Modellbasierte Testfallgenerierung am Beispiel einer RBC-Funktion

Model-based test case generation using the example of an RBC function

Daniel Schwencke

Während die Ausführung funktionaler Tests von Komponenten oder ganzen Systemen inzwischen oft automatisiert abläuft, erfolgt das Testdesign in der Regel manuell durch den Tester. Trotz Unterstützung durch Testmanagementwerkzeuge ist dies aufwendig und fehleranfällig, besonders bei Änderungen an der Testsuite. Die automatische Generierung von Testfällen aus Systemmodellen verspricht hier Besserung. Im vorliegenden Beitrag wird ein solcher Ansatz vorgestellt. Es wird von der Anwendung auf eine ausgewählte Funktion eines Radio Block Centers (RBC) und den dabei gemachten Erfahrungen berichtet.

1 Einleitung

1.1 Testprozesse

Das Testen bei der Entwicklung von Systemen der Leit- und Sicherungstechnik (LST) ist essentieller Bestandteil der Qualitätssicherung. Die einschlägigen Prozessnormen [1, 2] verpflichten den Anwender zur Verifikation und Validierung des entwickelten Systems und seiner Bestandteile, die in den allermeisten Fällen durch Testen realisiert wird. Die Erfahrungen nach der Inbetriebnahme von mehr bzw. weniger ausführlich getesteten Systemen bestätigen in der Regel die Bedeutung des Testens.

Testprozesse kann man grob in vier Phasen einteilen, wie in Bild 1 dargestellt. Die Hauptaufwände liegen dabei in den mittleren Phasen. Während die Testausführung je nach Schnittstellen des getesteten Systems (system under test = SUT) und Formalisierbarkeit der Tests heute oft automatisiert mithilfe von Testskripten abläuft, geschieht das Design der Testfälle für Komponenten- oder Systemtests in der Regel manuell. Der Tester erstellt eine Menge von Testfällen, die mindestens jede Systemanforderung einmal überprüfen, und hält die Testsuite bei Änderungen aktuell. Mehr als die Hälfte des gesamten Testaufwands fällt in dieser Phase an; und der gesamte Testaufwand wiederum kann bis zur Hälfte der Aufwände eines Entwicklungsprojekts betragen.

Insgesamt besteht also beim Testdesign großes Potenzial für Automatisierungsansätze, wenngleich eine vollständige Automatisierung wegen der textuell vorliegenden Systemanforderungen schwer vorstellbar ist. Auch die Qualität der Testsuite kann durch-

Bild 1: Gliederung des Testprozesses und durchschnittliche Aufwandsverteilung
Fig. 1: The structure of the test process and the average cost distribution

While the execution of the functional tests for components or entire systems is nowadays mostly automated, the test design is usually performed manually by the tester. Despite some support from test management tools, this is a time-consuming and error-prone task, especially if changes need to be made to the test suite. The automatic generation of test cases from system models can help to improve the situation. This article presents one such approach. It specifically reports on the application of this approach to a selected function of a radio block centre (RBC), as well as the experiences gained.

1 Introduction

1.1 Test processes

Testing during the development of signalling systems is an integral part of quality assurance. The verification and validation of the systems and their constituents is required by the relevant process standards [1, 2] and this is realised by testing in the vast majority of the cases. Experience from the commissioning of more or less comprehensively tested systems usually confirms the importance of testing.

Test processes can be roughly divided into four phases as shown in fig. 1. The main costs pertain to the central phases. While the test execution is now often automated using test scripts depending on the interfaces in the system under test (SUT) and the amenability of the tests to formalisation, the design of the test cases for component or system testing is usually performed manually. The tester creates a set of test cases, which checks each system requirement at least once, and keeps the test suite up to date when any changes occur. More than half the overall test costs pertain to that phase and in turn the overall test costs can amount up to half of the costs for a development project.

As such, the test design phase has great potential for automation approaches, even though complete automation is difficult to imagine due to the system requirements which exist in textual form. In addition, the test suite quality can profit from a (partially) automated and thus multi-step test design, i. e. any problems in the required specifications can be revealed with a higher probability than during the direct manual transformation of requirements into test cases. Moreover, the consistency of the test suite can be improved by such an approach or it can be preserved during any changes.

1.2 Model-based testing

Model-based testing constitutes just such an automation approach. It means the (automatic) generation of test cases from

aus von einem (teil)automatisierten und damit mehrschrittigen Testdesign profitieren. Zum Beispiel können Probleme in der Anforderungsspezifikation so eher entdeckt werden als bei einer direkten manuellen Umsetzung von Anforderungen in Testfälle. Auch die Konsistenz der Testsuite kann durch einen solchen Ansatz verbessert bzw. bei Änderungen erhalten werden.

1.2 Modellbasiertes Testen

Modellbasiertes Testen stellt einen solchen Ansatz dar. Es bezeichnet die (automatische) Generierung von Testfällen aus (grafischen) Modellen, die auf Basis der Systemanforderungen vom Tester (mithilfe eines Modellierungswerkzeuges) erstellt werden. Kern des Ansatzes ist ein entsprechendes Werkzeug zur Testfallgenerierung. Bei der Entscheidung für ein Werkzeug sind verschiedene Aspekte wie die Leistungsfähigkeit, die Konfigurierbarkeit und die Integration in den Testprozess von entscheidender Bedeutung, um die gewünschten Automatisierungs- und Testziele zu erreichen. Es gibt deutliche Unterschiede, was unter „modellbasiertem Testen" verstanden wird. Grundsätzlich unterscheidet man

- grafische Testfallnotation,
- umgebungsmodellbasierten Ansatz und
- systemmodellbasierten Ansatz.

Während im ersten Fall lediglich Testskripte aus einer grafischen Darstellung von Testfällen erzeugt werden, werden im zweiten Fall tatsächlich Testfälle mit entsprechenden Eingabedaten aus dem Umgebungsmodell abgeleitet, die zuvor nicht explizit modelliert wurden. Das größte Automatisierungspotenzial besitzt der dritte Fall, in dem zusätzlich das erwartete Ergebnis eines Testfalls generiert wird und die Abdeckung von Systemfunktionalitäten durch die Testfälle nachvollziehbar wird. Allerdings stößt man in diesem Fall auch auf die größte Generierungskomplexität; werden die Modelle zu groß, kann es zu inakzeptabel langen Berechnungszeiten durch die Werkzeuge kommen, die meist mit Techniken wie symbolischer Ausführung und Theoremsolvern arbeiten.
Vorteile modellbasierten Testens wurden bereits in [3] andiskutiert und liegen

- in der Zeit- und Kostenersparnis durch Automatisierung,
- im Frontloading (Testen von Teilprodukten durch Teilmodelle, frühes Fehlerfinden),
- in einer besseren Wartbarkeit der Testsuite,
- in der strukturellen Abdeckung des Modells (zusätzlich zur Anforderungsabdeckung) und
- in der Nutzung eines (grafischen) Modells mit seiner Präzision, Kompaktheit und Simulierbarkeit.

Der im vorliegenden Beitrag vorgestellte Ansatz ist bewusst systemmodellbasiert. Er mag damit zwar zunächst relativ anspruchsvoll sein und in der Modellerstellung eher Entwicklerfähigkeiten als klassische Kompetenzen eines Testers benötigen; doch die Motivation dafür ist eine möglichst starke Automatisierung, die letztendlich sowohl den Nutzen als auch die Anwendbarkeit maximiert. Als Modellierungsumgebung und Testfallgenerator wurde das UML / SysML-basierte Werkzeug Rational Rhapsody [4] von IBM gewählt (UML = Unified Modeling Language, SysML = Systems Modeling Language). Es ist prinzipiell in allen Entwicklungsphasen eines Systems anwendbar (vom Unit- bis zum Systemtest), wobei der Nutzen für spätere Phasen als größer eingeschätzt wird.

