Modellbasierter Systemtest eines Bahnübergangscontrollers

Model-based system testing of a level crossing controller

Daniel Schwencke

Modellbasiertes Testen (MBT) kann Qualität, Nachvollziehbarkeit und Automatisierungsgrad des Testdesigns durch die Generierung von Testfällen aus einem Modell verbessern. Für moderne modulare (System-of-Systems) Leit- und Sicherungstechnikarchitekturen erscheint eine Anwendung für Tests auf Systemebene besonders attraktiv. Der Beitrag präsentiert eine MBT-Fallstudie für einen Bahnübergangscontroller, die sowohl die Testdesign- als auch die Testdurchführungsphase umfasst. Die wichtigsten Schritte des angewandten MBT-Prozesses und die Ergebnisse der Fallstudie werden vorgestellt.

1 Einleitung

1.1 Verifikation von Leit- und Sicherungstechnik

Gemäß EULYNX-Konzept [1] und der darauf aufbauenden EULYNX-Systemarchitektur [2] verlagern sich die von einem Infrastrukturbetreiber (IB) bestellten und von einem Hersteller gelieferten Leit- und Sicherungstechnikeinheiten von kompletten Stellwerken hin zu Teilsystemen wie Feldelementcontrollern und Stellwerkskernlogik. Dies bedeutet, dass kleinere, aber auch mehr Systeme geliefert werden. Gleichzeitig gibt es die Tendenz, dass sich immer mehr Validierungs- und Verifikationsziele (V&V-Ziele) auf dieser „Lieferebene" konzentrieren, die alle auf die V&V gegen die spezifizierten Systemanforderungen hinauslaufen:

- Die klassische Notwendigkeit von Abnahmeprüfungen durch den IB bleibt bestehen.
- Die von den Herstellern im Rahmen der Entwicklung durchgeführte System-V&V nähert sich aufgrund präziserer Systemanforderungsspezifikationen teilweise der Verifikation anhand der vom IB aufgestellten Anforderungen an (vgl. den modellbasierten Systems-Engineering-Ansatz von EULYNX [3]).
- Die Zulassung wird sich weitgehend auf die Zertifizierung der gelieferten Systeme stützen müssen, wobei die Konformität mit einer standardisierten Spezifikation überprüft wird, von der bekannt ist, dass sie die Sicherheit und andere relevante Eigenschaften gewährleistet. Andernfalls würde der Zertifizierungsaufwand aufgrund der Kombinatorik der von verschiedenen Herstellern gelieferten Systeme unzumutbar werden.

1.2 Modellbasiertes Testen

Die hohe Bedeutung der Systemverifikation gegen eine präzise Systemanforderungsspezifikation für verschiedene Zwecke / Stakeholder erfordert eine rigorose Verifikationsmethode und einen standardisierten, nachvollziehbaren Satz an Prüfungen. Formale Methoden [4] stellen eindeutig den rigorosesten Ansatz dar, können jedoch in der Praxis nicht immer (für alle Prüfungen) angewendet werden. Daher muss

Model-based testing (MBT) can increase the quality, transparency and automation of a test design by generating test cases from a model. For modern modular (system-of-systems) signalling architectures, it becomes particularly attractive for testing on the system level. This article presents an MBT case study for a level crossing controller that covers both the test design and the test execution phases. The most important steps in the applied MBT process and the findings from the case study are discussed.

1 Introduction

1.1 Verification of railway signalling systems

According to the EULYNX Concept [1] and the subsequent EULYNX System Architecture [2], the signalling system units ordered by an infrastructure manager (IM) and delivered by a supplier are shifting from complete interlockings to subsystems such as field element controllers and interlocking core logic. This implies that smaller, but also more systems are being delivered. At the same time, there is also a tendency for more and more validation and verification (V&V) goals to be concentrated at this "delivery level", all of which results in V&V against the specified system requirements:

- the classic need for acceptance checks by the IM remains.
- the system V&V carried out by suppliers as part of development partly converges with the verification concerning the requirements drawn up by the IM due to more precise system requirements specifications (cf. the model-based systems engineering approach adopted by EULYNX [3]).
- the authorisation will have to rely in large part on the certification of the delivered systems, checking conformity to a standardized specification which is known to enforce safety and other properties of interest. Otherwise, the certification effort would become unbearable due to the combinatorics of systems delivered by different suppliers.

1.2 Model-based testing

The importance of system verification against a precise system requirements specification for multiple purposes / stakeholders calls for a rigorous verification method and a comprehensible set of standardised checks. Formal methods [4] clearly constitute the most rigorous approach, but usually cannot always be applied in practice (for all checks). Thus, testing needs to be employed as a less rigorous, but extremely versatile verification method. However, classic manual test design often relies on the test designer's experience and assumptions.

das Testen als weniger rigorose, aber äußerst vielseitige Verifikationsmethode eingesetzt werden. Das klassische manuelle Testdesign stützt sich jedoch häufig auf die Erfahrung und Annahmen des Testdesigners. Diese werden zu einem impliziten Bestandteil der resultierenden Testsuite, was deren Verständlichkeit und Wartbarkeit einschränkt. MBT [5, 6] im engeren Sinne bezieht sich auf Testfälle, die automatisch aus einem Modell gemäß vorgegebener Testauswahlkriterien generiert werden, wodurch sich der manuelle Anteil des Testdesigns auf die Modellerstellung und die Festlegung der Auswahlkriterien reduziert. Das Modell wird zum zentralen Artefakt des Testdesigns, was mehrere Vorteile mit sich bringt:

- Das Testdesign wird nachvollziehbarer – das Modell beschreibt es explizit und weniger verteilt; oft bietet es eine grafische Darstellung, die es zugänglich und verständlich macht, potenziell für mehrere Stakeholder; die Nachverfolgung der im Modell verknüpften Anforderungen wird einfacher.
- Die Qualität der Testsuite steigt – die Formalisierung in einem Modell sorgt für Präzision und ermöglicht automatisierte Konsistenzprüfungen.
- Der Automatisierungsgrad steigt – die Testsuite (Größe) kann durch Anpassung der Testauswahlkriterien oder des Modells schneller und flexibler geändert werden; die Pflege der Testsuite reduziert sich auf Änderungen am Modell (gefolgt von einer Neugenerierung der Testfälle).

1.3 Die Alex-Fallstudie

Die in diesem Beitrag beschriebene MBT-Anwendung war Teil einer umfassenderen Fallstudie [7] im Rahmen des Projekts X2Rail-2 (2017–2020), Arbeitspaket 5. Diese Fallstudie basierte auf einer textuellen Anforderungsspezifikation für einen Bahnübergang (BÜ) namens „Alex" [8] des schwedischen IB Trafikverket. Da der Alex-BÜ mehrere Arten bestehender BÜ ersetzen soll, umfasst die Spezifikation verschiedene Varianten. Für die Fallstudie wurde der Betrachtungsumfang auf die durch ein Stellwerk gesteuerte Variante und auf die BÜ-Steuerungssoftware beschränkt. Damit verblieben dennoch viele Funktionen im Betrachtungsumfang wie lokale BÜ-Steuerung, streckenseitige Geschwindigkeitssensoren, die Steuerung von straßenseitigen Warnlichtern, Schranken, Lautsprechern und Hindernisdetektoren und von streckenseitigen Signalen. Viele dieser Einrichtungen sind dabei in Bezug auf die Anzahl und Variante der gesteuerten Objekte konfigurierbar, einige auch mit Verzögerungswerten. Insgesamt verblieben 133 Anforderungen, die innerhalb des gewählten Betrachtungsumfangs anwendbar waren; Sicherheitsanforderungen waren bereits in der gesamten Spezifikation als solche gekennzeichnet.

Die Projekt-Fallstudie umfasste mehrere Entwicklungsstränge, darunter drei formale Entwicklungsansätze und einen traditionellen Ansatz:

- die refinementbasierte B-Methode (formale Entwicklung),
- modellbasiertes Design mit SCADE,
- konfigurationsbasierte Entwicklung mit Prover iLock (formale Entwicklung),
- contractbasierte Programmierung in SPARK (formale Entwicklung) und
- die auf Ladder-Logic basierende Westrace-Entwicklung (traditioneller Ansatz).

