(Hrsg.): Modellbasierte Entwicklung eingebetteter Systeme,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 1

Between Academics and Practice: Model-based Development of Generic Safety-Critical Systems

Daniel Schwencke[1], Hardi Hungar[2], Mirko Caspar[3]

Abstract: Academically, the utilization of models promises a quite precise and formalized description of system behavior already in early design stages. Practically, there are gaps between the necessary and provided capabilities of formalism, tools, and processes. Based on experiences in modeling a safety critical railway Radio Block Center, this paper presents the modeling method and experiences from the development of the model. It discusses the questions of how to validate the model, how to derive verification and testing schemes, and how to verify the conformance of implementations to the model. Gaps between academic needs and practical reality are discussed. Hence, this work aims to focus the further development of model-based approaches in the safety-critical embedded domain.

Keywords: Model-based Development, Verification, Code Generation, Railway, Safety, SysML

1 Introduction

Already in early stages of system development models can be used for capturing system behavior. The formal or at least semi-formal nature of models allows for reaching a high precision in the formulation. Additionally, it constitutes the basis for a number of techniques which can support or even automate design activities. In particular this includes verification, validation and code generation.

In the article at hand, aspects of modeling a radio block center (RBC) are presented. Being part of the European Train Control System (ETCS), an RBC constitutes the radio interface between train and interlocking (see Fig. 1). The work aims at the creation of a system which can serve as a reference of an RBC in the ETCS: on the one hand the model shall be utilizable in product specifications and substantially support their completeness and correctness, as demanded e.g. by the German new type approval ("Neue Typzulassung" or NTZ, cf. [Su11]). On the other hand, after annotation with program instructions and addition of further components, program code can be generated from the model. So far, the basic RBC functionality has been modeled so that simple operational scenarios can be managed by the generated and executed RBC code.

[1] Deutsches Zentrum für Luft- und Raumfahrt e.V., Institut für Verkehrssystemtechnik, Lilienthalplatz 7, 38108 Braunschweig, daniel.schwencke@dlr.de
[2] Deutsches Zentrum für Luft- und Raumfahrt e.V., Institut für Verkehrssystemtechnik, Lilienthalplatz 7, 38108 Braunschweig, hardi.hungar@dlr.de
[3] Deutsches Zentrum für Luft- und Raumfahrt e.V., Institut für Verkehrssystemtechnik, Lilienthalplatz 7, 38108 Braunschweig, mirko.caspar@dlr.de

A particular challenge for employing model based techniques is the generic nature of systems like the RBC. Such generic systems are to be instantiated to the particular track layout and control equipment of the area where they are be installed. Thus, the models have to represent mere patterns of concrete facilities. In the case of the model at hand, this has been achieved so that the generated code can be configured for a particular railway line, becoming an implementation. The implementation in turn can be validated in simulations and then provides a behavioral reference—for that configuration. This does not yet solve the problem of general verification and validation. This is addressed by elaborating the questions in more detail and presenting approaches how to proceed adequately.

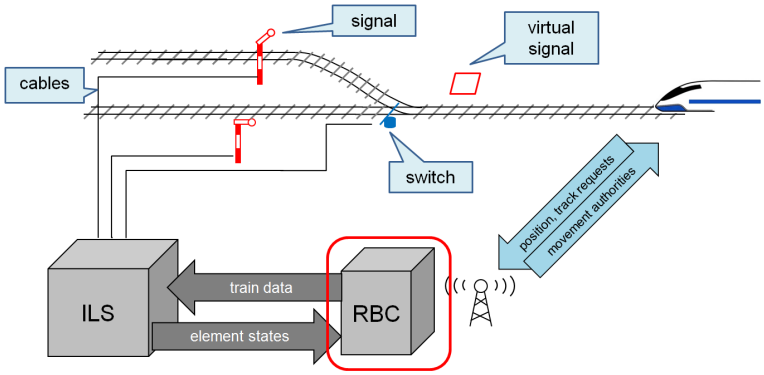

Fig. 1: The RBC as part of the ETCS level 2/3 (simplified presentation, ILS = interlocking system)

The model development is related to activities of the German railway infrastructure manager DB Netz AG and further European infrastructure managers towards the standardization of interfaces in the interlocking area (the DB project "Neuausrichtung der Produktionssteuerung", NeuPro for short, and the "European Initiative Linking Interlocking Subsystems", EULYNX for short, see [14b] for the latter). Results from those activities could already serve as patterns for parts of the model. A more detailed look into the model than can be provided here has been given in the article [SH16] which focuses on the ILS interface of the RBC.

