OETCS/WP4/CSM – 01/00

openETCS

ITEA2 Project
Call 6 11025
2012 – 2015

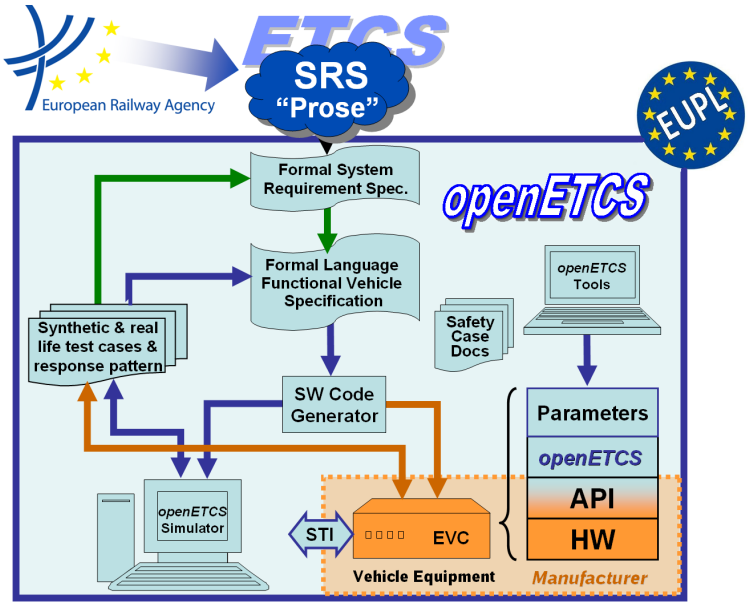
Work Package 4: “Verification & Validation Strategy”
Work Package 7: Tool Chain

A SysML Test Model and Test Suite for the ETCS Ceiling Speed Monitor

Technical report

Cécile Braunstein, Wen-ling Huang, Felix Hübner, Jan Peleska and Uwe Schulze

2015-10-09



Funded by:

Federal Ministry of Education and Research
Région de Bruxelles-Capitale
GOBIERNO DE ESPAÑA MINISTERIO DE INDUSTRIA, ENERGÍA Y TURISMO

This page is intentionally left blank

Work Package 4: "Verification & Validation Strategy" **OETCS/WP4/CSM – 01/00**
Work Package 7: Tool Chain **2015-10-09**

A SysML Test Model and Test Suite for the ETCS Ceiling Speed Monitor

Technical report

Cécile Braunstein, Wen-ling Huang, Felix Hübner, Jan Peleska and Uwe Schulze
University Bremen

Technical Report

Prepared for openETCS@ITEA2 Project

Abstract: This technical report contributes to the openETCS project, work packages WP4 – Verification & Validation Strategy , and WP7 – Tool Chain. An analysis of the existing ETCS SUBSET-076 system tests for the ETCS onboard ceiling speed monitor is presented. Alternative algorithms with higher test strength are investigated. The underlying testing technology has been implemented in the RT-Tester model-based testing tool which has been made available for the openETCS tool chain as part of WP7.

Apart from functional and non-functional requirements specifications, the published standard of the European Train Control System (ECTS) also contains a collection of system test suites. One of these suites is aimed at testing the Ceiling Speed Monitor (CSM) which is a function of the European Vital Computer (EVC), the main onboard controller of ETCS trains. In this report we present a detailed comparison of the CSM test suite specified in the ETCS standard with tests generated from a CSM test model, using several automated generation strategies. The test strength of the suites is evaluated using mutations of a CSM software implementation. It turn out that the test suite specified in the ETCS standard is significantly weaker than any of the suites generated with the model-based approach. The greatest test strength is provided by an equivalence partition testing strategy which has been previously elaborated by the authors. The CSM test model has been elaborated by the authors from the ETCS system requirements. Both the model and the model-based test suites have been made publicly available.