1.3 Anwendung im LST-Kontext

Allgemein steigt die Komplexität moderner LST-Systeme, speziell die Komplexität der eingesetzten Software. Dies stellt immer höhere Anforderungen an ein gutes Testdesign. Modellbasiertes Testen kann hier helfen, den Überblick über die Testsuite zu behalten, zu ei-

(graphic) models, which are created by the tester (using a modelling tool) based on the system requirements. The appropriate tool for the test case generation lies at the heart of the approach. When choosing a tool, the consideration of several aspects such as its performance, configurability and integration into the test process is of great importance for the achievement of the desired automation and test goals. Significant differences exist among model-based testing approaches. In essence, it is possible to differentiate between

- graphic test case notation,
- an environmental model-based approach, and
- a system-model based approach.

Whereas only test scripts are generated from the graphic representation of the test cases in the first approach, test cases which have not been explicitly modelled before are derived from the environment model, including their input data, in the second approach. The third approach has the biggest automation potential, in that the expected test case results are also generated and the coverage of system functionality by the test cases thus becomes traceable. However, this approach also involves the biggest generation complexity. If the models become too large, this can lead to unacceptably long calculation times using tools which mostly rely on techniques such as symbolic execution and theorem solving.
The advantages of model-based testing have already been touched on in [3] and include

- time and cost savings through automation,
- frontloading (the testing of sub-products using sub-models, early error discovery),
- the improved maintainability of the test suite,
- the structural coverage of the model (in addition to coverage of the requirements), and
- the utilisation of a (graphic) model with its precision, compactness and the option of simulating it.

The approach presented in the article at hand has been intentionally chosen to be system-model-based. As such, it may be relatively demanding and require developer skills rather than classic tester competencies during model creation; however, our motivation for that choice was to achieve the strongest possible automation, which ultimately maximises the benefit as well as the applicability. We chose the UML / SysML-based Rational Rhapsody tool [4] by IBM as the modelling environment and test case generator (UML = Unified Modelling Language, SysML = Systems Modelling Language). In principle, it can be applied in all the system development phases (from the unit to the system test), but we would expect a greater benefit to be achieved from its use in the later development phases.

1.3 Application within the context of signalling systems

In general, the complexity of modern signalling systems is increasing, especially with regard to the complexity of the used software. This poses increasing challenges to good test design. Model-based testing can help to maintain an overview of the test suite, to arrive at a structured choice of tests and to create and manage a larger set of test cases efficiently. On the other hand, the model complexity is limited by the performance of the generation tool (cf. section 1.2). Here, initiatives such as EULYNX [5] and Reference CCS Architecture (RCA) [6] have done model-based testing a favour by modularising the overall signalling system. Additionally, the standard interfaces defined by them have paved the way towards standard test suites, whose high quality demands are another argument for the use of automated test de-

ner strukturierteren Auswahl der Tests zu gelangen und eine größere Menge von Testfällen effizient zu erstellen und zu verwalten. Andererseits sind (vgl. Abschnitt 1.2) der Modellkomplexität durch die Generatorleistung Grenzen gesetzt – hier kommen Initiativen wie EULYNX [5] und Reference CCS Architecture (RCA) [6] durch die Gesamtsystemmodularisierung dem modellbasierten Testen entgegen. Die dort definierten Standardschnittstellen ebnen zudem den Weg zu Standard-Testsuiten, deren hohe Qualitätsanforderungen ebenfalls für einen Einsatz automatisierten Testdesigns sprechen. Ein wichtiger Grund, der den Nutzen modellbasierten Testens deutlich steigert, sind häufiger werdende Softwareupdates und die dafür durchzuführenden zusätzlichen Regressionstests.
Im vorliegenden Beitrag wird über die Testfallgenerierung für eine realistische Funktion eines (deutschen) RBC berichtet: die Unterstützung der vom Stellwerk durchgeführten Durchrutschwegauflösung. Weitere RBC-Funktionalitäten werden dabei nur so weit betrachtet, wie dies zur Modellierung der gewählten Funktion nötig ist. Ziel war es, die prinzipielle Anwendbarkeit des gewählten modellbasierten Ansatzes an einem kompakten, ansonsten jedoch realitätsnahen Beispiel nachzuweisen.

1.4 Aufbau des Beitrags

Im folgenden Kapitel 2 wird der gewählte Ansatz für das modellbasierte Testen vorgestellt, inklusive der genutzten Werkzeuge und dem verwendeten Prozess. Kapitel 3 enthält nach einer Einführung in das Anwendungsbeispiel den Bericht über die Anwendung des vorgestellten Ansatzes darauf. In Kapitel 4 werden die dabei gemachten Erfahrungen reflektiert und die Anwendbarkeit auf weitere Systeme wird andiskutiert. Fazit und Ausblick finden sich schließlich in Kapitel 5.

2 Ansatz

2.1 Rhapsody und Test-Add-ons

Für die vorliegenden Arbeiten wurde das „Automatic Test Generation“ (ATG) Add-on [7] des UML / SysML-Werkzeuges Rational Rhapsody von IBM für die Testfallgenerierung verwendet. Dieses basiert wiederum auf dem „TestConductor“ Add-on [7]. Insgesamt bilden Rhapsody und die beiden Add-ons eine eng integrierte Umgebung, die den gesamten Prozess des modellbasierten Testens von der Modellerstellung (Rational Rhapsody) über die Testfallgenerierung (ATG Add-on) bis hin zur Testausführung (TestConductor Add-on) abdecken. Es handelt sich um proprietäre, kostenpflichtige Software.
Der ATG-Testfallgenerator arbeitet auf durch Rhapsody generiertem C++-Code des Modells. Dieser muss ausführbar sein, da ATG auf dessen Basis eine virtuelle Modellausführung im Rahmen der Testfallgenerierung vornimmt. Je nach Wahl versucht ATG, eine strukturelle Abdeckung des Modells (Betreten von Zuständen, Durchführen von Zustandsübergängen und Ausführen von Operationen) und / oder die Abdeckung des generierten Codes (modified condition / decision coverage = MC/DC) zu erreichen. Die dazu genutzte Eingabeschnittstelle des SUT kann flexibel auf Basis des Modells gewählt werden. Die generierten Testfälle bestehen aus einer Reihung von eingehenden (Stimuli) und ausgehenden (Reaktionen) Operationen des SUT samt ihren Parametern (Testdaten bzw. Systemausgaben). Sie können in Rhapsody als Sequenzdiagramme grafisch dargestellt werden. Die manuelle Erstellung weiterer solcher Testfälle ist möglich.

2.2 Modellierungs- und Generierungsprozess

Die im Rahmen der Testfallgenerierung angewandten Schritte und die dazu benötigten Werkzeuge sind in Bild 2 dargestellt. Er geht da-

signs. An important factor which clearly increases the benefit of model-based testing is the rising number of software updates and the necessary regression testing connected with them.
This article reports on the generation of tests for a real function of a (German) RBC, namely support for the overlap release which is performed by the interlocking. In doing so, further RBC functionality is regarded only to the extent necessary for modelling the selected function. The aim was to prove the feasibility of the chosen model-based approach using a compact example which is otherwise very close to reality.

1.4 The structure of the article

The following chapter 2 provides an introduction to the chosen approach for model-based testing, including the used tools and process. After presenting an example of the application, chapter 3 reports on the application of the introduced approach in relation to the example. Chapter 4 reflects on the experiences gained during the exercise and on its applicability to further systems. Finally, the conclusion and outlook can be found in chapter 5.