Einige der daraus resultierenden Implementierungen dienten als zu testendes System (SUT – system under test) für den zusätzlich betrachteten Testfallgenerierungs- und -durchführungsstrang (MBT-Strang), über den dieser Beitrag berichtet. Der letztgenannte Strang umfasste Black-Box-Tests, wie sie für die in Abschnitt 1.1 beschriebene „Lieferebene" angemessen sind.

These then become an implicit part of the resulting test suite, thereby limiting its comprehensibility and maintainability.

In a narrower sense, MBT [5, 6] refers to test cases being automatically generated from a model according to given test selection criteria, thereby reducing the manual test design to the tasks of model creation and the choice of criteria. The model thus becomes the central artefact of the test design, resulting in several benefits:

- the test design becomes more transparent – the model is explicit and less distributed; there is often a graphic view that makes it accessible and comprehensible, possibly to multiple stakeholders; it becomes easier to trace the requirements linked in the model.
- the quality of the test suite is increased – formalisation into a model enforces precision and allows for automated consistency checks.
- the level of automation is increased – the test suite (size) can be changed more quickly and flexibly by adapting the test selection criteria or the model; test suite maintenance is reduced to changes in the model (followed by the re-generation of the test cases).

1.3 The Alex case study

The MBT application reported in this article has been part of a larger case study [7] in Work Package 5 of the X2Rail-2 project (2017-2020). This case study was based on a textual requirements specification for a level crossing (LX) called "Alex" [8] that was issued by the Swedish Trafikverket IM. As the Alex LX is supposed to replace several types of existing LX, the specification includes variants. The scope of the case study was limited to the variant controlled by an interlocking and to the LX control software. This still left many features within the scope, such as local control, trackside speed sensors and the control of the road-facing lights, barriers, sound, obstacle detection and track-facing signals. Many of these facilities are configurable in relation to the number and the variants of the controlled objects, as well as with some time delay values. 133 requirements remained applicable within the chosen scope; the safety requirements had already been marked as such throughout the specification.

The project case study included several development tracks, including three formal development approaches and one traditional approach:

- the refinement-based B method (formal development),
- model-based design with SCADE,
- configuration-based development with Prover iLock (formal development),
- contract-based programming in SPARK (formal development) and
- ladder logic based Westrace development (a traditional approach).

Some of the resulting implementations served as the system under test (SUT) for the additional test case generation and execution track (MBT track) reported in this article. The latter track encompassed black box testing as appropriate for the "system delivery level" discussed in Section 1.1.

1.4 The MBT process

Fig. 1 shows the process steps followed in the MBT track along with the tools used in each step. While IBM Rhapsody [9] is a UML/SysML modelling tool used to create a SysML model for test generation, its TestConductor and Automatic Test Generation (ATG) add-ons allow the execution and generation of

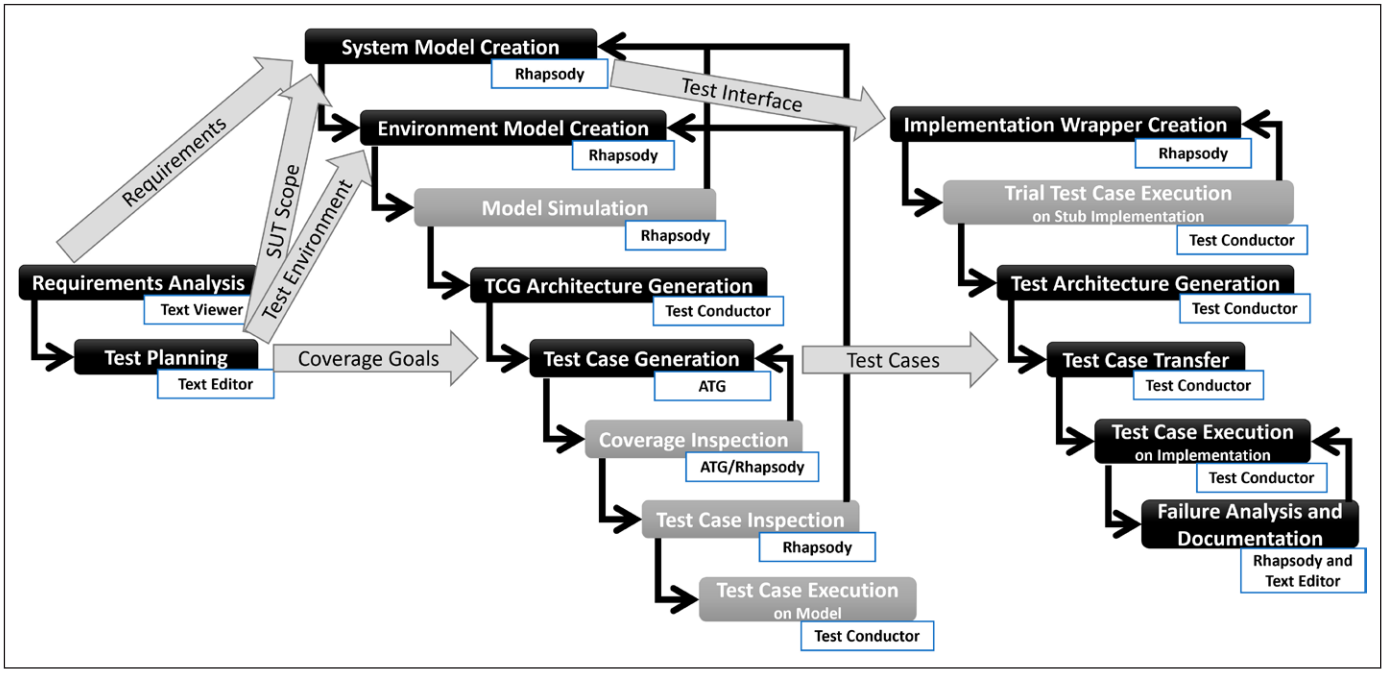

Bild 1: Übersicht über den in der Fallstudie angewandten MBT-Prozess, unterteilt in Vorbereitungsphase (links), Testdesignphase (Mitte) und Testdurchführungsphase (rechts). Reguläre Prozessschritte sind schwarz, Verifizierungsschritte grau und pro Schritt verwendete Tools weiß hinterlegt. Schwarze Pfeile definieren die Reihenfolge der Schritte, einschließlich einiger möglicher Prozessiterationen. Beschriftete Pfeile zwischen den Phasen kennzeichnen die Weitergabe von Ergebnissen.

Fig. 1: An overview of the MBT process applied in the case study divided into the preparation phase (left), the test design phase (centre) and the test execution phase (right). The regular process steps have a black background, the verification steps are grey and the tools used in a step have a white background. The black arrows define the order of the steps, including some possible process iterations. The labelled arrows between the phases denote the transfer of results.

Quelle aller Bilder / *Source of all figs.*: DLR

1.4 MBT-Prozess

Bild 1 zeigt die Schritte des im MBT-Strang verfolgten Prozesses zusammen mit den in jedem Schritt verwendeten Tools. Während IBM Rhapsody [9] ein UML/SysML-Modellierungstool ist, das zur Erstellung eines SysML-Modells für die Testgenerierung verwendet wird, ermöglichen seine Add-ons TestConductor und Automatic Test Generation (ATG) die Ausführung bzw. Generierung von Testfällen. In diesem Beitrag werden nur die wichtigsten Schritte beschrieben. Während der allgemeine Ansatz des MBT-Strangs bereits zuvor in [10] erörtert wurde, berichtet der vorliegende Beitrag erstmals über den Strang als Ganzes, einschließlich Testdurchführung und Gesamtaufwand.

2 Anforderungsanalyse und Testplanung

Sowohl die Entwicklungsstränge als auch der MBT-Strang erstellten präzise Verhaltensmodelle. Daher mussten beide die Anforderungsspezifikation analysieren, relevante Konzepte identifizieren und das System strukturieren. Um redundante Arbeit zu vermeiden und eine gemeinsame Grundlage zu schaffen, wurde eine grundlegende Systemarchitektur in einem tabellarischen, textbasierten Format erstellt. Dieses „Objektmodell" verdeutlichte neben der Strukturierung des Systems in Objekte mit ihren jeweiligen Ein- und Ausgaben auch, welche Timer und internen Zustände pro Objekt vorhanden sein müssen.