In the remainder of this article, first an overview of the general framework of the model development (relevant specifications, languages and tools) is given in Section 2 (taken from [SH16]). In Section 3, the need of generic models in railway signalling is motivated and our approach and directions for improvement are discussed. Experiences regarding the usage of state charts and manually written code are discussed in Section 4. An elaboration on the challenges of validation and verification in the pursued model-based approach is presented in Sections 5 and 6. Section 7 concludes with a presentation of future plans and a preliminary evaluation of the model-based approach.

2 General Framework of the Modeling Activities

2.1 Relevant Specifications

The ETCS is being specified by the EU Agency for Railways (formerly European Railway Agency), the Union Industry of Signalling (UNISIG) as well as the European Rail Traffic Management System (ERTMS) Users Group in a variety of documents, the so-called subsets. The central document is the system requirements specification [14e], which is complemented by various specifications for single system components and interfaces. For the RBC, the functional interface specification for the handover of trains between RBC [14f] is of importance. A specification of the RBC as such however is not part of the subsets but left to the respective manufacturers and national infrastructure managers, for which they have to take into account the European specifications.

ETCS is used in different countries to different extent and with different parametrizations. In Germany this is laid down in the specification [14a] of DB Netze. Finally, as part of the NeuPro activities, DB Netze specified the interface between an electronic interlocking system (ILS) and an RBC, called Standard Communication Interface RBC (SCI-RBC) [14d]. Such national specifications are of importance for the manufacturer of an RBC for the respective country.

The specifications differ considerably in their abstraction level: even the European specifications range from functional requirements over system states and procedures to concrete data formats and values, depending on the importance of a system component for European interoperability. As a basis for modeling activities, a detailed and well-structured system specification is advantageous. Whereas most of the aforementioned specifications mainly consist of textual requirements, the SCI-RBC already is of semi-formal nature. Besides explanatory text, here the mandatory behavior of the interface is specified in SysML state diagrams.

2.2 Approach

The DB specification SCI-RBC largely uses SysML (Systems Modeling Language) diagrams and thus represents a good starting point for modeling the RBC. Besides other types of diagrams mainly state machines are used. They bindingly specify the behavior defined by the semantics of the diagram elements, but not the use of such diagrams or their architecture in the further development. The system actions in the state machines however are only given in textual form and are not yet formalized precisely. Consequently it is not possible to generate code directly from those state machines. Furthermore some requirements which are not expressible in diagrams remain in textual form.

For the formal modeling of the RBC it seemed sensible to use SysML as well: besides being able to stay close to the specification, SysML has the advantage of being widespread and

having rich support by modeling tools, code and test case generators. As target language for code generation from the model an object oriented language is favorable, enabling the configuration of a (generic) RBC for different railway infrastructures by means of instantiation of classes; here C++ was chosen. Regarding a suitable tool for modeling and code generation, PTC Integrity Modeler (through 2014: Atego Modeler) together with PTC Automatic Code Synchronizer (formerly Artisan Studio Automatic Code Synchronizer) was identified. The Code Synchronizer supports code generation for SysML block definition diagrams/internal block diagrams and SysML state machine diagrams.

3 Creating a Generic Model

3.1 The Complexity and Diversity of Railway Signaling Systems

Classically, the central component of railway signaling is the interlocking system (ILS). It sets and secures the routes for the trains within the area it manages. In order to do so, it needs to be connected to up to several hundreds of field elements in that area – e.g. signals, switches, axle counters and level crossings. For train operation with ETCS Level 2, the additional component RBC (Fig. 1) enters the picture, which manages GSM-R (Global System for Mobile Communications — Rail(way)) radio connections to a number of trains currently in an area that typically is wider than an ILS area. Thus it is connected to several ILS which provide information necessary to generate the so-called movement authorities for the trains. Other less central interfaces to command and control workplaces and neighbor RBC shall be ignored in the sequel. Since the RBC obtains state information from the ILS (regarding signals and switches) and from the trains, it needs to manage all those elements internally.

Typically, within one RBC area the signaling equipment differs. Different ILS areas have been equipped by different manufacturers, and even within one ILS area typically there are several variants of signals due to cost reasons (only the signal functionality needed is installed for a certain signal with a certain purpose). The variety can be enormous and is inherited from the operational scenarios which need to be realized by the infrastructure.

In order to manage train operations, the RBC needs to know about certain aspects of that variety. Fortunately, for the ILS side including signals and switches, these aspects have been identified by DB in the interface specification SCI-RBC [14d]. For the train side the variants can be derived from the ETCS specification [14e] which defines a train data packet which is sent to the RBC after successfully establishing a communication session between a train and the RBC.

The signaling equipment manufacturers handle the above sketched complexity by developing *generic products* (safe computers, signal controllers, axle counters, etc.) which are grouped to form a *generic application* (e.g. an electronic interlocking, which is here understood to include the field elements and their connections). For each real system that is installed

(the *specific application*), the generic application is *configured* according to the track and route layout (see e.g. [07]). This comprises the choice, number and variants of the generic products needed as well as the design of the communication and energy network for them.