Keywords Model-based testing, Equivalence class partition testing, UML/SysML, European Train Control System ETCS, Ceiling Speed Monitoring

1 Introduction

1.1 Background

Model-based testing (MBT) has received much attention over the last years [28, 2]. In particular, we advocate the application of the MBT approach for verification and validation (V&V) of safety-critical control systems. State-of-the-art MBT technology allows for a high degree of automation (typically aided by the use of automated constraint solvers), so that testing experts are relieved of the burden of manual test data generation and test procedure development. As a consequence, more tests can be executed with acceptable effort than would be possible in a scenario where test case and test data generation, as well as test procedure programming still has to be elaborated manually. Moreover, automated MBT allows for the application of more complex test case generation strategies, so that not only the quantity, but also the quality of test cases can be improved. These benefits make up for the additional effort of having to develop a test model as the key enabler for a highly automated testing process. Since the availability of test models also facilitates the compilation of traceability data – this aspect will be described in more detail in the subsequent sections of this article – V&V activities for safety-critical systems become considerably more efficient, while simultaneously increasing the trustworthiness of V&V results through more comprehensive test suites with higher test strength.

Among the more complex test strategies, those possessing *completeness* properties are of special interest. Completeness is defined as *soundness* – a correct system under test (SUT) never fails a test of a suite that is sound – in combination with *exhaustiveness* – every erroneous SUT fails at least one test case of an exhaustive suite [31]. In other words, complete testing strategies are able to *prove* the correctness of implementations. In the context of black-box testing, exhaustiveness cannot be unconditionally guaranteed by any test suite, because the SUT may contain internal state variables (so-called "time bombs") whose changing values trigger erroneous behaviour after a certain number of processing steps, and this internal state information is not observable during black-box testing. Therefore completeness of black-box test suites can only be guaranteed in relation to a *fault model* $\mathcal{F} = (\mathcal{S}, \preceq, \mathcal{D})$ [27]. Fault models consist of a reference model $\mathcal{S}$ specifying the desired behaviour of implementations and a conformance relation $\preceq$ representing the decision criterion for acceptable SUT behaviour: if an implementation behaves like another model $\mathcal{S}'$, its behaviour is accepted if and only if $\mathcal{S}' \preceq \mathcal{S}$. Finally, the fault domain $\mathcal{D}$ specifies a collection of models $\mathcal{S}' \in \mathcal{D}$ that may or may not conform to the reference model $\mathcal{S}$. Test suites that are exhaustive with respect to $\mathcal{F}$ guarantee that every conformance violation $\mathcal{S}' \not\preceq \mathcal{S}$ will be detected by at least one test case, as long as $\mathcal{S}'$ is a member of the fault domain $\mathcal{D}$.

1.2 Objectives

Due to their ability to provide conformance proofs, complete test suites are of considerable theoretical value. Their practical application, however, raises two critical questions. (Q1) How shall we cope with situations where the test suites generated by complete strategies are too large to be executed with acceptable effort? (Q2) In practical V&V campaigns it is hard to determine whether the true behaviour of an SUT is reflected by a model $\mathcal{S}'$ which is still inside the fault domain. For $\mathcal{S}' \notin \mathcal{D}$, exhaustiveness of the suite can no longer be guaranteed. Therefore it remains to be investigated whether simpler model-coverage strategies (these are also described in the next sections) perform equally well or even better for $\mathcal{S}' \notin \mathcal{D}$, while resulting in smaller test suites. In the light of these considerations, the main objective of this article is twofold.

1. Perform an in-depth evaluation of MBT strategies, with special emphasis on the test strength of model-coverage strategies considered already as "standard" in the MBT community, in

comparison to a complete equivalence class partition testing strategy developed by three of the authors [14, 15].

2. Apply the insight gained in 1 to assess the existing test suite specified for the European Train Control System (ETCS) in the standard [10], which has been manually designed by domain experts. As a working example, the ETCS *Ceiling Speed Monitor (CSM)* is used. Its functionality is specified in the ETCS system specification [9].