Darüber hinaus wurden die möglichen Konfigurationsoptionen und vier konkrete Sätze von Konfigurationsdaten vereinbart, die bei allen Verifikationsaktivitäten verwendet werden sollten. Außerdem wurden gemeinsam die konkrete Struktur und die Übertragungsmechanismen einer gemeinsamen SUT-Schnittstelle (Bild 2) definiert, die zur Integration aller SUT-Implementierungen mit der Test-

test cases, respectively. Only the most important steps are reported in this article. Whereas the general approach of the MBT track was discussed earlier [10], the present article reports on the track as a whole for the first time, including the test execution and overall effort.

2 Requirements analysis and test planning

Both the development tracks and the MBT track created precise behavioural models. As such, both had to analyse the requirements specification so as to identify any relevant concepts and to structure the system. In order to avoid any redundant work and to achieve a common basis in this regard, a basic system architecture in a tabular, text-based format was created. In addition to the structuring of the system into objects with their respective inputs and outputs, this "object model" also clarified which timers and internal states needed to be present per object.

In addition, the possible configuration options and four concrete sets of configuration data to be used throughout all verification activities were agreed on. The concrete structure and transfer mechanisms of the common SUT interface (fig. 2) to be used to connect all the SUT implementations with the test environment were also defined together. Finally, "LX open" was chosen as the initial state for use during testing.

The tracks then continued separately afterwards. This particularly ensured the appropriate independence of the development and test as required by the applicable standards [11]. First, the development tracks identified a set of 31 mostly safety, formally verifiable requirements that were subject to formal verification in the formal development tracks and consequently did not need

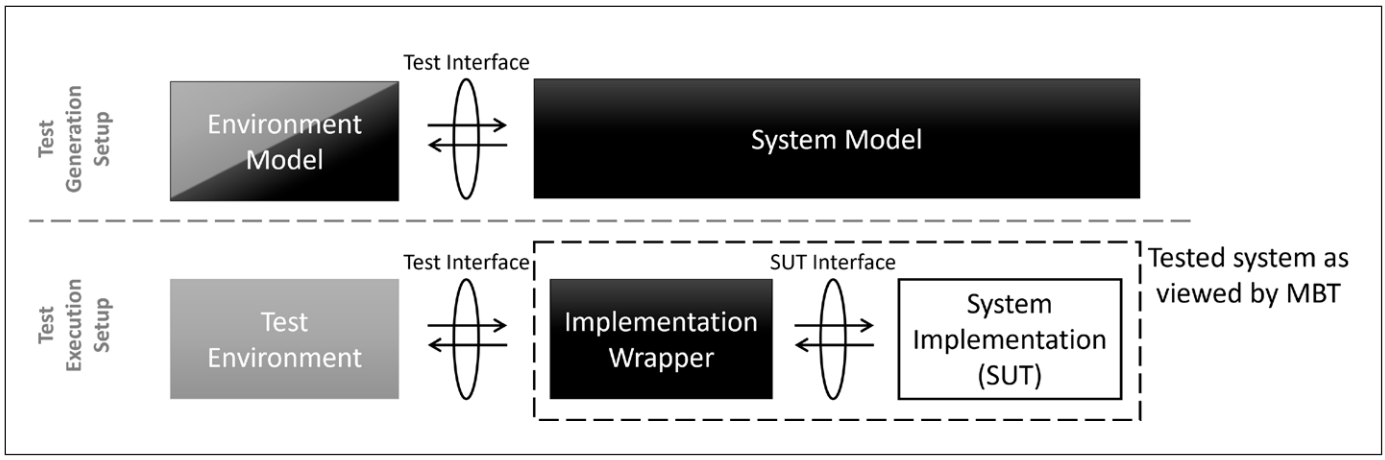

Bild 2: Anordnung, die bei der Testgenerierung (oben) und bei der Testdurchführung (unten) verwendet wird. In Rhapsody modellierte Komponenten sind schwarz, von Rhapsody generierte / bereitgestellte Komponenten grau und außerhalb des Teststrangs implementierte Komponenten weiß dargestellt.

Fig. 2: The setup used at test generation time (top) and test execution time (bottom). The components modelled in Rhapsody have a black background, the components generated / provided by Rhapsody are in grey and the components implemented externally to the testing have a white background.

umgebung verwendet werden sollte. Schließlich wurde „BÜ geöffnet" als Ausgangszustand für die Tests ausgewählt.

Danach wurden die Stränge getrennt fortgesetzt. Dadurch wurde insbesondere die angemessene Unabhängigkeit von Entwicklung und Test gemäß der geltenden Normen [11] sichergestellt. Zunächst wurden im Rahmen der Entwicklungsstränge 31 überwiegend sicherheitsrelevante, formal überprüfbare Anforderungen identifiziert, die einer formalen Verifikation in den formalen Entwicklungssträngen unterzogen wurden und daher nicht mehr getestet werden mussten. Anschließend wurden im Rahmen des MBT-Strangs 26 ergänzende, überwiegend nicht sicherheitsrelevante, testbare Anforderungen identifiziert, die von diesem Strang getestet werden sollten.

3 Modellerstellung

Das für die Testfallgenerierung erstellte Modell ist ein generisches SysML-Modell: Es enthält alle Funktionen, die für alle möglichen Konfigurationen des Alex-BÜ-Controllers erforderlich sind. Im Modell wird ein SysML-Variationspunkt verwendet, um einfach zwischen den vier ausgewählten Konfigurationen wechseln zu können, die als SysML-Varianten abgebildet wurden. Für jede Variante gibt es eine bestimmte Implementierung der Operation „configure", die das Modell entsprechend der jeweiligen Konfiguration instanziiert.

3.1 Modellstruktur

Der obere Teil von Bild 2 zeigt, dass das für die Testfallgenerierung verwendete Modell aus zwei Teilen besteht: Das Umgebungsmodell („Environment Model") beschreibt Annahmen, welche die Systemeingaben einschränken, die in den Testfällen generiert werden können. Das Systemmodell („System Model") beschreibt, wie das System Eingaben verarbeitet, wodurch der Testfallgenerator die erwarteten Systemausgaben für die Testfälle ableiten kann. Zwischen diesen beiden Teilen befindet sich die Testschnittstelle, die alle (uneingeschränkten) Eingaben und Ausgaben umfasst, die in den generierten Testfällen verwendet werden können. Es ist zu beachten, dass sich der Begriff „Testschnittstelle" auf die Generierung und Ausführung von Testfällen bezieht (oberer und unterer Teil von Bild 2) und nicht mit der SUT-Schnittstelle der Systemimplementierungen zu verwechseln ist. Aufgrund der unterschiedlichen Beschaf-

to be tested anymore. The MBT track then identified 26 complementary, mostly non-safety, testable requirements to be tested by that track.

3 Model creation

The model created for the test case generation is a generic SysML model: it contains all the features needed for any possible configurations of the Alex LX controller. The model uses a SysML variation point to easily switch between the four chosen configurations represented as SysML variants. Each variant has a particular implementation of the "configure" operation which instantiates the model according to the given configuration.

3.1 The model structure

The upper part of fig. 2 shows that the model used for test case generation consists of two parts: the environment model describes the assumptions that restrict the system inputs that may be generated in the test cases. The system model describes how the system processes inputs, thereby allowing the test case generator to derive the expected system outputs for the test cases. The test interface between these two parts consists of all the (unrestricted) inputs and outputs that may be used in the generated test cases. Note that the term "test interface" is related to both the generation and execution view of the test cases (the upper and lower parts of fig. 2); it is not to be confused with the SUT interface for the system implementations. In fact, the different natures of the interfaces (message-based vs relay-based high / low values) meant that it was necessary to use implementation wrapper software (the lower part of fig. 2) to translate between them. It should also be noted that both the test and SUT interfaces are dependent on the system configuration as different numbers of controlled devices may be configured.