Classically, the communication of signaling components is based on the manufacturer's proprietary message formats and protocols. Meanwhile DB specified a standard protocol [14c] for IP-based communication for the ILS interfaces; the GSM-R radio communication for the RBC-train interface is laid down in the ETCS-specifications [15]. However, a model for signaling equipment testing purposes should allow for flexible communication mechanisms anyway to be ready for connecting to stubs or to laboratories (on-site or remote).

Finally, the RBC internal passing of information needs to be flexible, too: some messages from the ILS regard single elements, some regard certain groups of elements (group outage telegrams for signals and switches).

3.2 Modeling Approach

The complexity and diversity discussed above is mirrored in the RBC model. It comprises RBC internal representations of ILS areas, signals, switches and trains. Aspects and states of these elements are relevant for the RBC and, hence, need to be synchronized between RBC and ILS.

Generally, the approach for the RBC-internal representations of elements whose number depends on the configuration is the use of classes in the model so that they may be instantiated as often as needed at runtime (theoretically, an earlier instantiation in the specification, in the model or in the code is possible but makes no sense due to the repeated work for each RBC and bad maintainability). Currently, the instantiation code (see Listing 1) is provided in a separate configuration file which is referenced by the model.

```
//Add ILS area to RBC
ILSArea * ilsAreaHBS = addILSArea(mLogger);
ILSArea * ilsAreaHGLI = addILSArea(mLogger);

//Add points to ILS area
PointElement * w1 = ilsAreaHGLI->addPoint(3434343, "W1");
PointElement * w2 = ilsAreaHGLI->addPoint(3434343, "W2");

//Create track topology
Track * t1 = mInfrastructure.addTrack(5000);
Track * t2 = mInfrastructure.addTrack(150);
mInfrastructure.connectTrackParts(w1, 1, t2, 0);
mInfrastructure.connectTrackParts(t2, 1, w2, 1);
```

```
//Add signals to ILS area
ilsAreaHBS->addSignal(5656565, "30N2", 0, t1, 4850, 1);
ilsAreaHBS->addSignal(3434343, "Bk4141", 0, t4, 21170, 1);

//Add balise groups to RBC infrastructure
mInfrastructure.addBaliseGroup(96, 201, 1, 3, 0, t1, 1980, 1);

//Add gradient profile to tracks
t1->addGradient(2, true, 0, 1);
t1->addGradient(3, false, 500, 1);

//Add basic speed profile to tracks
t1->addBasicSpeed(12, 0, 1);
t1->addBasicSpeed(12, 5000, -1);
```

List. 1: Example configuration code for the RBC

For the realization of variants in the model, different approaches have been exploited: in one case where only two variants existed, those have been modeled separately. In another case, Boolean or enum-type constants/variables guarding state chart transitions and code segments which belong to the respective variants have been used. In a third case, an abstract method together with a standard implementation has been provided which might be replaced for different variants.

Flexibility regarding the communication mechanism has been realized via generalization as well (abstract methods for connection set-up, (dis-)connecting, send/receive, connection status). This way arbitrary implementations for different communication mechanisms can now be developed without changing the model. To keep purely code-written variants which regard the model interfaces out of the model is a good principle anyway: in the RBC model, it has also been applied to an optional train message version converter module. This tool can just be activated or deactivated by setting a Boolean constant in the model, otherwise the converter is external to the model.

Message broadcasting has been realized by providing an abstract field element class from which signals and switches inherit group/subgroup IDs as well as a group outage event (in form of a method). The broadcast, containing the target group/subgroup IDs, is passed to each element; comparing with its own IDs the element decides whether it accepts or rejects the message.

3.3 Discussion and Possibilities for Improvement

Having available a class/instantiation concept for modeling is essential since many systems contain multiple instances of some elements. However, especially in models of railway

signaling systems which need to be configured according to the local track infrastructure and the intended routes, it is desirable to have the possibility to specify at least an initial configuration of instances in a separate (part of the) model. This should include the choice of the element variant and the setting of initial parameters and might be extended to a mechanism for the dynamic instantiation and destruction of elements during runtime.

Regarding the modeling of variants there exist several options such as

- each variant separately,
- through child classes of a parent class (inheritance),
- Orthogonal Variability Model (OVM) including assignment of transitions/operations to one or several variants, or
- Boolean/enum constant which is set during instantiation and evaluated as precondition to the execution of transitions or code.

Some have been used in the RBC model as explained in Section 3.2. The experience from that modeling activity is that each case needs to be considered separately and that the decision for a modeling option depends on

- the number of variants,
- what varies (attributes, behavior),
- how much it varies (how many model elements are concerned),
- the relation of the variants (orthogonal, overlapping, excluded/enforced combinations), and
- the need for dynamic change of the variant.