Question (Q1) stated above is addressed in this article by applying a complete strategy that allows for an abstraction of the state space by means of state equivalence class partitions and input equivalence classes. The CSM has inputs like current speed and applicable speed restrictions whose domains are too large (even if discretisation techniques are applied) to be enumerated explicitly in test suites. We show that for control systems of similar complexity like the CSM, test suites of manageable size are generated. Question (Q2) is addressed by showing that a randomisation of the complete strategy which preserves the exhaustiveness property for SUT behaviours inside the fault domain also results in superior test strength for the cases where $\mathcal{S}' \notin \mathcal{D}$.

1.3 Main Contributions

The work presented in this article is based on the previous publication [6]. There, an initial version of the CSM test model has been published, and the application of model-based testing has been exemplified in a qualitative way, that is, for a small set of hand-crafted examples that were suitable to highlight the effects of each individual test strategy on specific classes of errors. The tests from the standard [10] had not yet been considered. The novel contributions of the present article are characterised as follows.

- The initial CSM test model version has been slightly updated to improve its readability. In contrast to the qualitative assessment in [6], however, we perform a detailed *quantitative* test strength evaluation for the test strategies involved. This is achieved by using automated mutant generators on a reference implementation of the CSM programmed in Java.
- Moreover, we use an extended version of the equivalence testing strategy applied in [6]: as shown in [14], that strategy is complete with respect to a given fault model. It has been demonstrated in [15] that this complete strategy can be modified to obtain improved test strength when applied against SUTs whose true behaviour is reflected by models *outside* the fault domain. This is achieved by performing random selections from each input equivalence class, instead of testing with one representative per class only. This modified strategy is applied for the results presented in this article.
- Finally, this article presents a detailed comparison of the model-based test strategies with the manually designed CSM test suite published in the ETCS standard. It turns out that the model-based test strategies have considerably higher test strength than the tests from the standard. This understandable, because the test suites from the standard do not claim completeness, but aim at checking the basic conformity of EVC implementations to the standard. We can, however, still suggest to extend the current baseline of the ETCS test suites in a way that will improve the trustworthiness of conformity verdicts, without unduly increasing the number of test cases.

In 2011 the *model-based testing benchmarks website* www.mbt-benchmarks.org has been created. Its objective is to publish test models that may serve as challenges or benchmarks for validating

testing theories and for comparing the capabilities of MBT tools [24]. The results presented in this article are publicly available on this website. The material published there comprises

- a SysML model of the CSM,
- the model-based test suites generated according to the various MBT strategies,
- a formalisation of the relevant ETCS test cases from the standard [10] as formulas specified in the temporal logic LTL,
- a CSM reference implementation in Java, and
- configuration parameters that were used for the mutation generator in order to create faulty test candidates.

All automated test generation strategies described here have been implemented in the model-based test automation tool RT-Tester [23] which applies the SONOLAR SMT solver[1] for constraint solving and test data calculation. RT-Tester licences are freely available for academic research.

1.4 Overview

In Section 2 the SysML model for the ceiling speed monitor is described, and we briefly summarise its behavioural semantics. With a test model at hand, two complementary approaches to test case generation are available: (1) test cases can be derived from requirements which can be traced back to the model, and (2) test cases can be derived from model coverage goals. The underlying methods for these alternative approaches are described in Section 3, and test case examples for the CSM are presented. In Section 4 we describe the complete equivalence class partition testing method in more detail, together with its randomisation that aims at higher test strength for SUT outside the fault domain. For formal proofs of the strategy's properties we refer to previous publications. We present the experimental results in Section 5, where the test suites created by the different approaches described before are evaluated and compared. In Section 6 related work is described, and Section 7 presents the conclusions and ongoing work related to the results presented here.