The structure of the system model is depicted in fig. 3. It is based on the object model (cf. Section 2) and consists of a number of SysML blocks and their associations as shown in the central and lower parts of the figure. They can be instantiated at runtime, possibly several times, e.g. if several signals or barriers are being controlled. A central block called "Alex" is responsible for the instantiation of the rest of the system which is organised hi-

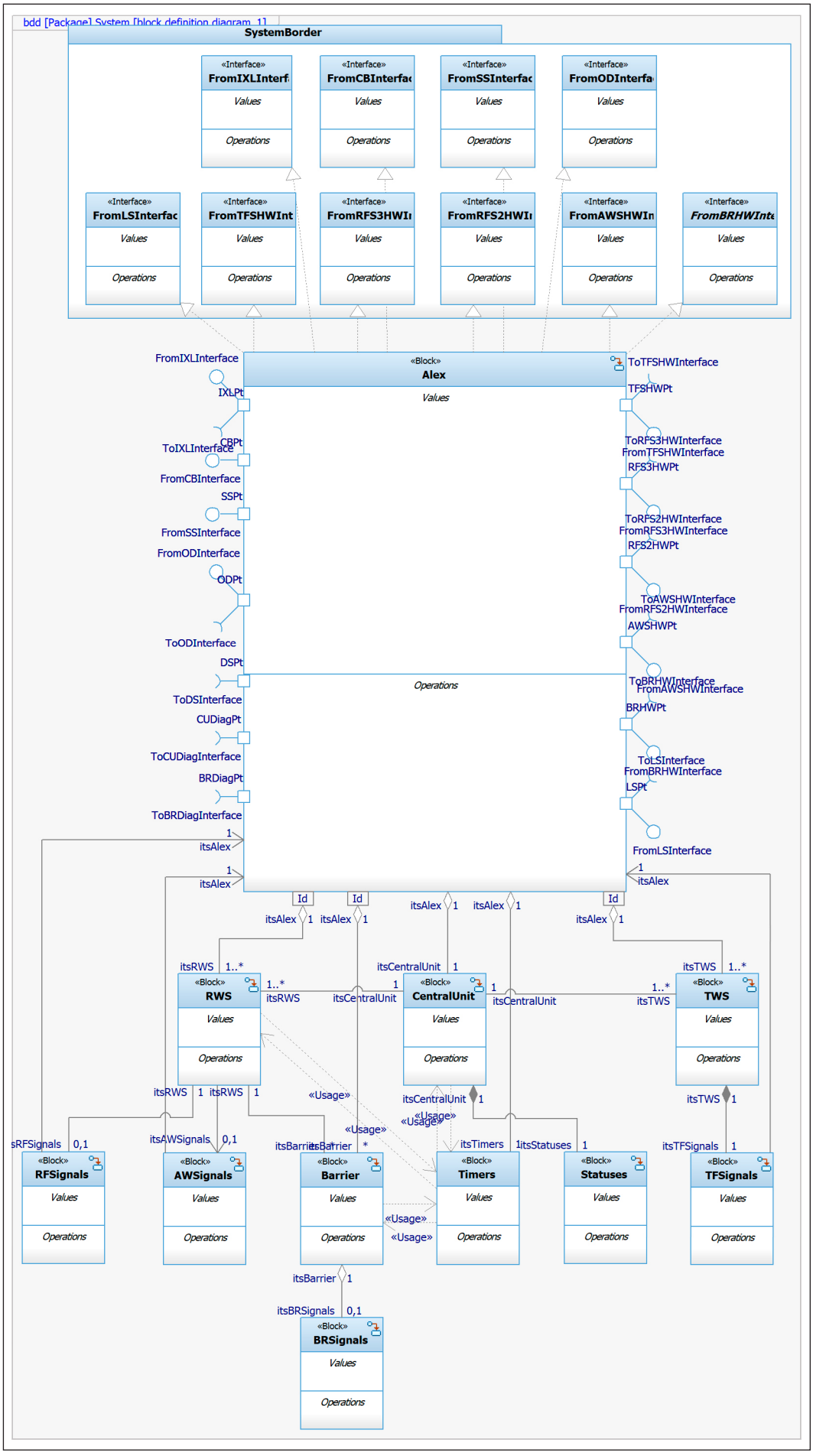

Bild 3: SysML Blockdefinitionsdiagramm des Systemmodells
Fig. 3: The SysML block definition diagram of the system model

fenheit der Schnittstellen (nachrichtenbasiert vs. relaisbasiert mit High-/Low-Werten) war es erforderlich, eine Implementierungs-Wrapper-Software (unterer Teil von Bild 2) für die Übersetzung zwischen beiden zu verwenden. Es ist ebenso zu beachten, dass sowohl die Test- als auch die SUT-Schnittstelle von der Systemkonfiguration abhängig sind, da eine unterschiedliche Anzahl von gesteuerten Elementen konfiguriert werden kann.

Die Struktur des Systemmodells ist in Bild 3 dargestellt. Es basiert auf dem Objektmodell (vgl. Abschnitt 2) und besteht aus einer Reihe von SysML-Blöcken und deren Assoziationen, wie im mittleren und unteren Teil der Abbildung dargestellt. Sie können zur Laufzeit instanziiert werden, möglicherweise mehrmals, z. B. wenn mehrere Signale oder Schranken gesteuert werden. Ein zentraler Block namens „Alex" ist für die Instanziierung des restlichen Systems verantwortlich, das hierarchisch organisiert ist: Die „CentralUnit" steuert potenziell mehrere straßenseitige Warnsysteme („RWS"), eines für jeden Verkehrsfluss, der den BÜ passiert, sowie potenziell mehrere streckenseitige Warnsysteme („TWS"), eines für jedes Gleis, das den BÜ passiert. RWS und TWS steuern wiederum potenziell mehrere straßenseitige bzw. streckenseitige Elemente, wie z. B. Signale (Lichter) und Schranken. Der zentrale Alex-Block stellt auch die Testschnittstelle unter Verwendung von SysML-Standardports dar, die als quadratische Kästchen am Rand des Blocks zu sehen sind. Die eingehenden und ausgehenden Nachrichten für jeden Port sind als Teil eines SysML-Schnittstellenblocks definiert; für die eingehenden Nachrichten sind diese Schnittstellenblöcke im oberen Teil von Bild 3 dargestellt.

Die Struktur des Umgebungsmodells wurde automatisch mit der Funktion „Testarchitekturgenerierung" von TestConductor erstellt. Sie spiegelt die externe Schnittstelle des Systemmodells: Für jeden Systemmodellport gibt es einen Umgebungs-SysML-Block mit einem Port mit umgekehrten Eingangs- und Ausgangsnachrichten.

3.2 Verhaltensmodellierung

Das Verhalten wurde so weit wie möglich durch Zustandsdiagramme modelliert (und weniger durch SysML-Operationen), da dies aussagekräftige strukturelle Abdeckungsmessungen der generierten Testfälle (siehe Abschnitt 4.1 unten) und visuelles Feedback ermöglicht. Jedem Block kann ein Zustandsdiagramm zugewiesen werden, das zur Laufzeit zusammen mit dem zugehörigen Block instanziiert wird. In Bild 3 sind die Blöcke des Systemmodells, die Zustandsdiagramme besitzen, an dem kleinen Zustandsdiagramm-Symbol in der oberen rechten Ecke zu erkennen. Ein Auszug aus dem Zustandsdiagramm des Blocks „CentralUnit" ist in Bild 4 dargestellt.

erarchically: the "CentralUnit" controls potential multiple roadside warning systems ("RWS"), one for each traffic flow passing the LX, as well as potential multiple trackside warning systems ("TWS"), one for each railway track crossing the LX. The RWS and TWS in turn control potential multiple roadside and trackside devices respectively, such as signals (lights) and barriers. The central Alex block also constitutes the test interface using SysML standard ports shown as square boxes on the border of the block. The in and outgoing messages for each port are defined as part of a SysML interface block; the interface blocks for the ingoing messages are shown in the upper part of fig. 3.

The structure of the environment model was automatically generated using the "test architecture generation" feature of TestConductor. It mirrors the external interface of the system model: each system model port has an environment SysML block with a port with switched in and output messages.

3.2 Behaviour modelling

The behaviour has been modelled using statecharts as much as reasonably possible (and less using SysML operations), because this enables meaningful structural coverage measures of the generated test cases (see Section 4.1 below) and enables visual feedback. Each block can be assigned a statechart that is instantiated together with its owning block at runtime. In fig. 3, the system model blocks with statecharts can be recognised by the small statechart symbol in the upper right-hand corner. An excerpt from the "CentralUnit" block statechart is shown in fig. 4.

A model should abstract the real system behaviour appropriately. Such abstraction may be undertaken by means of the aggregation or omission of details, for example. An example of omissions in the system model involves the functionality that prevents barrier commands when moved manually (a requirement defined as safety relevant and as such subject to formal verification). An example of behaviour aggregation can be found in the fact that different possible points in time (no requirements pertaining to this exist) for taking back commands to begin or end obstacle detection are abstracted to check that this has been done at the latest possible point in time.