Generally, the use of variant diagrams (OVM) is desirable in order to quickly identify variation points and variants (except for the modeling through inheritance which usually is captured diagrammatically already). Those model alternatives [RF14] and describes one aspect of a class or block having different or extended detailed behaviors. OVM can be realized in SysML by block diagrams or extended profiles. The PTC Integrity Modeler supports such diagrams; however, the code generator does not. So if code generation is desired as in the case of the RBC model, a different approach based on elements supported by the code generator needs to be taken. Unfortunately, the limited number of diagrams and elements for which code can be generated is a problem encountered in several situations, cf. Chapter 4.

The use of generalization/abstract methods for creating flexible interfaces of the model has been quite a positive experience. Of course this mechanism has been designed for such

purposes in object-oriented programming (code/code interfaces), but here it has proved well-suited for model/code interfaces as well.

Finally let us remark that for model internal message broadcasting there are several approaches which range from a central to a distributed logic in order to identify the addressed elements. They differ in performance, distribution in the model, their impact on the model structure and the extent to which they are captured in diagrams. The decision for an approach must also exhibit a solution which reasonably supports all message flows to the single elements, e.g. additional direct forwarding of messages to an element. But since message broadcasting is a topic in itself which, we shall not discuss this further here.

4 Diagrams versus Handwritten Code

4.1 Use of Diagrams in the Model

Both structure and behavior of a system can be modeled by diagrams. SysML provides different types of diagrams for both categories. However for the RBC model we were restricted to Block Definition Diagrams/Internal Block Diagrams for structure and State Diagrams for behavior since they are supported by the code generator (the PTC Integrity Modeler Automatic Code Synchronizer). Additionally, package diagrams may be used but provide not much advantage since a package hierarchy is a relatively simple structure which can be overseen well in the classic folder view.

To model the structure of a system via blocks and their dependencies seems possible for nearly every system. Even if there are no dependencies at all, blocks can be used just to group operations which are manually coded. But of course the benefits of modeling comes with capturing complexity in diagrams. The RBC model contains block definition diagrams ranging from such simple grouping functionality (e.g. for operations assembling message packets) to quite complex structures, the most complex one being the RBC internal infrastructure image (including the track topology, different field elements and track profiles).

The model based approach makes even more sense where behavior is captured diagrammatically as well. It does however not always make sense to use state charts – our experience with the PTC Integrity Modeler is that ideally those should only be drawn where there are block internal states which rule the block's behavior. This is because in order to make states permanently available block externally extra variables and actions need to be foreseen which duplicate the state and clutter the state diagram; but sometimes such drawbacks simply need to be accepted. In the RBC model state charts primarily model the behavior of the most important RBC internal images of external elements (ILS, signal, switch, train; for an example see Fig. 2) whose state changes are reported to the RBC through incoming messages. Furthermore, the central RBC logic which connects train- and ILS-side is modeled in state charts which forward events to the concerned elements.

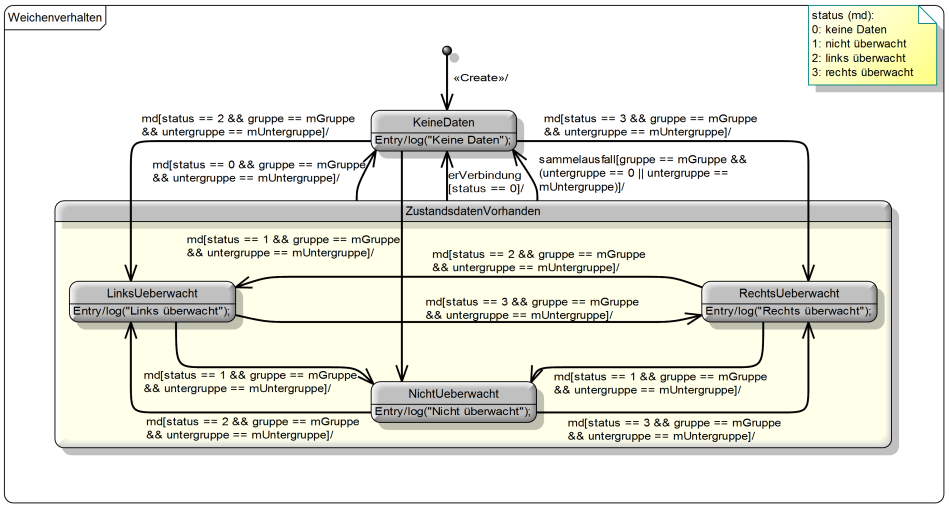

Fig. 2: SysML state diagram for the logical switch element of the RBC

4.2 Use of Direct Code in the Model

As indicated in the previous section, it does not always make sense to capture behavior by state charts. In addition to the above discussion, also "algorithmic" behavior which consists of calculations or depends on complex data structures is not suited for being modeled in state charts; it is formulated easier and more compact in program code. In other cases, the missing support of diagram types or of code generation of certain statements enforces a direct implementation. In the RBC model, handwritten C++ code was added mainly for the following purposes:

- main program loop, initialization and configuration code;
- add/remove methods for objects to/from lists or variables (instantiation/destruction of objects is not supported by code generation);
- iteration over instances of a class;
- central algorithms such as traversing the infrastructure image;
- algorithms for (de-)coding messages including parsing and integrity checks;
- mapping between message data and events (method calls) or storing variables, possibly including data conversions and get/set methods; and
- logging functionality.

4.3 How to Improve

Generally it is desirable to capture more aspects of a system by diagrams than could be achieved in the case of the RBC model. This could further increase comprehensibility, maintainability and automation in system development. An exception may be those cases which are at a very fine-grained level where a sequence of statements is easier written down and overseen than putting it into a diagram. Concrete ideas towards a "more model-based" approach include:

1. For a wider applicability of state charts, a state publication/inquiry feature should be added to the model and code generation in order to easily and permanently provide the current state locally/globally.
2. Variant diagrams to create orthogonal variability models are already supported by the PTC Integrity Modeler. What is missing is the code generation capability.
3. For providing (initial) system configurations by instantiation of blocks and setting block properties, object diagrams may be used. They are available in the Integrity Modeler as well, so a generation of a configuration code seems feasible and should be added.
4. Control flow diagrams can be used for modeling the structure of algorithms. They should be provided for modeling operations, and also to generate code from them should not be difficult.

Another possibility for more formalization (not necessarily by means of a diagram) might be a structured management of input/output inside the model: from our experience with repeatedly handwritten logging messages the idea would be to allow for assigning a block to a logger which organizes message types, messages, message formats and output channels. Then a message could be inserted using drag and drop in different places in the model and the correct output statement could be generated. Finally, the generation of get/set methods for variables or even better, other easy means of accessing a model element from other parts of the model would be helpful.

5 Verification of the Model

The RBC model presented in this paper shall be a reference model that can be used as base for a concrete implementation, as test oracle, or even as base for deriving test cases. Naturally, this model - or its representation in a formal tool - needs to be checked extensively. Using formal verification techniques is an obvious approach since state machines are used for modeling the system behavior. However, these concepts are not in the focus of this paper but the question of model based verification and testing.

5.1 Instances, Projections, Coverage

One challenging question is how the formal reference model can be used for the verification of implementations. The aim is to reach a full test coverage, i.e. having test cases that cover each state transition defined in the reference model. Dynamic testing is a common concept for black-box verification. Here two aspects are relevant: the definition of model instances and the derivation of test cases. Whereas the second aspect is subject to intensive research and development, questions of model instantiation and configuration as well as the influences on the test coverage have not been analyzed in detail yet.

As described in Section 3.1, the RBC model is generic. The state machines describe the behavior for different abstract subcomponents, e.g. a light signal. All possible behavioral combinations for all types of signals are modeled (e.g. see requirement 1335 in [14d], p. 25). A generic implementation needs to realize all parts of it since the specified functionality is mandatory and a generic type approval can be reached. But practically, there might be no physical instance of this component that uses all behavioral combinations. It might be configured for a concrete usage. E.g. a station entry signal has other signal aspects than a block signal.

Hence, the question is what parts of the model can be tested by which configured instance (projection is the common term in railway sector) of its implementation. The number of combinations of necessary instances may become very high if several components of complex models are considered and a full coverage of model transitions needs to be reached. Practically in railway domain, the generation and set-up of each projection is extensive and expensive. Accordingly, the optimization of necessary and useful projections helps to decrease the verification costs.

A formal representation of the model and the implementation variations is necessary in order to optimize the set of configurations automatically. This may contain information about instantiation, addressing, inheritances, alternative functionalities, and relations between instances. As stated in Section 3.2, SysML can express only some of these aspects, e.g. the inheritances. In literature in industry, additional concepts exists in order to manage variabilities in modeled systems. Feature Modeling is one of these approaches [LKL02]. Features of a system are categorized and formalized. This concept is mainly used for product variation management. Another approach to manage alternatives in models or implementations is the Orthogonal Variability Modeling (OVM), see Section 3.3 above.

As mentioned, OVM diagrams are used in the RBC specification [14d] but cannot be evaluated formally yet. For verification purposes, the OVM can be extended by a relational semantics that helps to automatically derive excluding combinations of variations. It is also possible to construct a set of valid variations if the semantics offers according constructs.