2 The approach

2.1 Rhapsody and test add-ons

The “Automatic Test Generation” (ATG) add-on [7] for the Rational Rhapsody UML/SysML tool by IBM was used for the test case generation presented here. This in turn is based on the “TestConductor” add-on [7]. Altogether, Rhapsody and the two add-ons form a tightly integrated environment, which covers the whole model-based testing process starting with the model creation (Rational Rhapsody) and moving on to the test case generation (ATG add-on) and the test execution (TestConductor add-on). All of this is proprietary commercial software.
The ATG test case generator works with C++ code for the model which has been generated by Rhapsody. That code must be executable, because ATG performs a virtual model execution based on it during the course of the test case generation. ATG tries to achieve the structural coverage of the model (entering each state, taking each state transition and implementing each operation) and / or the coverage of the generated code (modified condition / decision coverage = MC/DC) depending on the user’s selection. The input interface of the SUT used for that purpose can be flexibly adapted (based on the model). The generated test cases consist of a sequence of incoming (stimuli) and outgoing (reactions) operations of the SUT, including their parameters (test data or system outputs, respectively). They can be displayed graphically as sequence diagrams by Rhapsody. The manual creation of further such test cases is also possible.

2.2 The modelling and generation process

The steps belonging to the test case generation process and the tools supporting them are depicted in fig. 2. It is assumed that textual SUT requirements exist as the input for the process. These first need to be comprehended, including their interplay, and they need to be reduced to requirements which are relevant for the SUT (the “System Spec Analysis” step in the figure), if applicable. In the second step (“TCG Model Creation”), the SysML model, including state charts capturing the system behaviour, is created. It is important to note that the model is indeed a system model, but one intended for the purpose of test case generation and not for implementation: it is legitimate to abstract the system’s behaviour, i. e. by means of aggregation or omission (in order to reduce the number and length of the generated test

von aus, dass Anforderungen an das SUT in textueller Form vorliegen. Diese müssen zunächst in ihrem Zusammenwirken verstanden und ggf. auf für das SUT relevante Anforderungen reduziert werden („System Spec Analysis“ in der Abbildung). Im zweiten Schritt („TCG Model Creation“) wird das SysML-Modell inklusive der Verhaltensbeschreibung in Form von Zustandsdiagrammen erstellt. Wichtig ist, dass das Modell zwar ein Systemmodell ist, jedoch zur Testfallgenerierung und nicht als Implementierung verwendet wird: Es ist legitim, das Verhalten des Systems z. B. durch Aggregation oder Auslassungen zu abstrahieren (zur Reduktion der Menge und Größe der generierten Testfälle). Andersherum ist es erlaubt, in einer Implementierung implizit vorhandene Verhaltensvarianten explizit zu modellieren (um entsprechende generierte Testfälle zu erzwingen). Um das Modell zu validieren, ist es sinnvoll, es während der Erstellung immer wieder mit verschiedenen Eingabesequenzen zu simulieren („TCG Model Simulation“). Dazu wird entsprechender Simulationscode generiert und interaktiv ausgeführt. So können Probleme und Fehler in Spezifikation und Modell früh erkannt und behoben werden. Die für die Testfallgenerierung nötige Generierung der Testarchitektur („TCG Architecture Generation“) erfolgt weitgehend automatisiert; der Nutzer legt dabei jedoch fest, welcher Teil des Modells das SUT darstellt. Nun folgt die eigentliche Generierung der Testfälle („Test Case Generation“). Dabei wird neben der gewünschten Abdeckung der Teil der externen SUT-Schnittstelle gewählt, für den ATG Eingabedaten generieren soll, sowie der Teil, der in den Testfällen durch ATG aufgezeichnet werden soll. Bereits während der Generierung gibt ATG Rückmeldung zur aktuell erreichten Abdeckung. Zuletzt werden die generierten Testfälle nach Rhapsody exportiert und dort inspiziert („Test Case Inspection“); sowohl als einzelne Sequenzdiagramme als auch durch die Ausführung der gesamten Testsuite auf dem Modell, um die von ATG berechnete Abdeckung zu verifizieren. Ist das Ergebnis nicht wie gewünscht, kann eine erneute Generierung z. B. mit geänderter Eingabeschnittstelle für ATG erfolgen, oder sogar das Modell angepasst werden.

3 Anwendung auf die RBC-Funktion

3.1 Beschreibung der Funktion

Das RBC ist Teil des europäischen Zugsicherungssystems ETCS (Level 2/3). Es fungiert für einen bestimmten Streckenbereich als Funkschnittstelle des Stellwerks zu den Zügen. Wichtigste Aufgabe ist die Erteilung von Fahrerlaubnissen an den Zug gemäß der vom Stellwerk erhaltenen Fahrstraßeninformationen. Daneben können viele weitere nützliche Funktionalitäten mithilfe des RBC realisiert werden; da die Schnittstelle zum Stellwerk nicht im ETCS standardisiert ist, ist dies abhängig vom jeweiligen Infrastrukturbetreiber.

Eine solche durch DB Netz definierte Funktion [8] ist die Unterstützung der Durchrutschweg-Auflösung. Klassisch wird der hinter dem Zielsignal einer Fahrstraße reservierte Durchrutschweg (D-Weg) zeitgesteuert vom Stellwerk aufgelöst, nachdem der Zug in den letzten Blockabschnitt eingefahren ist. Das RBC bekommt jedoch im Gegensatz zum Stellwerk eine sofortige Information vom Zug, wenn dieser zum Stillstand gekommen ist; wenn der Zug durch ETCS überwacht wird, macht es Sinn, diesen viel präziseren Zeitpunkt für die D-Weg-Auflösung zu verwenden. Neben einer entsprechenden Information zur D-Weg-Auflösung an das Stellwerk ist es dabei wichtig, das Stellwerk auch über die Vorbedingung der ETCS-Überwachung des Zuges zu informieren; d. h., ob eine Fahrerlaubnis vorliegt, die am Zielsignal endet. Eine beispielhafte Interaktion zwischen Stellwerk, RBC und Zug ist in Bild 3 dargestellt.

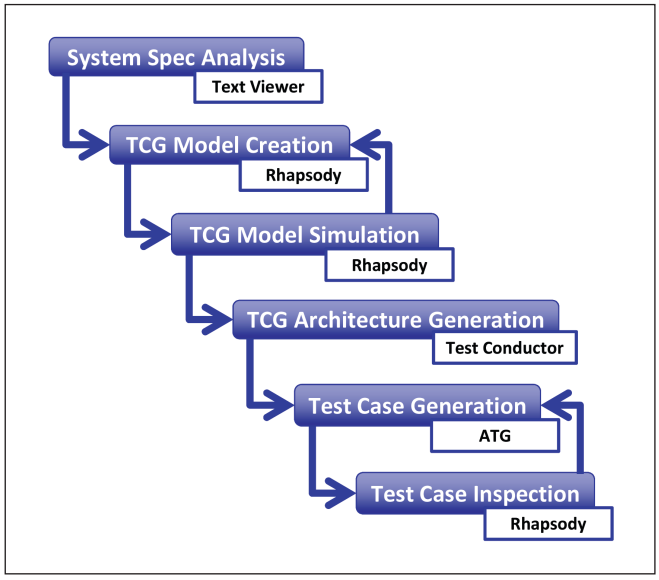

Bild 2: Prozessschritte und verwendete Werkzeuge im Testfallgenerierungsprozess
Fig. 2: The procedural steps and tools used as part of the test case generation process

cases). On the other hand, it is also legitimate to explicitly model behavioural variants which are implicit in an implementation (in order to force the generation of corresponding test cases). In order to validate the model, it makes sense to simulate the model using different input sequences (“TCG Model Simulation”) from time to time throughout its creation. Simulation-ready code is generated and interactively executed to this end. This means that problems and errors in the specifications and in the model can be detected and resolved early on. The generation of the test architecture (“TCG Architecture Generation”) is prerequisite to the test case generation and is mostly automated; the user only needs to select which part of the model represents the SUT. The actual generation of the test cases (“Test Case Generation”) then follows. In addition to the desired coverage, the part of the external SUT interface for which ATG should generate input data and the part which should be recorded in the test cases by ATG are also chosen. ATG provides feedback on the currently achieved coverage during the generation. Lastly, the generated test cases are exported to Rhapsody and can be inspected there (“Test Case Inspection”), either as single sequence diagrams or executing the complete test suite against the model in order to verify the coverage calculated by ATG. If the result is not as wanted, another generation can be run, i. e. with a changed input interface for ATG or even on an adapted model.