All in all, the system model comprises 136 states, 199 transitions and 153 operations that were taken into consideration for test case generation coverage. Furthermore, the 26 requirements mentioned in Section 2 have been assigned to model elements, e.g. see the assignment of two requirements to state transitions in fig. 4. As the "satisfy" relationship indicates, the meaning of

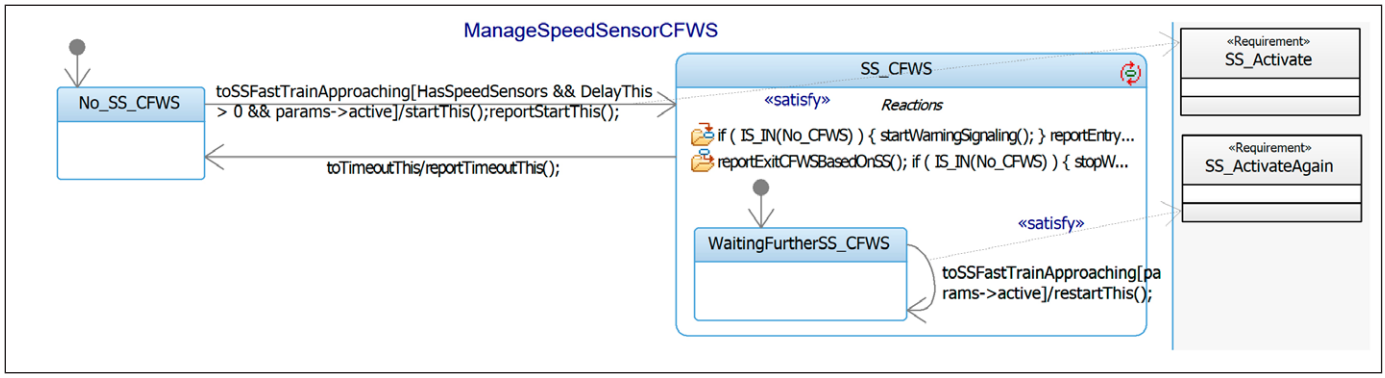

Bild 4: Eine parallele Region im Zustandsdiagramm des Blocks „CentralUnit" zur Verwaltung der geschwindigkeitsabhängigen Ansteuerung des Schließens des BÜ, mit zwei Anforderungen, die mit Zustandsübergängen verbunden sind

Fig. 4: A parallel region in the statechart of the "CentralUnit" block for managing speed sensor-based calls for LX warning signalling with two requirements associated to transitions

Ein Modell sollte das tatsächliche Systemverhalten angemessen abstrahieren. Eine solche Abstraktion kann beispielsweise durch Aggregation oder Auslassung von Details erfolgen. Ein Beispiel für eine Auslassung im erstellten Systemmodell ist Funktionalität, die eine Kommandierung der Schranken verhindert, wenn diese manuell bewegt werden (eine Anforderung, die als sicherheitsrelevant gekennzeichnet ist und daher bereits der formalen Verifikation unterlag). Ein Beispiel für eine Verhaltensaggregation ist, dass verschiedene mögliche Zeitpunkte (die durch die Anforderungen nicht näher spezifiziert sind) für die Rücknahme von Befehlen zum Starten oder Beenden der Hinderniserkennung zu einer Überprüfung abstrahiert werden, ob dies zum spätestmöglichen Zeitpunkt erfolgt ist.

Insgesamt umfasst das Systemmodell 136 Zustände, 199 Zustandsübergänge und 153 Operationen, die für die Testfallgenerierung berücksichtigt wurden. Darüber hinaus wurden die 26 in Abschnitt 2 genannten Anforderungen Modellelementen zugeordnet, siehe z. B. die Zuordnung von zwei Anforderungen zu Zustandsübergängen in Bild 4. Wie die Beziehung „satisfy" angibt, bedeutet eine solche Zuordnung, dass immer dann, wenn das Modellelement durch einen Testfall abgedeckt ist, die Anforderung durch diesen Testfall überprüft wird.

Das Verhalten des Umgebungsmodells wurde (neben Operationen) ebenfalls mit Zustandsdiagrammen modelliert. Technisch gesehen ist dies für die Testgenerierung nicht erforderlich. Insbesondere bei größeren Systemmodellen kann die Einschränkung des Umgebungsverhaltens durch die Implementierung von Annahmen jedoch für eine erfolgreiche Testgenerierung von entscheidender Bedeutung sein, da

- dadurch die Anzahl und die Wertebereiche der zu untersuchenden Eingaben reduziert werden können,
- der Fokus der generierten Testfälle durch die Vorgabe bestimmter Arten von Umgebungsverhalten (möglich, realistisch, nominell, fehlerhaft usw.) gesetzt werden kann und
- das Systemmodell nicht mehr beliebige Eingabekombinationen und -sequenzen verarbeiten können muss, was zu einem kleineren und zielgenaueren Systemmodell führt.

Vorerst wurde beschlossen, die Testgenerierung auf normales Verhalten zu beschränken. Folglich wurden Stub-Implementierungen aller Umgebungssysteme erstellt, die normalerweise ein deterministisches Verhalten in Bezug auf die Steuerausgaben des SUT aufweisen. Auf diese Weise wurden beispielsweise Warnlichter oder Lautsprecher gezwungen, rückzumelden, dass sie sich ein- oder ausschalten, wenn dies entsprechend kommandiert wurde. Im Gegensatz dazu wurden Eingaben von Stellwerk, Steuerungskasten, Geschwindigkeitssensor und Hindernisdetektor für die Generierung offengelassen. Dies bedeutete eine erhebliche Reduzierung von etwa 40 auf etwa neun offene boolesche Eingaben für eine durchschnittliche Konfiguration. Die meisten Umgebungsmodelle waren sehr einfach; die einzige Ausnahme war das Schrankenmodell, das die Bewegung, Blockierung und Position des Schlagbaums erfassen musste und daher Timer und Algorithmen für die Schrankenbewegung enthielt.

4 Testgenerierung

4.1 Einstellungen

Neben dem Umgebungsmodell ist es in Rhapsody ATG auch möglich, generierte Testeingaben über Testgenerierungseinstellungen einzuschränken. In der Fallstudie wurde dies für einige Parameter genutzt, die IDs von Elementen enthielten, von denen Eingabenachrichten gesendet wurden und die entsprechend auf die ID-Werte beschränkt waren, die tatsächlich in der betrachteten Konfiguration vorkamen.

Als Testauswahlkriterium wurde die strukturelle Abdeckung derjenigen SysML-Blöcke aus dem Systemmodell gewählt, deren Zu-

such an assignment lies in the fact that the requirement is tested by a test case, whenever the model element is covered by said test case.

The environment model behaviour has also been modelled using statecharts (alongside operations). Technically this is not required to perform test generation, but restricting the environment behaviour by implementing assumptions on it can be essential for successful test generation, particularly for larger system models, because

- it can reduce the number and value ranges of the inputs that need to be explored,
- it can set the focus of the generated test cases by imposing certain kinds of environment behaviour (possible, realistic, nominal, faulty, etc.) and
- it relieves the system model from caring about arbitrary input combinations and sequences, which leads to a smaller and more to the point system model.

For the time being, a decision has been made to restrict the test generation to normal behaviour. Consequently, stub implementations of all the environment systems that normally exhibit deterministic behaviour with regard to the control outputs of the SUT were created. As such, the lights or sounds were forced to report that they had turned on / off whenever they were commanded to do so. By contrast, the inputs from the interlocking, control box, speed sensor and obstacle detector were left open to be generated. This involved a considerable reduction from around 40 to about nine open Boolean inputs for an average configuration. Mostly, the environment models were very simple; the only exception was the barrier model, which needed to capture barrier boom movement, blockage and position and thus included timers and algorithms for barrier movement.

4 Test generation

4.1 Settings

In addition to the environmental model, it is also possible to restrict generated test inputs using the test generation settings in Rhapsody ATG. The case study did this for some parameters with device IDs from which input messages were sent so that they were restricted to those ID values that actually occurred in the configuration under consideration.