However, variants are not the only aspects that influences the test coverage and the test case selection. These may result not only from the modeled system but also from the

implementation (e.g. by code generation). An obvious example is the concept of instantiation and identification of concrete objects of the same class.

Currently we are investigating possibilities of automated projection optimizations in the railway domain, among others on the presented RBC model. A practical problem is how the model can be accessed if commercial modeling tools are used.

5.2 Dynamic Test of the RBC Model

Dynamic tests were conducted for the proposed model – in particular for its ILS interface – in order to verify the model and additional manually coded components. All tests are based on the execution of the program code obtained after code generation and compilation. To this end, different calls of test routines were added to the generated code in the main function before compilation.

In a first step, telegrams and routines have been added to a separate package of the model for testing the telegram parsing, the telegram data checking and the composing of telegrams.

As a second step, further telegrams and routines have been added in order to test the telegram processing of the single interface parts. This includes positive testing for the most common telegram messages as well as tests of erroneous situations, where the selection of the latter was done along the lines of the SCI-RBC test specification [13]. For that second step both interface parts have been configured according to a fictitious station and the expected behavior was compared to the states logged and error messages produced during execution.

Thirdly, in order to test whether the whole interface exhibits the intended behavior, external events have been simulated and the logged messages of both interface parts have been examined.

One basic experience of the conducted tests is, that the combination of automated and handwritten code is a main source of errors. Accordingly, the discussed improvements in Section 4.3 may help to improve the design process.

Finally, a validation of the RBC model was conducted by means of coupling it via TCP/IP with the Rail Simulation and Testing (RailSiTe) laboratory of the German Aerospace Center (DLR), which acted as ILS and as train.

6 Towards Using the Model for Verification of Implementations

6.1 Refinement Relations

The task of proving the correctness of an implementation is an instance of the well-known problem of establishing a refinement relation. Two systems $S_1 = (I, O, V_1)$ and $S_2 = (I, O, V_2)$

having the same external interface of input and output variables I and O (or channels, or any other name) but different internals V_1 and V_2 are behaviorally equivalent if

$$[\![S_1]\!]|_{I,O} = [\![S_2]\!]|_{I,O} \ , \tag{1}$$

i.e., if their behaviors agree on the external variables. Refinement means behavioral inclusion:

$$[\![S_2]\!]|_{I,O} \subseteq [\![S_1]\!]|_{I,O} \ . \tag{2}$$

in the formulas above, $[\![S]\!]$ may denote a set of traces as the semantics of a system S. Given a domain for time $\mathcal{T}$ and one for variable values $\mathcal{V}$, the set of traces $\mathcal{S}$ are the mappings

$$[\mathcal{T} \to [\mathcal{X} \to \mathcal{V}]]$$

which assign to each point in time a variable valuation. Equivalently, we can view each variable semantically as a sequence of values, arriving at

$$[\mathcal{X} \to [\mathcal{T} \to \mathcal{V}]]$$

as semantical domain, and treat the semantics of a system like a logical formula. Then, refinement is expressed by the logical formula

$$\exists V_2 \,.\, [\![S_2]\!] \Rightarrow \exists V_1 \,.\, [\![S_1]\!] \ . \tag{3}$$

This is essentially the form in which Lamport [La91] studied refinement in his "temporal logic of actions".

To actually prove that S_2 is a refinement of S_1, the most common practical way is to define a mapping from the internal variables of S_2 to those of S_1 (or, more general, a relation between them) and show that this mapping induces a simulation. It may be said that this "practical way" is already rather difficult. And there are even more general forms of refinement, cf. [Br97], which are not easier to prove.

For a discussion of the problem at hand, the notion expressed in (3) can be taken as a preliminary definition.

6.2 Refinement in Practice: Coping with Genericity

Having to show the correctness of some claimed refinement step is not a new concept in safety-critical system development: Each step towards an implementation has to be verified. For instance, when the architecture of a software is defined and the specifications of its components are designed, it has to be shown that the software requirements are correctly refined. It is the form of the requirements which is the cause for the difficulties, here.

Though model-based techniques are not adopted universally in the safety-critical domain, there are numerous examples of their application. It makes a big difference, however,

to introduce model-based techniques within one organisation, or to install them at an organisational boundary. A rather widespread mode of how to develop within a model-based way is to iteratively and incrementally refine some abstract design model. The end result of this continuous refinement is the implementation. From the very concept of that idea, high-level states are *refined* into lower-level ones. This does not need to be a strict refinement in the mathematical sense above. But essentially, the development produces a chain of refinement maps from the abstract to the concrete layer.