3 The application to the RBC function

3.1 A description of the function

The RBC is part of the European Train Control System ETCS (Level 2/3). It functions as the radio interface between the interlocking and the trains within a defined track area. Its foremost task is the granting of movement authorisations to the trains according to the route information received from the interlocking. In addition, many further useful functions can also be realised with the help of the RBC. This depends on the given infrastructure operator, because the interface to the interlocking has not been standardised in ETCS.

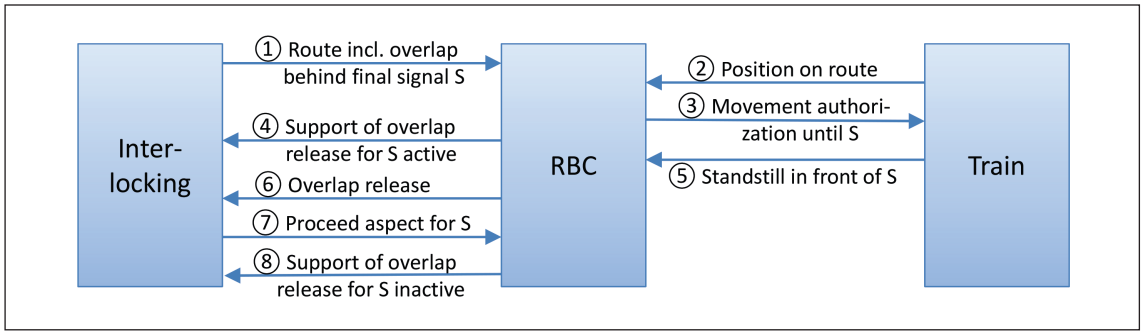

Bild 3: Beispielhafte Interaktion zwischen Stellwerk, RBC und Zug bei der durch das RBC unterstützten D-Weg-Auflösung
Fig. 3: An example interaction between the interlocking, the RBC and the train during an overlap release supported by the RBC

3.2 Modellierung

Die Erstellung des Modells ist der wichtigste und aufwendigste Schritt im in Abschnitt 2.2 vorgestellten Vorgehen. Es ist ratsam, sich zu Beginn darüber klar zu werden,

- was in welchem Umfang getestet werden soll,
- wie die externen Schnittstellen des SUT aussehen und
- welche Annahmen an die SUT-Umgebung gemacht werden.

Im vorliegenden Fall wurde entschieden, Tests für das RBC als Ganzes zu generieren, jedoch eine eingeschränkte und stark vereinfachte Version der Schnittstellen zu Stellwerk und Zug zu modellieren. Ein Grund für letzteres war, dass die Komplexität der Schnittstellen starken Einfluss auf den Schwierigkeitsgrad der Testfallgenerierung hat und hier noch keine Erfahrungen mit Rhapsody vorlagen, die für eine brauchbare Abschätzung genutzt werden konnten. Damit das eigentliche Ziel, der Test der Unterstützung der D-Weg-Auflösung (vgl. Abschnitt 3.1), erreicht werden konnte, war es nötig, auch das grundsätzliche RBC-Verhalten zu modellieren, das diese Funktion überhaupt erst ermöglicht; im Wesentlichen eine vereinfachte Fahrstraßen- und Zugverwaltung. Auch war es nötig, einen (auf ein einziges, nicht verzweigendes Gleis beschränkten) Mechanismus zur Konfiguration des RBC zu modellieren, da in der Realität RBC-verwaltete Signalelemente für die Unterstützung der D-Weg-Auflösung explizit konfiguriert werden müssen. Getroffene Annahmen an die Umgebung betrafen vor allem die korrekte Funktionsweise von Stellwerk und Zug; z. B. wurde in den entsprechenden Testkomponenten modelliert, dass für fahrtzeigende Signale kein reservierter D-Weg und eine Zugposition erst nach Anmeldung des Zuges am RBC an das RBC gemeldet werden kann. Aber auch eine Rekonfiguration des RBC während des Betriebs wurde ausgeschlossen.

Nach Festlegung der externen Schnittstellen wurde die innere Modellstruktur erstellt; Bild 4 zeigt beides zusammen in Form eines SysML Blockdefinitionsdiagramms. Die externen Schnittstellen werden durch Ports des RBC-Blocks repräsentiert: über den „ConfigPt" erhält das RBC seine Konfiguration; über den „IXLPt" erhält es Signaldaten (Signalbegriff, D-Weg-Reservierung) vom Stellwerk und sendet ihm die Nachrichten im Rahmen der Unterstützung der D-Weg-Auflösung; und über den „GSMRPt" erhält es Anmeldungs- und Positionsdaten (inkl. Stillstandsinformation) von Zügen und sendet ihnen Fahrerlaubnisse. All diese Nachrichten und Daten sind in den jeweiligen „XtoYInterfaces" definiert. Die innere Modellstruktur besteht aus zwei Blöcken „SignalElement" und „TrainElement", die Signale bzw. Züge repräsentieren. Die Fahrstraße eines Zuges ist über das Assoziationsende „itsRouteSignals" zu einer Liste von Signalen repräsentiert; und jedes Signal kennt über das Assoziationsende „itsRouteTrain" den Zug, zu dessen Fahrstraße es gehört.

Entscheidender für die Testfallgenerierung ist jedoch die Modellierung des RBC-Verhaltens. Jeder der drei Blöcke besitzt ein Zustandsdiagramm und Operationen, in denen sein Verhalten mo-

One such function is support for the overlap release, which has been defined by DB Netz [8]. Classically, an overlap which has been reserved behind the final signal on a route is released by the interlocking, if some time has elapsed since the train entered the last block section. However, in contrast to the interlocking, the RBC receives immediate information from the train when it reaches a standstill. As such, it makes sense to use this more precise point in time for the overlap release, if a train is being supervised by ETCS. In addition to the corresponding message for the overlap release sent to the interlocking, it is also important to inform the interlocking about any precondition of the ETCS train supervision, i. e. whether there is a movement authorisation which ends at the final signal. Fig. 3 depicts an exemplified interaction between an interlocking, the RBC and a train.

3.2 Modelling

The model creation is the most important and laborious step of the procedure presented in section 2.2. It is advisable to clarify at the very beginning:

- what is to be tested and to what extent,
- what the external interfaces of the SUT look like, and
- which assumptions about the SUT environment are being made.

In the given case, a decision was made to generate tests for the RBC as a whole, but to model a restricted and very simplified version of the interfaces with the interlocking and the train. One reason for the latter involved the fact that interface complexity has a strong impact on the degree of difficulty of the test case generation and there was no available experience with Rhapsody which could have been used to make a well-founded estimation. In order to reach the actual goal – to test the support for the overlap release (cf. section 3.1) – it was also necessary to model some basic RBC behaviour, which enables this function to work, mainly the simplified route and train management. Moreover it was necessary to model a mechanism (restricted to a single, non-branching track) for the configuration of the RBC, because in reality signal elements managed by the RBC must be explicitly configured to support the overlap release. The assumptions made about the environment mainly involved the correct functioning of the interlocking and the train, i. e. the corresponding test components were modelled so that no reserved overlap could be reported to the RBC for signals at the proceed aspect and so that a train position could only be reported after the train had registered with the RBC. In addition, any reconfiguration of the RBC during operations was excluded.