The structural coverage of those SysML blocks from the system model whose statecharts actually realise the requirements to be tested was chosen as the test selection criteria. This means that the test generator aimed to cover all the states, state transitions and operations from the CentralUnit, Statuses, RWS, TWS, TFSignals, RFSignals, AWSignals, Barrier and BRSignals blocks (cf. fig. 3).

The beam search algorithm implemented in ATG was chosen with a beam width of 500 for an efficient search.

4.2 Results

Test suites were generated for all four configurations. The generation time varied between about 0.5 and 5 minutes according to the configuration size, excluding the pre and post-calculation times not measured by ATG. Some characteristic figures are provided in tab. 1 (the test case number in parentheses for configuration 17-008 refers to a test suite variant). A comparison of the total coverage with the overall numbers provided in Sections 3.2 and 2 shows that all the requirements, but not all the model elements were covered. The latter is caused by some model elements being unreachable due to the model's current construc-

Alex-Konfiguration	17-008	17-009	17-010	17-012	Gesamtabdeckung
Anzahl Testfälle	36 (38)	43	29	30	-
Anzahl abgedeckter Modellelemente (Zustände / Zustandsübergänge / Operation)	120/149/131	119/153/128	111/129/116	102/121/104	125/162/144
Anzahl abgedeckter Anforderungen	24	20	16	16	26

Tab. 1: Kenngrößen der für die verschiedenen Alex-BÜ-Konfigurationen generierten Testsuiten

Alex configuration	17-008	17-009	17-010	17-012	Total coverage
Number of test cases	36 (38)	43	29	30	-
Number of covered model elements (states / state transitions / operations)	120/149/131	119/153/128	111/129/116	102/121/104	125/162/144
Number of requirements covered	24	20	16	16	26

Tab. 1: Characteristic test suite figures generated for the different Alex LX configurations

standsdiagramme die zu testenden Anforderungen umsetzen. Konkret bedeutete dies, dass der Testgenerator darauf abzielte, alle Zustände, Zustandsübergänge und Operationen aus den Blöcken CentralUnit, Statuses, RWS, TWS, TFSignals, RFSignals, AWSignals, Barrier und BRSignals abzudecken (vgl. Bild 3).
Für eine effiziente Testfallsuche wurde der in ATG implementierte Beam-Search-Algorithmus mit einer Beam-Breite von 500 gewählt.

4.2 Ergebnisse

Für alle vier Konfigurationen wurden Testsuiten generiert. Die Generierungszeit variierte je nach Konfigurationsgröße zwischen etwa 0,5 und 5 Minuten, ohne die von ATG nicht gemessenen Vor- und Nachberechnungszeiten. Einige Kenngrößen sind in Tab. 1 aufgeführt (die Testfallanzahl in Klammern für die Konfiguration 17-008 bezieht sich auf eine Testsuite-Variante). Beim Vergleich der Gesamtabdeckung mit den in Abschnitten 3.2 und 2 angegebenen absoluten Zahlen wird ersichtlich, dass alle Anforderungen, jedoch nicht alle Modellelemente abgedeckt wurden. Letzteres ist darauf zurückzuführen, dass einige Modellelemente aufgrund des aktuellen Bearbeitungsstatus des Modells bei Generierung der Testfälle nicht erreichbar waren (nicht bereinigtes Modell). Eine detaillierte Abdeckungsanalyse bestätigte, dass tatsächlich alle erreichbaren Modellelemente für jede Konfiguration abgedeckt wurden.
Die durchschnittliche Anzahl der Testschritte (Eingabe- und Ausgabenachrichten) pro Testfall kann grob auf 30 geschätzt werden, wobei es bei einigen Testfällen zu erheblichen Abweichungen kommen kann (von wenigen bis zu über 250 Schritten). Ein Beispiel eines Testfalls ist in Bild 5 dargestellt.

5 Testdurchführung

Bei drei der fünf Implementierungen, die aus den in Abschnitt 1.3 aufgeführten Entwicklungssträngen resultierten, wurden alle Testfälle aus den Testsuiten für die vier Konfigurationen (Tab. 1) nach erfolgreicher SUT-Integration mit TestConductor ausgeführt. Durch die automatisierte Generierung der Testausführungsarchitektur und die Nutzung der Batch-Testfunktion von TestConductor konnte ein relativ hoher Automatisierungsgrad erreicht werden. Einige Änderungen wurden manuell vorgenommen, wie z. B. das Hinzufügen von Vor- und Nachausführungscode für das Starten und Beenden der SUT-Ausführung vor und nach jedem Testfall sowie für die Umleitung der Modellausgaben in eine Datei pro Testfall zu Protokollierungszwecken.
Insgesamt wurden 166 von 416 durchgeführten Tests (vollständig) bestanden; je nach Implementierung wurden zwölf bis 18 der 26 getesteten Anforderungen dadurch abgedeckt. Die Ausführung mit TestConductor lieferte eingefärbte Testfall-Sequenzdiagramme, die darstellen, welche Ein- und Ausgaben während des Testlaufs erfolgreich übertragen wurden und welche nicht.

tion status when the test cases were generated (the model had not been cleaned up). A detailed coverage analysis confirmed that all the reachable model elements had indeed been covered for each configuration.
A rough estimate of the average number of test steps (in and output messages) per test case amounts to 30, including large deviations for some test cases (a few to over 250 steps). An example test case is shown in fig. 5.

5 Test execution

All the test cases from the test suites for the four configurations (tab. 1) have been executed with TestConductor after successful SUT integration for three out of the five implementations resulting from the development tracks listed in Section 1.3. A relatively high grade of automation was able to be achieved due to the automated test execution architecture generation and the use of the batch testing capability in TestConductor. Some modifications have been made manually, such as the addition of pre and post-execution code for the SUT startup and shutdown after each test case and for redirecting the model outputs into a file per test case for logging purposes.
In total, 166 test cases out of the 416 tested passed (completely); twelve to 18 of the 26 tested requirements were covered depending on the implementation. Execution using TestConductor resulted in coloured test case sequence diagrams visualising which inputs and outputs were successfully transmitted during the test run and which were not.
70 issues responsible for the failing test cases were reported (issues occurring in multiple implementations and configurations were reported multiple times, the estimated number of such double-counts is 13). On several occasions, a single issue was responsible for most of the failed test cases of one tested implementation configuration. Most of the issues could be associated with one or more requirements that had been violated or interpreted differently by the implementation and the test; altogether, this involved 21 of the requirements within the scope. The issues can be categorised as follows (one issue was related to two categories so it has been counted as half an issue for both):

1. 35 (50 %) of the issues were rated as implementation failures. The majority of them seemed to be related to the code from the model generated in the given development track as the main source of the tested code; some errors related to extra handwritten code. Single errors in the configuration data and in the test interface realisation were also found. Interestingly, one error could be traced back to a problem in a code generator.
2. 17.5 (25 %) issues were related to a different interpretation of the Alex tender specification by the implementation and the