Comparing this with a situation where some manufacturer has a line of products verified (in the practical meaning of that word) against standard, text-based specifications, and the next version shall be proven consistent with a statechart specification. Usually, there will not be any "simple" refinement map, as this would only be available if the product architecture matches the one of the model. More often than not, for any system of notrivial complexity, the implementation mechanics will differ considerably from that of the specification model, and the existence of a useable refinement map is highly improbable. The example of the RBC model is no exception to that general observation.

6.3 Refinement in Practice: Verification by Testing

Testing is the classical apporach to verify properties of a system. This could be done either in the field or in lab. The different aspects are usually addressed in specifically targeted tests:

Safety: Tests focussed on extreme or erroneous behaviour

Conformity: Tests focussed on regular and typical functional behavior

Serviceability: Regular operational cases as well as operationally relevant disturbed cases.

These categories, with which all actors in the field are well acquainted, apply to *complete* systems. They describe in which respect a system is validated once it is developed. A safety validation step is required for any safety-critical system by the standards. During development, white box tests for unit verification and gray-box test for integration checks are applied. In the common depiction of standard development processes as a V, these tests start at the bottom (unit tests) and are applied throughout the phases on the right.

Here, we use the terms "verification" and "validation" in their technical meaning as it is defined in the CENELEC standards: *Verification* serves to establish that the output of one development step is an implementation of its input requirement, i.e., the output of the previous step. For instance, as a first step in a development, one may define the system architecture and apportion the requirements to the system components. Then, it must be *verified* that the combination of the components according to the architecture will satisfy all input requirements. *Validation* checks the full system against user requirements. More

general, validation is also used to denote checking whether some design artifact conforms to a specification given several steps earlier. One may, for instance, validate the software of the safety kernel after integrating its component modules against safety properties.

In model-based development as described above, testing begins already on the left side of the V. One may perform verification as well as validation already in earlier phases. This becomes possible if intermediate development artifacts take the form of executable models.

In the strictly regulated development of safety-critical rail systems, for each step a thorough verification method has to be chosen to stand a chance of getting the result certified for field use. We would consider systematic testing as a promising approach.

Here, the specification is an executable SysML specification composed of a few block diagrams whose behavior in turn is given by statecharts, enriched by program code. Such a specificaiton may be regarded, semantically, as an extended finite state machines (E-FSMs).

A method for checking behavioral conformance[4] of some system with an FSM (propositional, not "extended") has been described by Chow in [Ch78]. Essentially, one checks that each transition in the FSM is matched by an action the implementation. Lifting this idea to extended FSMs, one would not only check each transition, but also take the set of variable valuations into account, for instance, by requiring modified condition/decision coverage (MC/DC). This is what advanced commercial test generation tools promise to automate to a large extent. Rhapsody Automatic Test Generation (ATG) and RT-Tester are just two examples of such tools.

A test suite which covers the requirement specification thoroughly can then be used to verify that some other executable design model, or a program, conforms behaviorally to the requirement SysML-model. Even if the architecture of the implementation differs considerably from that of the requirement—what would complicate a refinement proof, even an informal one—, a successful test would provide useful evidence for arguing the correctness of the implementation.

7 Outlook and Conclusion

7.1 Future Use of the Model

The basic functionality of an ETCS RBC has been realized as a SysML model together with complementing program code. Applying code generation and compilation to it, one obtains an executable program which can communicate with ILS and trains. This program has been integrated into the rail operation simulation of the RailSiTe laboratory of DLR, so that the RBC can be utilized in the simulation of a demonstration scenario.

[4] In this case, "behavioral conformance" is to be interpreted as conformance with the full behavior spectrum of the requirement model—for implementing a model, *all* of the model's behavior is to be considered "regular".

In order to test whether an implemented RBC exhibits a behavior equal to that of the model, test cases are to be derived from the model systematically. This can be automated, at least in parts, with the help of suitable tools. Corresponding trials using parts of the model have already been conducted successfully and are to be continued in the future. The automated generation and optimization of instances and configurations is also in the focus of our ongoing work.

The use as a behavioral reference for a "real" RBC is always possible to the extent the functionality under test has been modeled already. A complete RBC model would require to model also the interface between two RBC. Besides that several practical questions such as the handling of manufacturer and country specific RBC variants have to be solved before the successful use for the test of "real" RBC.

7.2 Discussion and Conclusion

To construct a good model is a challenging task. In addition to familiarity with the system which is to be modeled, the language(s) and tools involved, there need to be taken several design decisions. Based on the experiences from modeling of a railway signalling system, several aspects of creating a generic and flexible model have been discussed in the paper at hand. In particular, alternatives and decision criteria for modeling of variants have been presented.