The inner model structure was created once the external interfaces had been determined. Fig. 4 shows both in a SysML block definition diagram. The external interfaces are represented by RBC block ports. The RBC receives its configuration via "ConfigPt". It receives the signal data (aspect, overlap reservation)

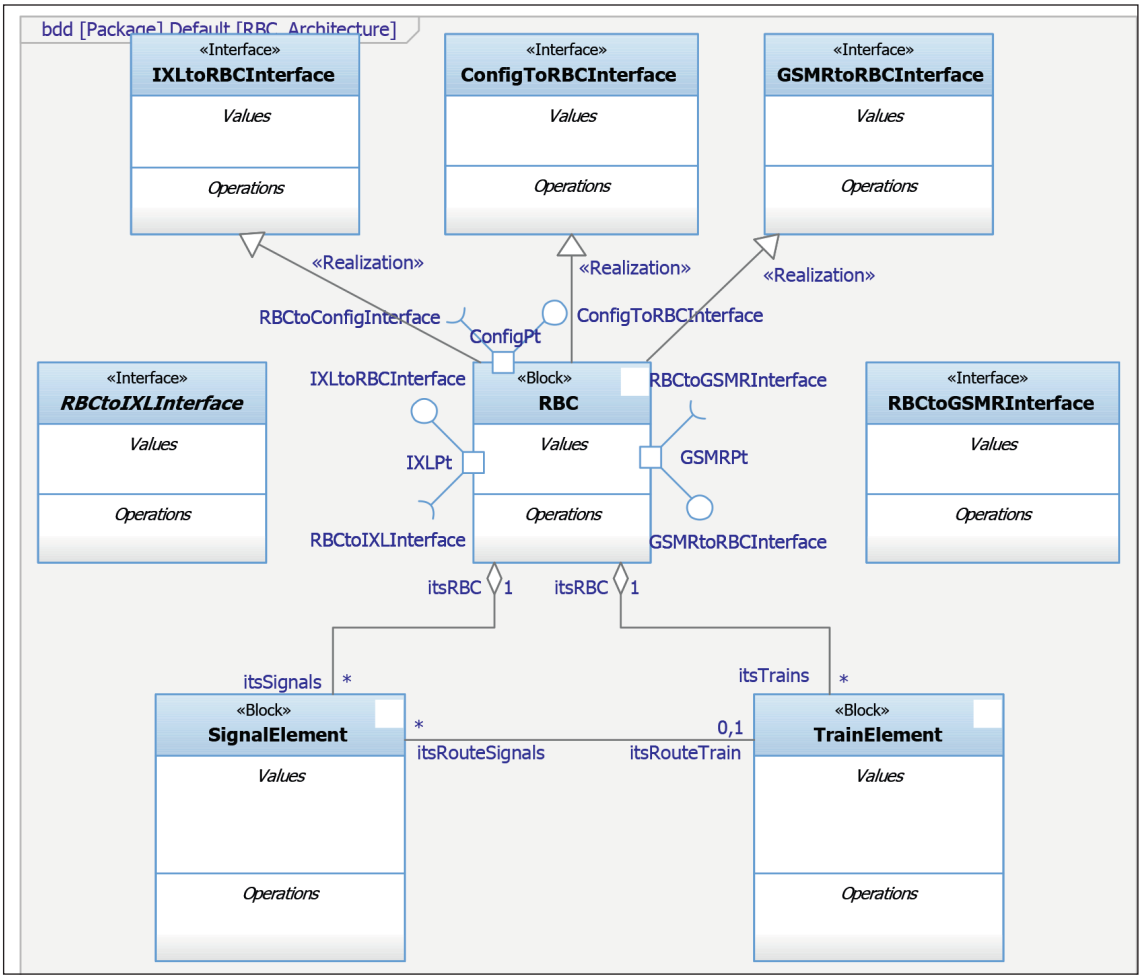

Bild 4: Struktur des RBC-Modells als SysML-Block-definitionsdiagramm
Fig. 4: The structure of the RBC model as a SysML block definition diagram

delliert bzw. als C++-Code hinterlegt ist. Der RBC-Block verwaltet die Signal- und Zugelemente, leitet einkommende Nachrichten an sie weiter und sendet ihre ausgehenden Nachrichten. Der TrainElement-Block verwaltet Fahrt / Stillstand des Zuges und seine Fahrstraße. Das Zustandsdiagramm des SignalElement-Blocks ist in Bild 5 dargestellt. Es verwaltet den aktuellen Signalbegriff, den D-Weg-Reservierungsstatus und realisiert die Funktionalität zur Unterstützung der D-Weg-Auflösung: nur wenn das Signal Haltbegriff zeigt, ein D-Weg reserviert ist und es für die Funktionalität konfiguriert ist, befindet es sich im Zustand „OverlapReservable“ (im Sinne einer Reservierung für die Auflösung durch ETCS), ansonsten im Zustand „OverlapNotReservable“. Wenn im ersteren Fall nun das Signal Ziel einer Fahrerlaubnis wird („toMADestination“), wechselt es vom Zustand „Not Reserved“ in den Zustand „Reserved“ und meldet dem Stellwerk, dass nun die D-Weg-Auflösung durch ETCS unterstützt wird. Hierfür sorgt im Falle eines Zugstillstands ein interner Übergang („toStandstill“) des Zustands „Reserved“. Sobald der D-Weg aufgelöst ist, das Signal Fahrtbegriff zeigt oder durch den Zug passiert wurde, wird dem Stellwerk das Ende der Unterstützung der D-Weg-Auflösung durch ETCS gemeldet und zurück in den Zustand „OverlapNotReservable“ gewechselt.

3.3 Testfallgenerierung

Zur Vorbereitung der Testfallgenerierung ist im Wesentlichen der durch ATG zu nutzende Teil der Eingabeschnittstellen zu definieren. Im vorliegenden Fall wurde sich auf einen Zug beschränkt, der entweder fahren oder stillstehen kann, sowie zunächst auf einen Streckenabschnitt mit drei Signalen, von denen zwei für die

from the interlocking and sends the interlocking messages related to the support for the overlap release via “IXLPt”. It receives registration and position data (including standstill information) from the trains and sends movement authorisations to them via “GSMRPt”. All of these messages and data are defined within the corresponding “XtoYInterfaces”. The inner model structure consists of the two “SignalElement” and “TrainElement” blocks which represent signals and trains respectively. The route of a train is represented by the “itsRouteSignals” association end which holds a list of signals; and each signal knows the train, to whose route it belongs, via the “itsRouteTrain” association end.
The crucial part for the test case generation involves the modelling of the RBC behaviour. Each of the three blocks has a state diagram and some operations, in which their behaviour is modelled or given as a C++ code respectively. The RBC block manages the signal and train elements, forwards incoming messages to them and sends their outgoing messages. The TrainElement block manages the train movement / standstill and its route. The state diagram of the SignalElement block is depicted in fig. 5. It manages the current signal aspect and the overlap reservation status and realises the functionality for the support of the overlap release. Only in case the signal shows stop, an overlap is reserved and the signal is configured for that functionality, it resides in the “OverlapReservable” state (in the sense of a reservation for the overlap release via ETCS). Otherwise, it is in the “OverlapNotReservable” state. In the first case, the status changes from “Not Reserved” to “Reserved”, if the signal becomes the destination of a movement authorisation (“toMADestination”) and the