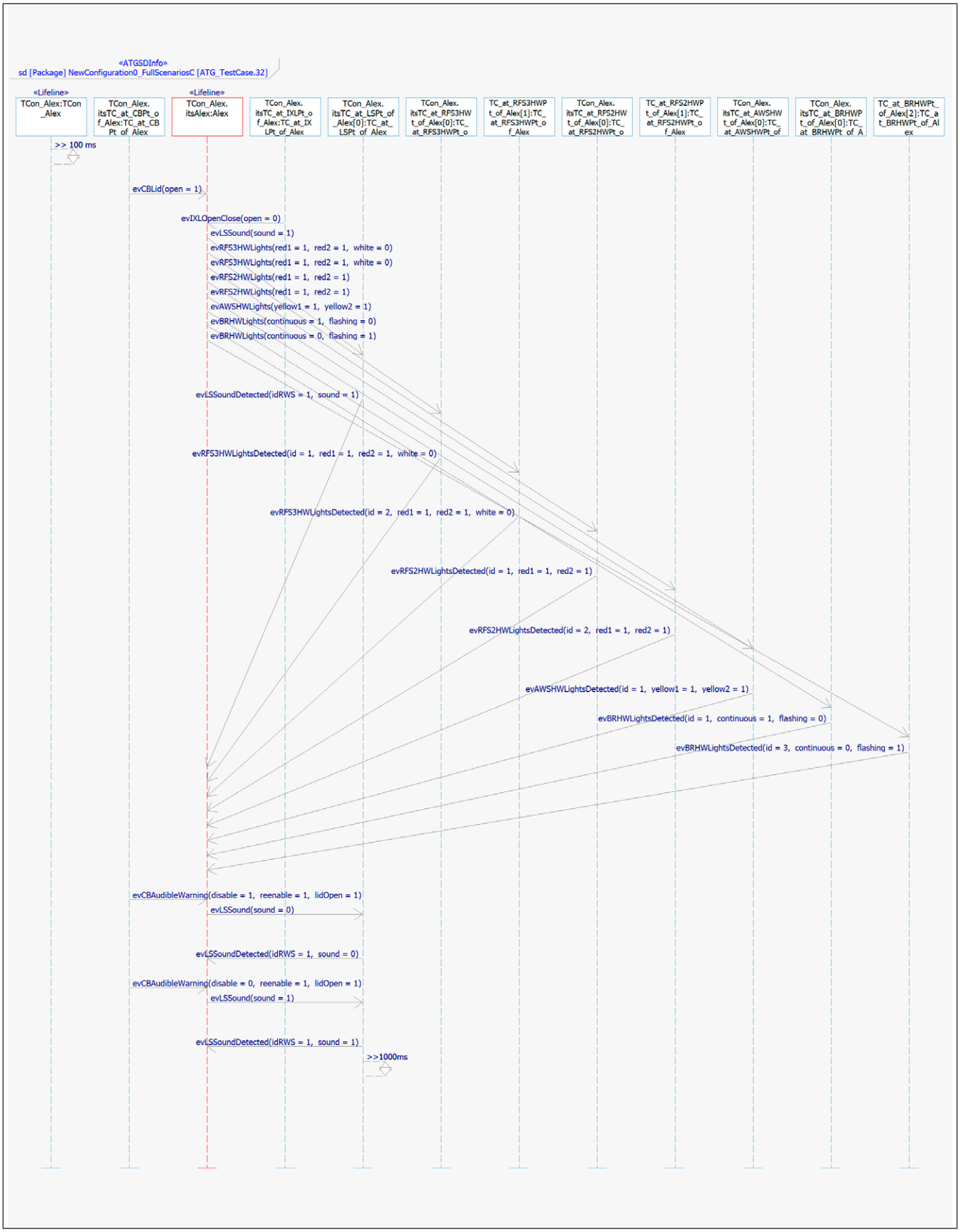

Bild 5: Ein ATG-generierter Testfall als Sequenzdiagramm. Das SUT (rote Lifeline) empfängt Eingaben von und sendet erwartete Ausgaben an die Komponenten der Testumgebung (blaue Lifelines).

Fig. 5: An ATG-generated test case as a sequence diagram. The SUT (red lifeline) receives inputs from and is expected to send outputs to the components of the test environment (blue lifelines).

Es wurden 70 Probleme gefunden, die für die fehlgeschlagenen Testfälle verantwortlich waren (Probleme, die in mehreren Implementierungen und Konfigurationen auftraten, wurden mehrfach gezählt, die geschätzte Anzahl solcher Doppelzählungen beträgt 13). Mehrfach war ein einzelnes Problem für die meisten fehlgeschlagenen Testfälle einer getesteten Konfiguration einer Implementierung verantwortlich. Die meisten Probleme konnten mit einer oder mehreren Anforderungen in Verbindung gebracht werden, die durch die Implementierung und den Test verletzt oder unterschiedlich interpretiert wurden; insgesamt betraf dies 21 der Anforderungen im Betrachtungsumfang. Die Probleme lassen sich wie folgt kategorisieren (ein Problem bezog sich auf zwei Kategorien und wurde daher für beide als halbes Problem gezählt):

1. 35 (50 %) der Probleme wurden als Implementierungsfehler bewertet. Die meisten davon schienen mit dem Code zusammenzuhängen, der aus dem in dem jeweiligen Entwicklungsstrang erstellten Modell als Hauptquelle des getesteten Codes generiert wurde; einige Fehler bezogen sich auf handgeschriebenen zusätzlichen Code. Es wurden auch einzelne Fehler in den Konfigurationsdaten und in der Realisierung der Testschnittstelle gefunden. Interessanterweise konnte ein Fehler auf ein Problem in einem Codegenerator zurückgeführt werden.
2. 17,5 (25 %) waren Probleme im Zusammenhang mit einer unterschiedlichen Auslegung der Alex-Anforderungsspezifikation durch die Implementierung und den Test. Ohne den Verfasser der Spezifikation lässt sich nicht mit Sicherheit entscheiden, ob die damit verbundenen Anforderungen absichtlich Interpretationsspielraum lassen oder ungenau / unvollständig sind. In den meisten Fällen schienen beide Auslegungen in Ordnung zu sein.
3. 13 (19 %) wurden als Timing- / Reihenfolge-Probleme bewertet. Meistens sind diese Probleme auf Ausgaben zurückzuführen, die von der Implementierung ein oder zwei Zyklen später als vom Test erwartet erzeugt werden. Probleme in dieser Kategorie können als ein Problem der Kombination einer sequenziellen Schnittstelle auf der Testseite mit einer parallelen Schnittstelle auf der Implementierungsseite angesehen werden; durch mehr Aufwand bei der Übersetzung zwischen diesen beiden Schnittstellen könnte die Anzahl solcher Probleme minimiert werden.
4. 4,5 (6 %) wurden als Fehler auf der Testseite bewertet. Dazu gehören Programmier- / Modellierungsfehler im Implementierungs-Wrapper und Fälle von über- / unterspezifiziertem Verhalten im Modell. Probleme dieser Kategorie wurden sofort nach ihrer Entdeckung korrigiert, sodass sie andere Probleme bei nachfolgenden Tests nicht maskieren konnten.

Es wurde eine Vielzahl von Ursachen für die Probleme identifiziert, von denen jede ein- bis mehrmals auftrat, aber keine als vorherrschend bezeichnet werden kann. Dies zeigt, dass generierte Testfälle in der Lage sind, verschiedenste Arten von Softwarefehlern zu finden.

6 Fazit

Die Fallstudie bestätigt, dass MBT für mittelgroße Systeme der Leit- und Sicherungstechnik durchführbar ist (z. B. angemessene Anzahl und Größe der Testfälle, akzeptable Testgenerierungszeiten) und verschiedene Fehler in der Software aufdecken kann. Sie basierte auf einer BÜ-Spezifikation des schwedischen IB Trafikverket und konzentrierte sich auf das Testen des darin spezifizierten normalen Verhaltens des Controllers. Folgende Schlussfolgerungen lassen sich aus der Fallstudie ziehen:

- Die in Abschnitt 1.2 aufgeführten allgemeinen Vorteile von MBT können alle drei bestätigt werden. Insbesondere die Tatsache, dass nach einer sehr kurzen Pre-Test-Phase während des Batch-Testens keine Änderungen am Modell / an den Testsuiten mehr erforderlich waren, belegte glaubhaft, dass MBT zu Testsuiten von hoher Qualität führen

test. Without the specification's creator, it is impossible to decide for sure whether the associated requirements intentionally leave room for interpretation or are imprecise / incomplete. It seemed in most cases that both interpretations were fine.

3. 13 (19 %) were rated timing / order problems. Most often, these issues go back to outputs that the implementation produces one or two cycles later than the test expects. The issues in this category can be seen as a problem resulting from the combination of a sequential interface on the test side with a parallel interface on the implementation side; putting more effort into the translation between these two could minimise the number of such issues.
4. 4.5 (6 %) were rated failures on the test side. This includes programming / modelling errors in the implementation wrapper and cases of over / underspecified behaviour in the model. The issues in this category were corrected immediately upon being discovered, so that they could not mask any other issues during the following tests.

A wide variety of causes of these issues has been identified, each of them occurring one to several times, but none of which could be called predominant. This shows that the generated test cases are able to find all kinds of different software errors.