Undoubtedly the model-based approach has many advantages: the diagrams are much clearer and thus easier to maintain than program code; the automatic code generation reduces the effort and error potential of programming. The precondition here is, of course, that sufficiently many parts of the system can be captured in the form of diagrams. However, in modeling practice one hits upon a limited supply of applicable model elements—the more complex the data types in use (e.g. message between the RBC and neighboring systems) and the more dynamic the system to be modeled (e.g. configurability of the RBC), the more complementing program code needs to be written and maintained in addition to the model. We have analyzed which parts of the RBC behavior could be modeled by state charts and which parts needed to be coded by hand. We mainly identified missing code generation abilities for several diagram types as a practical hindrance to achieve a higher degree of formalization in our model.

Regarding the validation of the model and checking the conformance of an implementation to the model, systematic testing is a plausible approach. In the case of generic systems like the RBC, however, it is not obvious how to instantiate this in practice. This is usually not addressed in conceptual accounts of model-based development processes. for a systematic approach, it might be a good idea to capture the railway infrastructure data in a suitable modeling language as well and to generate code from it for the configuration of the RBC.

To summarize, the developed model of the RBC can be expected to form a good basis for extensive, automatic and high-quality future tests of ETCS radio block centers and for demonstration of compliance with well-defined test criteria. And it is a well-suited case study for extending systematically to pure, model-based development approaches.

References

[07] TR SIG ZA – Zulassung und Abnahme von Signal- und Zugsicherungsanlagen gemäß BOStrab, tech. rep., issuing date: 12/2008, Verband Deutscher Verkehrsunternehmen, May 2007.

[13] Testspezifikation SCI-RBC. Version 01, tech. rep., issuing date: 21/02/2013, Thales TS GmbH, 2013.

[14a] Lastenheft BTSF3 - Betrieblich-technische Systemfunktionen für ETCS SRS Baseline 3. Version 1.4, tech. rep., DB Netz AG, 2014.

[14b] Benefits of standardising interlocking interfaces, initiative "euLyNX", published on 06/02/2014, accessed on 09/06/2016, 2014, URL: http://eulynx.eu.

[14c] Protokolldefinition - RaSTA (früher: SAHARA), tech. rep., issuing date: 04/06/2014, NeuPro-Konsortium, 2014.

[14d] SCI-RBC – FAS TAS TAV Schnittstelle ESTW-RBC V2. Baseline 0.19, tech. rep., issuing date: 06/06/2014, DB Netze AG, 2014.

[14e] SUBSET-026 – System Requirements Specification. Version 3.4.0, tech. rep., issuing date: 12/05/2014, ERTMS, 2014, URL: http://www.era.europa.eu/Core-Activities/ERTMS/Pages/Set-of-specifications-2.aspx.

[14f] SUBSET-039 - FIS for the RBC/RBC Handover. Version 3.1.0, tech. rep., issuing date: 09/05/2014, ERTMS, 2014, URL: http://www.era.europa.eu/Core-Activities/ERTMS/Pages/Set-of-specifications-2.aspx.

[15] SUBSET-037 – EuroRadioFIS. Version 3.2.0, tech. rep., issuing date: 17/12/2015, ERTMS, 2015, URL: http://www.era.europa.eu/Core-Activities/ERTMS/Pages/Set-of-specifications-3.aspx.

[Br97] Broy, M.: Compositional refinement of interactive systems. J. ACM 44/6, pp. 850–891, 1997.

[Ch78] Chow, T. S.: Testing software design modeled by finite-state machines. IEEE Trans. on Software Eng. 4/, pp. 178–187, 1978.

[La91] Lamport, L.: The temporal logic of actions. ACM Transactions on Programming Languages and Systems 16/, pp. 872–923, 1991.

[LKL02] Lee, K.; Kang, K. C.; Lee, J.: Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In (Gacek, C., ed.): Software Reuse: Methods, Techniques, and Tools: 7th International Conference, ICSR-7 Austin, TX, USA, April 15–19, 2002 Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 62–77, 2002, ISBN: 978-3-540-46020-6, URL: http://dx.doi.org/10.1007/3-540-46020-9_5.

[RF14] Reinhartz-Berger, I.; Figl, K.: Comprehensibility of Orthogonal Variability Modeling Languages: The Cases of CVL and OVM. In: Proceedings of the 18th International Software Product Line Conference - Volume 1. SPLC '14, ACM, Florence, Italy, pp. 42–51, 2014, ISBN: 978-1-4503-2740-4, URL: http://doi.acm.org/10.1145/2648511.2648516.

[SH16] Schwencke, D.; Hungar, H.: Development of Reference Models of Signaling Components: the Example of the Radio Block Center. Signal+Draht 108/9, pp. 24–32, 2016.

[Su11] Suwe, K.-H.: Proposal for a new type approval in signalling and telecommunications. Signal+Draht 103/7-8, pp. 6–19, 2011.