defining and evaluating consistency relation between architectural views imposed by various heterogeneous models and a base architecture for the concrete system model. The consistency check happens on different abstraction levels if the concrete architecture model is consistent to the more abstract and less detailed abstract view. Instead, this approach checks compatibility on the same abstraction level.

The approach from [Da14] presents an architectural framework where multiple views on one system can be defined. The consistency is checked by leaving out details unnecessary for comparison, which is termed as a “lifting” operation. It lifts the more detailed software view to the more abstract functional view. This lifting operation causes valuable information loss, so the architectural framework is insufficient to check the structural compatibility as described in Sec. 2. The *SysML* meta-model which is used, supports only a subset of features available in the meta-model presented in Sec. 3.

6 Conclusion and Outlook

This paper has presented a first overview of an infrastructure for compatibility constraint checks using OCL. It was exemplified using a model from automotive domain but it is not limited to only this. Comparatively *MATLAB Simulink* is also used in other domains like aerospace and medical engineering. The main contribution is the proposed meta-model which is based on an intensive analysis of well-established modeling languages to support heterogeneous C&C architectures. It includes all meta-elements of the most known industrial meta-models (SysML, Simulink, Modelica, AUTOSAR, EAST-ADL). All of these ADLs can be mapped and as a result only a meta-model transformation must be written.

While this paper deals with the modeling aspect of the proposed infrastructure, future work will give more information about the generative part. This will contain how to generate SMT code for the Z3 solver using OCL/P constraints. The solver results will be presented as user-friendly error messages describing the reason for constraint violations based on counter-example witnesses. The highly modular infrastructure is capable of supporting new third party plug-ins and consequently allows a seamless integration into industrial development environments. To demonstrate the extensibility of the proposed infrastructure further modeling notations will be implemented.

References

[AU08] AUTOSAR: SW-C and System Modeling Guide. Technical report, 2008.

[Bh11] Bhave, A.; Krogh, B.H.; Garlan, D.; Schmerl, B.: View Consistency in Architectures for Cyber-Physical Systems. In: ICCPS. IEEE, 2011.

[CGR08] Cengarle, María Victoria; Grönniger, Hans; Rumpe, Bernhard: System Model Semantics of Class Diagrams. Technical report, TU Braunschweig, 2008.

[Da14] Dajsuren, Yanja; Gerpheide, Christine M.; Serebrenik, Alexander; Wijs, Anton; Vasilescu, Bogdan; van den Brand, Mark G.J.: Formalizing Correspondence Rules for Automotive Architecture Views. In: QoSA. ACM New York, 2014.

[DH14] Dabney, James B.; Harman, Thomas L.: Mastering Simulink. Prentice Hall, 2014.

[EA13] EAST-ADL: EAST-ADL Domain Model Specification. Technical report, 2013.

[Gl01] Glass, Robert L.: Frequently Forgotten Fundamental Facts About Software Engineering. IEEE Software, 2001.

[Gr15] Greifenberg, Timo; Hölldobler, Katrin; Kolassa, Carsten; Look, Markus; Mir Seyed Nazari, Pedram; Müller, Klaus; Navarro Perez, Antonio; Plotnikov, Dimitri; Reiss, Dirk; Roth, Alexander; Rumpe, Bernhard; Schindler, Martin; Wortmann, Andreas: A Comparison of Mechanisms for Integrating Handwritten and Generated Code for Object-Oriented Programming Languages. In: Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development. 2015.

[Ha06] Hause, Matthew: The SysML modelling language. In: SysCon. 2006.

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

[HRR12] Haber, Arne; Ringert, Jan Oliver; Rumpe, Bernard: MontiArc - Architectural Modeling of Interactive Distributed and Cyber-Physical Systems. Technical report, RWTH Aachen, 2012.

[MB08] Moura, Leonardo; Bjørner, Nikolaj: Z3: An Efficient SMT Solver. In: TACAS. volume 4963 of LNCS. Springer, 2008.

[Mo12] Modelica Association: Modelica - A Unified Object-Oriented Language for Systems Modeling. Technical report, 2012.

[MRR11] Maoz, Shahar; Ringert, Jan Oliver; Rumpe, Bernhard: Modal Object Diagrams. In: ECOOP. volume 6813 of LNCS. Springer, 2011.

[Ob14] Object Management Group: Object Constraint Language, Version 2.4. Technical report, 2014.

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

[Ru11] Rumpe, Bernhard: Modellierung mit UML. Springer, 2011.

[Ru12] Rumpe, Bernhard: Agile Modellierung mit UML. Springer, 2012.

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