6 Conclusion

The case study confirmed that MBT of mid-sized signalling systems is feasible (e.g. reasonable test case number and size, acceptable test generation times) and capable of uncovering various software errors. It was based on an LX specification by the Swedish IM Trafikverket and focused on testing the normal behaviour of the controller specified therein. The findings from the case study include the following:

- All three general benefits of MBT listed in Section 1.2 can be confirmed. In particular, the fact that no more changes of the model / test suites were necessary during batch testing after a very short phase of preliminary testing credibly demonstrated that MBT can result in high-quality test suites, even though the process verification steps (the grey steps in fig. 1) had not been applied to the full extent.
- The overall effort for the MBT track amounted to about 810 hours of work, most of them related to test execution and failed test case analysis (~200 hours) and model creation (~165 hours). This included nearly no regression testing, but still seems quite moderate for the creation of about 140 test cases pertaining to four different system configurations and their application in three test campaigns.
- Although the Alex specification is rated good quality, the tension between the need for (a) precise and consistent specifications and (b) capturing behaviour at the right level of abstraction / flexibility when formalising the requirements in a model was clearly noticeable: 25 % of the failed test cases were due to different interpretations of the requirements. The learnings gained from this involve systematically inspecting each modelling decision to ensure that it is justified by the specification, only accepting good quality specifications for MBT and maybe also that a good model needs to mature through feedback.
- Compared to manual test design, typical shifts reported within an MBT context include less realistic and more redundant test cases. The former is true for the current case study; still, several of the short to mid-sized generated test cases exhibited realistic sequences. The latter has not been experienced to a noteworthy extent.
- When MBT is applied alongside formal verification / development, synergies can be exploited during the requirement analysis

kann – und dies, obwohl die Prozessverifikationsschritte (graue Schritte in Bild 1) in der Fallstudie nicht vollumfänglich angewendet wurden.
- Der Gesamtaufwand für den MBT-Strang belief sich auf etwa 810 Arbeitsstunden, von denen die meisten für die Testdurchführung und Analyse fehlgeschlagener Testfälle (~200 Stunden) sowie die Modellerstellung (~165 Stunden) benötigt wurden. Auch wenn in dieser Stundenzahl nahezu keine Regressionstests enthalten sind, erscheint sie für die Erstellung von etwa 140 Testfällen für vier verschiedene Systemkonfigurationen und deren Anwendung in drei Testkampagnen recht moderat.
- Obwohl die Alex-Spezifikation als qualitativ hochwertig eingestuft wird, war die Spannung zwischen der Notwendigkeit (a) präziser und konsistenter Spezifikationen und (b) der Erfassung des Verhaltens auf dem richtigen Abstraktions-/Flexibilitätsniveau bei der Formalisierung von Anforderungen in einem Modell deutlich spürbar: 25 % der fehlgeschlagenen Testfälle waren auf unterschiedliche Interpretationen der Anforderungen zurückzuführen. Die daraus gewonnenen Erkenntnisse sind, jede Modellierungsentscheidung systematisch daraufhin zu überprüfen, ob sie durch die Spezifikation gerechtfertigt ist, weiterhin nur Spezifikationen von guter Qualität für MBT zu akzeptieren und möglicherweise auch, dass ein gutes Modell durch Feedback reifen muss.
- Typische Verschiebungen, die im Zusammenhang mit MBT im Vergleich zum manuellen Testdesign berichtet werden, sind weniger realistische und redundantere Testfälle. Ersteres trifft auf die aktuelle Fallstudie zu; dennoch weisen mehrere der generierten Testfälle kleiner bis mittlerer Größe realistische Sequenzen auf. Letzteres wurde nicht in nennenswertem Umfang beobachtet.
- Wenn MBT zusammen mit formaler Verifikation/formaler Entwicklung angewendet wird, können Synergien während der Anforderungsanalyse und der Formalisierungsphase genutzt werden. Andererseits besteht der Eindruck, dass sich die Anforderungen, die mit formalen Methoden bzw. MBT verifiziert werden können, weitgehend überschneiden, sodass in konkreten Projekten zumindest untersucht werden sollte, ob ein einziger Ansatz ausreicht.

Weitere spezielle Studien wären erforderlich, um fundierte Ergebnisse zu möglichen Einsparungen im Vergleich zu einem klassischen Testdesignansatz zu liefern und die Grenzen von Testgenerierungswerkzeugen in Bezug auf verschiedene Systemtypen und -größen zu bestimmen. Ein weiteres interessantes Thema könnten verschiedene Strategien für die Wartung generierter Testsuiten sein. ■

Danksagung und Disclaimer

Diese Arbeit wurde im Rahmen des Projekts X2Rail-2 durchgeführt. Dieses Projekt wurde vom Joint Undertaking (JU) Shift2Rail im Rahmen des Forschungs- und Innovationsprogramms „Horizon 2020" der Europäischen Union unter der Fördervereinbarung Nr. 777465 finanziert. Diese Arbeit gibt ausschließlich die Meinung des Autors wieder; das JU ist nicht verantwortlich für die Verwendung der darin enthaltenen Informationen.

DOI: 10.61067/260652

AUTOR | AUTHOR

Dr. Daniel Schwencke
Wissenschaftlicher Mitarbeiter Schienensysteme und -technologien /
Research Associate, Rail Systems and Technologies
Deutsches Zentrum für Luft- und Raumfahrt e. V. (DLR)
Institut für Verkehrssystemtechnik
Anschrift / *Address*: Lilienthalplatz 7, D-38108 Braunschweig
E-Mail: daniel.schwencke@dlr.de

and formalisation phase. On the other hand, the impression exists that the sets of requirements that can be verified by formal methods and MBT respectively may overlap to a large extent, so concrete projects should at least investigate whether a single approach can be sufficient.

Further dedicated studies would be required to provide substantiated results on possible savings compared to a classic test design approach and to determine the limits of test generation tools regarding different system types and sizes. Another topic of interest may be different strategies for maintaining generated test suites. ■

Acknowledgement and disclaimer

This work has been conducted within the X2Rail-2 project. The project has received funding under grant agreement No 777465 from the Shift2Rail Joint Undertaking (JU) in the European Union's Horizon 2020 research and innovation programme. This work only reflects the author's view; the JU is not responsible for any use that may be made of the information it contains.

LITERATUR | LITERATURE

[1] EULYNX Partners (2019) EULYNX Concept. Dokument Eu.Doc.6, Version 2.0 (2.A), veröffentlich mit EULYNX Baseline 3
[2] EULYNX Partners (2025) EULYNX System definition: Appendix A1 EULYNX System. Dokument Eu.Doc.7_A1, Version 4.2 (4.A), veröffentlich mit EULYNX Baseline 4
[3] EULYNX Partners (2017) System engineering process. Dokument Eu.Doc.27, Version 2.0 (0.A), veröffentlich mit EULYNX Baseline 2
[4] Löfving, C.; Borälv, A.; Mejia, L.-F.; Vitiello, A.; Berglehner, R.; Schwencke, D. (2018): Formal Methods (Taxonomy and Survey), Proposed Methods and Applications. Projektbericht D5.1, 51 Seiten
[5] Winter, M.; Roßner, T.; Brandes, C.; Götz, H. (2016): Basiswissen modellbasierter Test. 2. vollständig überarbeitete und aktualisierte Auflage, dpunkt.verlag GmbH, Heidelberg
[6] Utting, M.; Pretschner, A.; Legeard, B. (2012): A taxonomy of model-based testing approaches. Softw. Test. Verif. Reliab. 22:297–312, doi: 10.1002/stvr.456
[7] Borälv, A.; Berglehner, R.; Cherif, I.; Fredholm, D.; Hansen, D.; Magro, J.; Mejia, L.-F.; Mentré, D.; Rasheeq, A.; Schwencke, D.; Werner, T. (2022): Holistic Study of Formal Methods and Standardization in Specification, Development, Verification and Validation of Railway Signalling System Software. World Congress on Railway Research 2022, Birmingham, UK
[8] Alex Projektteam bei Trafikverket (2017): Requirements for functions and interfaces in the Alex product. Dokument ALEX17-001, Version 1.1, englische Übersetzung des Dokuments ALEX16-006, Version 1.1
[9] IBM Engineering Systems Design Rhapsody documentation (2026), https://www.ibm.com/docs/en/engineering-lifecycle-management-suite/design-rhapsody, letzter Zugriff am 10.03.2026 um 16:42
[10] Schwencke, D. (2020): Test Case Generation for a Level Crossing Controller. In Meyer zu Hörste, M. (ed.): Proceedings of the 2nd SmartRaCon Scientific Seminar. Berichte aus dem DLR-Institut für Verkehrssystemtechnik, Bd. 37, S. 89–98
[11] EN 50128:2011 Railway applications – Communication, signalling and processing systems – Software for railway control and protection systems