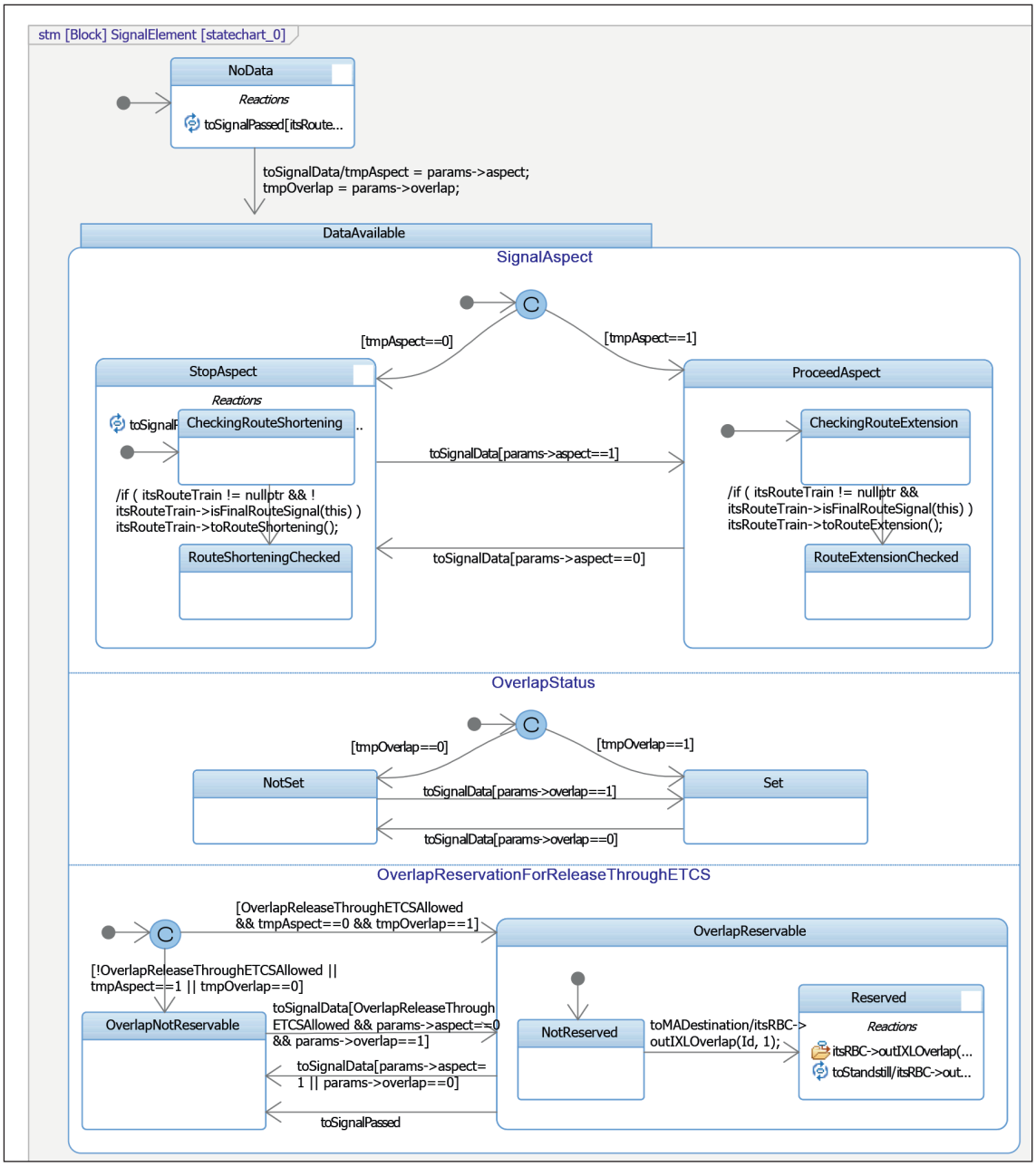

Bild 5: Verhalten eines RBC-Signalelements als SysML-Zustandsdiagramm
Fig. 5: The behaviour of an RBC signal element as a SysML state machine

D-Weg-Auflösung durch ETCS konfiguriert waren. In sieben Sekunden Generierungszeit wurde eine 100 %ige strukturelle Modellabdeckung der Blöcke „SignalElement“ und „TrainElement“ erreicht; es wurden dabei 20 Testfälle generiert. Die meisten davon sind sehr kurz und dienen der Abdeckung der verschiedenen Zustände und Zustandswechsel von Signal- und Zugelementen (Signalbegriffe, D-Weg-Reservierung, Fahrt / Stillstand). Ein generierter Testfall, der sich mit der Funktionalität der D-Weg-Auflösung durch ETCS befasst, ist als Sequenzdiagramm in Bild 6 gezeigt. Das SUT (mittlere, rote Lifeline) wird zunächst über den ConfigPt mit den drei Signalen konfiguriert; dann wird für eines der für die Funktionalität konfigurierten Signale über den IXLPt Haltbegriff und reservierter D-Weg gemeldet. In der Folge meldet sich ein Zug am RBC an und übermittelt, dass er vor dem genannten Signal steht. Das RBC meldet über den IXLPt die Unterstützung der D-Weg-Auflösung sowie nach einer Erteilung einer Fahrterlaubnis bis zum Signal über den GSMRPt, den Zugstillstand / die Zustimmung zur D-Weg-Auflösung.

interlocking is notified that the overlap release via ETCS is now being supported. The actual release is triggered by an internal transition (“toStandstill”) of the “Reserved” state once the train comes to a standstill. The interlocking is notified of the end of the overlap release support via ETCS as soon as the overlap is released, the signal shows proceed or it has been passed by the train and the signal then returns to the “OverlapNotReservable” state.

3.3 Test case generation

The preparation of the test case generation mainly requires the definition of the part of the input interface to be used by ATG. The present example is restricted to a single train which can either move or stand still, and initially to one track section with three signals, two of which have been configured to support overlap release via ETCS. 100 % structural model coverage of the “SignalElement” and “TrainElement” blocks was achieved within seven seconds of generation time. 20 test cases were generated during that time. The majority of the test cases are very short

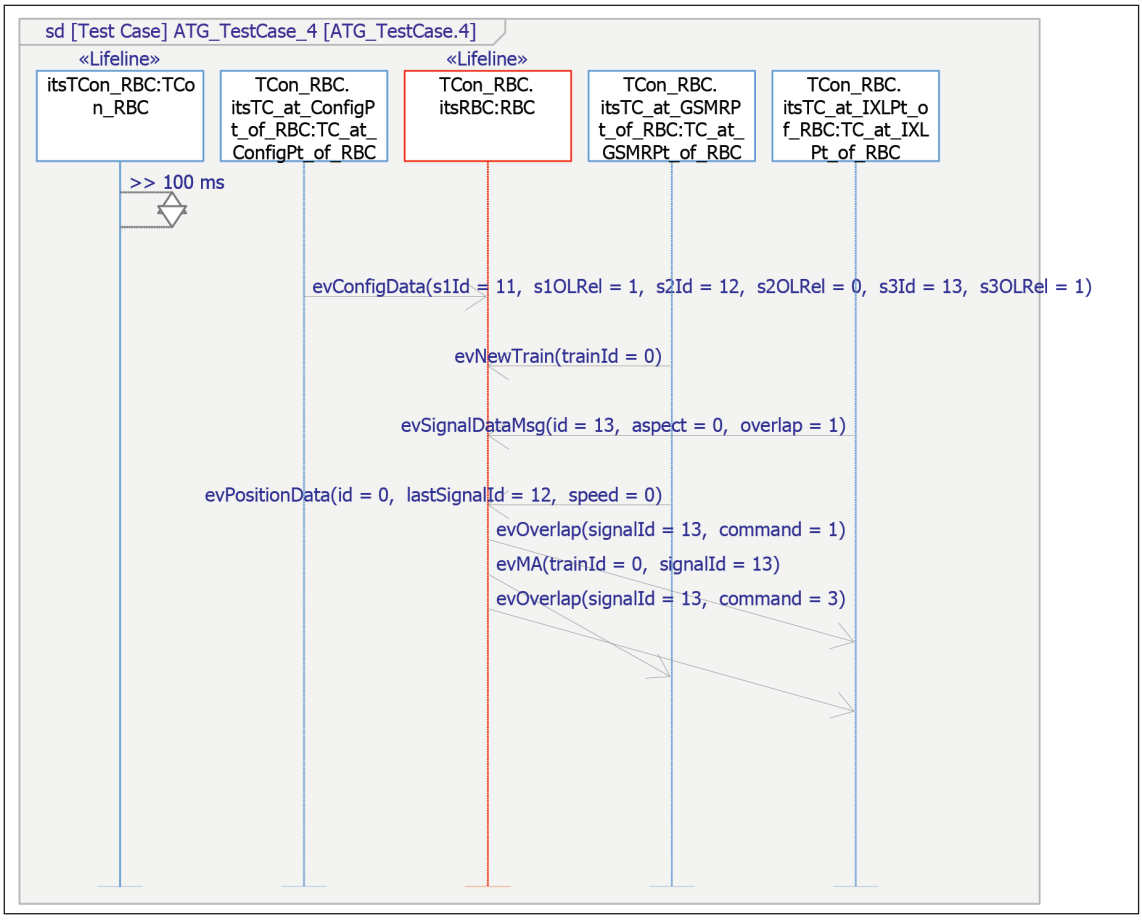

Bild 6: Durch ATG aus dem RBC-Modell generierter Testfall als SysML-Sequenzdiagramm
Fig. 6: The test case generated by ATG from the RBC model as a SysML sequence diagram

4 Erfahrungen und Anwendbarkeit für LST-Systeme

Es ist festzuhalten, dass die Abstraktion des SUT-Verhaltens für die Erstellung des Modells (vgl. Abschnitt 2.2) durchaus herausfordernd ist und dabei stark vom konkreten Anwendungsbeispiel abhängt. Allgemein kann die Modellierung dazu verleiten, das Systemverhalten zu detailliert, fast auf dem Level einer Implementierung, vorzunehmen. Zudem sind Abhängigkeiten in einem komplexen System schwer zu überblicken; aber genau dies wäre nötig, um zu wissen, welches Verhalten durch eine Abstraktion verloren geht. Das Denken in Funktionalitäten (wie in Abschnitt 3.1) statt in Systemarchitekturen kann hier helfen. Eng verwobene Konzepte eines Systems – in unserem Fall diejenigen der Fahrstraßenverwaltung und der Fahrerlaubniserteilung – können im Modell zu einem Konstrukt verschmolzen werden; und komplexe Schnittstellen sollten radikal auf die nötigen Informationen reduziert werden – in unserem Fall z. B. die Anmeldung des Zuges beim RBC durch die einfache Übermittlung einer Zug-ID statt eines Austausches mehrerer Nachrichten gemäß ETCS-Spezifikation [9].

Die im vorliegenden Fall generierten Testfälle lieferten in jedem Fall wertvolle Rückmeldungen: zum einen bzgl. der Systemmodellierung, z. B. wurden nicht erreichbare Modellelemente sowie ungünstig für die Generierung modellierte Modellteile identifiziert. Zum anderen bzgl. nicht bedachter Szenarien, z. B. zu Wiederholungen exakt gleicher Nachrichten, Szenarien, in denen keine Reaktion des Systems erfolgen soll oder einer unvollständig spezifizierten Kombinatorik von Funktionalitäten.

Bei der Testfallinspektion ergaben sich auch Hinweise auf mögliche Nachteile der Generierung: Es wurde offensichtlich, dass Modell und Generator keinen Begriff von einem „Normalverhalten“

and serve to cover the different states and status transitions of the signal and train elements (signal aspects, overlap reservation, movement/standstill). A generated test case which involves an overlap release via the ETCS functionality is shown as a sequence diagram in fig. 6. The SUT (the red lifeline in the centre) is initially configured via ConfigPt with data for the three signals. Afterwards, a stopping aspect and a reserved overlap for one of the signals which has been configured to support the functionality are received via the IXLPt. The train then registers with the RBC and transmits the information that it is standing in front of the signal under consideration. The RBC announces its support for the overlap release and the train standstill / approval of the overlap release via the IXLPt, once a movement authorisation has been sent to the signal via GSMRPt.

4 Experience and the applicability for signalling systems

It can be said that the abstraction of the SUT behaviour during model creation (cf. section 2.2) can indeed be challenging and it strongly depends on the specific application example. Generally, the modelling may tempt the user to create an overly detailed system behaviour, which comes close to an implementation. Furthermore, dependencies in a complex system are difficult to overview. However, this is precisely what is is necessary in order to be able to know which behaviour will be lost or affected by an abstraction. Thinking in functionalities (as in section 3.1) instead of system architectures can be helpful here. Tightly interrelated system concepts (in our case, route management and granting movement authorisations) can be made into a single construct in the model, while complex interfaces should

besitzen; Fixpunkt für den Generator ist die Erreichung der strukturellen Abdeckung des Modells. Auch wenn die Generierung von Standardszenarien prinzipiell im Modell erzwingbar ist, ist der aktuell wohl praktischere Ansatz, solche Szenarien als einige wenige händisch erstellte Testfälle hinzuzufügen. Ein anderer möglicher Nachteil generierter Testfälle ist unnötige Redundanz. Trotz einer gewissen Minimierung der Testsuite (ATG entfernt Testfälle, die Präfix eines anderen sind) blieben gelegentlich sehr ähnliche Testfälle in der Suite, die sich z. B. nur um ein (zufällig?) eingefügtes Zeitintervall unterschieden; Redundanz in problematischer Größenordnung konnte jedoch nicht festgestellt werden.
Das in Abschnitt 3 behandelte Anwendungsbeispiel ist, wenngleich der Realität entlehnt, bewusst kompakt gehalten, um zunächst die Machbarkeit des gewählten Testfallgenerierungsansatzes zu zeigen. Dies resultierte in absolut akzeptablen Größenordnungen von Generierungszeit, Testfällen und Testschritten. Es ist schwer, daraus Prognosen für komplexere Beispiele abzuleiten; unser Eindruck ist, dass weniger die reine Größe des Modells als vielmehr tief geschachteltes Verhalten sowie komplexe Schnittstellen sich vor allem in einer stark steigenden Generierungszeit niederschlagen könnten. Die generierten Testfälle scheinen dagegen handhabbar.
Die Testfallgenerierung für verschiedene Systemvarianten und -konfigurationen lässt sich technisch auf verschiedene Weisen im Modell organisieren; in unserem Fall war das generische 150 %-Modell durch eine mitmodellierte Konfigurationsschnittstelle konfigurierbar. Möchte man die bei diesem Ansatz potenziell auftretende Nicht-Erreichbarkeit von Modellelementen durch den Testfallgenerator bzgl. einer bestimmten Konfiguration vermeiden, ist die Nutzung von SysML Variation Points möglich.

5 (Zwischen)Bilanz und Ausblick

Einige Vor- und mögliche Nachteile der Testfallgenerierung wurden bereits in [3] diskutiert. Die in Abschnitt 3 beschriebene Anwendung des in Abschnitt 2.2 vorgestellten Modellierungs- und Testfallgenerierungsprozesses hat bestätigt, dass die Nutzung von (grafischer) Modellierung und Testfallinspektion sowie automatisierter Transformationen (C++ Compiler, Testarchitektur- und -fallgenerator) nicht nur die Effizienz erhöht, sondern auch den Nutzer früh auf Fehler oder Probleme hinweist. Die flexible Wahl des Anteils der SUT-Eingabeschnittstellen, die für den Testfallgenerator nutzbar sind, hat sich als mächtiges Mittel zur Steuerung der Generierungszeit erwiesen; z. B. für die Frage, ob ein Zug oder mehrere für die Testfallgenerierung berücksichtigt werden sollen.
Die Lessons Learned des RBC-Fallbeispiels beinhalten beispielsweise die Bedeutung einer frühen klaren Definition von Systemgrenze und Umgebungsmodell – ansonsten ist man später im Systemmodell für die Testfallgenerierung mit unnötigen Änderungen und zu viel modelliertem Verhalten konfrontiert. Weiterhin sollten die initialen Kosten, die anfängliche Lernkurve zur Nutzung der Werkzeuge und die Notwendigkeit eines durchgängigen Verständnisses und zielführenden Konzepts für einen in mehreren Schritten automatisierten Prozess nicht unterschätzt werden.
Für eine bessere Beurteilung der Anwendbarkeit des Ansatzes zum Testen verschiedener LST-Systeme sind nun weitere Schritte erforderlich. Aktuell wird im Rahmen des Shift2Rail-Projektes X2Rail-2 ein bis hin zur Testausführung generierter Tests erweitertes Vorgehen definiert und erprobt und zugleich mit einem Bahnübergangcontroller ein erweitertes Anwendungsbeispiel untersucht. Darüber hinaus wäre künftig gezielt anhand einer komplexen Funktionalität und / oder Systemkonfiguration zu erproben,

be rigorously reduced to the essential information (in our case, the registration of the train with the RBC by means of a simple transmission of the train ID, instead of the exchange of multiple messages according to the ETCS specifications [9]).
The test cases generated in the given example provided valuable feedback. On the one hand, this involved the system modelling, i. e. unreachable model elements and model parts which had not been optimally arranged for the generation could be identified. On the other hand, this involved non-anticipated scenarios, i. e. scenarios showing repetitions of the very same message, scenarios in which no system reaction should occur or scenarios uncovering the incompletely specified combinatorics of functionalities were able to be detected.
During the test case inspection, there were also indications of some possible disadvantages of the generation: it became obvious that the model and generator had no notion of "normal behaviour". The benchmark for the generator was the achievement of the model's structural coverage. Although it is possible in principle to force the generation of standard scenarios in the model, the more practical approach currently seems to involve the manual addition of a few such scenarios. Another possible disadvantage of the generated test cases involves unnecessary redundancy. Despite a certain degree of minimisation of the test suite (ATG removes test cases which are the prefix of one another) very similar test cases, which only differed, for example, in a (randomly?) inserted time interval, occasionally remained in the suite. However, no redundancy was ascertained at a problematic extent.
Although taken from the real world, the application example presented in section 3 was kept intentionally compact in order to show the feasibility of the chosen test case generation approach as a first step. This resulted in thoroughly acceptable dimensions for the generation time, the test cases and the test steps. It is difficult to derive prognoses for more complex examples from that. We are of the impression that it may be deeply nested behaviour and complex interfaces rather than the pure size of the model which could induce a quick rise in generation time. In contrast, the generated test cases seem to be manageable.
The generation of test cases for different system variants and configurations can be organised in the model in different ways. In our case, we created a generic 150 % model which could be configured using a configuration interface which is part of the model. This approach could lead to different configurations for certain model elements being unreachable for the test case generator; this can be avoided using the Variation Point construct provided by SysML.

5 (Intermediate) results and prospects

Some of the advantages and possible disadvantages of test case generation have already been discussed in [3]. The application (see section 3) of the modelling and test case generation process described in section 2.2 has confirmed that the use of (graphic) modelling and test case inspection, as well as automated transformations (the C++ compiler, test architecture and test case generators) not only increases efficiency, but also reveals errors or problems to the user early on. The flexible choice of the part of the SUT input interface which is to be used by the test case generator has proven to be a powerful tool for controlling the generation time, e. g. with regard to whether scenarios with one or several trains should be considered for test case generation.
The lessons learned from the RBC case study include, for example, the importance of an early and clear definition of the system border and the environment model. Otherwise, one may be con-

wo der Testfallgenerator an seine Grenzen stößt. Da dies immer auch von der Modellierung mit beeinflusst wird, wäre ein entsprechender Modellierungsleitfaden zu erarbeiten. Schließlich wäre die Verwendbarkeit zur Erreichung unterschiedlicher Testziele zu untersuchen; in einem ersten Schritt beispielsweise die Generierung von Testfällen für Fehlerszenarien. ■

fronted with unnecessary changes and superfluous modelled behaviour in the system model for test case generation later. Furthermore, the initial costs, the learning curve for using the tools and the necessity of a universal understanding and targeted concept for a process which is automated in multiple steps should not be underestimated.

Further steps need to be taken for a better assessment of the applicability of the presented approach for testing different signalling systems. An extended process, including the execution of the generated test, has currently been defined and applied in the X2Rail-2 Shift2Rail project. At the same time, a further application example involving a level crossing controller is being investigated in that project. In addition to this activity, a designated exploration of the limits of the test case generator should be undertaken using a complex functionality and / or system configuration. The creation of a modelling guide should be considered, since the modelling also influences the generator performance. Finally, the applicability for achieving different test goals should be investigated. For example, the generation of test cases for failure scenarios could be considered in the first step. ■

LITERATUR | **LITERATURE**

[1] DIN EN 50126-1:2018. Bahnanwendungen – Spezifikation und Nachweis von Zuverlässigkeit, Verfügbarkeit, Instandhaltbarkeit und Sicherheit (RAMS) - Teil 1: Generischer RAMS-Prozess; Deutsche Fassung EN 50126-1:2017
[2] DIN EN 50128:2012. Bahnanwendungen – Telekommunikationstechnik, Signaltechnik und Datenverarbeitungssysteme – Software für Eisenbahnsteuerungs- und Überwachungssysteme; Deutsche Fassung EN 50128:2011
[3] Caspar, C.; Schwencke, D.; Hungar, H.: Effizientes Testen modularisierter und standardisierter Stellwerkskomponenten, SIGNAL+DRAHT 9/2018
[4] IBM Rational Rhapsody Webseite, https://www.ibm.com/de-de/marketplace/uml-tools, zuletzt aufgerufen am 13.11.2019 um 15:46 Uhr
[5] EULYNX Webseite, https://eulynx.eu, zuletzt aufgerufen am 13.11.2019 um 15:54 Uhr
[6] Reference CCS Architecture Workgroup auf der ERTMS Users Group Webseite, https://ertms.be/workgroups/ccs_architecture, zuletzt aufgerufen am 13.11.2019 um 15:56 Uhr
[7] TestConductor und ATG Web-Dokumentation auf der IBM-Webseite, https://www.ibm.com/support/knowledgecenter/en/SSB2MU_8.4.0/com.btc.tcatg.user.doc/topics/com.btc.tcatg.user.doc.html, zuletzt aufgerufen am 13.11.2019 um 15:50 Uhr
[8] DB Netz AG: Lastenheft SCI-RBC. Version 0.20, 28.09.2015
[9] ERA, UNISIG, EEIG ERTMS Users Group: ERTMS/ETCS System Requirements Specification. Subset-026, Version 3.6.0, 13.05.2016

AUTOR | **AUTHOR**

Dr. Daniel Schwencke
Wissenschaftlicher Mitarbeiter Verifikations- und Validierungsmethoden / *Scientific staff Verification and Validation Methods*
Deutsches Zentrum für Luft- und Raumfahrt e.V. / *German Aerospace Center*
Institut für Verkehrssystemtechnik / *Institute of Transportation Systems*
Anschrift / *Address*: Lilienthalplatz 7, D-38108 Braunschweig
E-Mail: daniel.schwencke@dlr.de