2 CSM Model Description

2.1 Functional Objectives

The European Train Control System ETCS relies on the existence of an onboard controller in train engines, the *European Vital Computer EVC*. Its functionality and basic architectural features are described in the public ETCS system specification [11]. One functional category of the EVC covers aspects of speed and distance monitoring, to accomplish the *"... supervision of the speed of the train versus its position, in order to assure that the train remains within the given speed and distance limits."* [9, 3.13.1.1]. While displaying actual and allowed speed to the train engine driver, the monitoring functions automatically trigger the brakes in case of speed limit violations. Speed and distance monitoring is decomposed into three sub-functions [9, 3.13.10.1.2], where only one out of these three is active at a point in time: (1) *Ceiling speed monitoring (CSM)* supervises the observance of the maximal speed allowed according to the current most restrictive speed profile (MRSP)[2]. CSM is active while the train does not approach a target (train station,

[1] http://smtcomp.sourceforge.net/2012/reports/sonolar.pdf

[2] In some situations, more than one speed restriction may apply, and then the most restrictive one has to be enforced.

level crossing, or any other point that must be reached with predefined speed). (2) *Target speed monitoring (TSM)* enforces speed restrictions applicable while the train brakes to a target, for example, a track section where a significantly lower maximal speed has to be observed. (3) *Release speed monitoring (RSM)* applies when the special target "end of movement authority (EOA)" is approached, where the train must come to a stop. RSM supervises the observance of the distance-depending so-called release speed, when the train approaches the EOA and is allowed to drive at a reduced speed.

The model presented here captures the CSM functionality.

2.2 System Requirements

The ETCS system specification [9] defines the 11 following requirements for the CSM (see Table 1). For traceability purpose, The requirement identifiers refer to the sections of the ETCS specification [9]. The requirements are decomposed into two parts: the general requirements that concern all the three supervision functions (including the CSM) and the ones that only concern the CSM function.

Table 1. Requirements for the ceiling speed monitoring function (quoted verbatim from [9, 3.13.10]).

id	Description
	General Requirements
REQ-3.13.10.2.1	The train speed indicated to the driver shall be identical to the speed used for the speed monitoring. This shall be the estimated speed.
REQ-3.13.10.2.2	Once a Train Interface command (traction cut-off, service brake or emergency brake) is triggered, the on-board shall apply it until its corresponding revocation condition is met.
REQ-3.13.10.2.3	If there is no on-board interface with the service brake or if the use of the service brake command is not allowed by a National Value (only in Target speed monitoring),whenever a service brake command is specified, the emergency brake command shall be triggered instead.
REQ-3.13.10.2.4	The emergency brake command, which is triggered instead of the service brake command when an SBI supervision limit is exceeded, shall be revoked according to the requirements specified for the revocation of service brake command, unless the emergency brake command has been also triggered due to an EBI supervision limit. In such case, the condition for revoking the emergency brake command due to EBI supervision limit shall prevail.
REQ-3.13.10.2.5	The on-board shall revoke the Intervention status only when no brake command is applied by the speed and distance monitoring function.
	Requirements for CSM
REQ-3.13.10.3.1	The on-board equipment shall display the permitted speed.
REQ-3.13.10.3.2	When the supervision status is Overspeed, Warning or Intervention, the on-board equipment shall display the SBI speed.
REQ-3.13.10.3.3	The on-board shall compare the estimated speed with the ceiling supervision limits defined in [9, 3.13.9.2] and shall trigger/revoke commands to the train interface (service brake if implemented or emergency brake) and supervision statuses as described in Table 2 (from [9, Table 5]) and Table 3 (from [9, Table 6]).
REQ-3.13.10.3.4	The on-board equipment shall execute the transitions between the different supervision statuses as described in Table 4 (see [9, 4.6.1] for details about the symbols). This table takes into account the order of precedence between the supervision statuses and the possible updates of the MRSP while in ceiling speed monitoring (e.g. when a TSR is revoked).
REQ-3.13.10.3.5	When the speed and distance monitoring function becomes active and the ceiling speed monitoring is the first one entered, the triggering condition t1 defined in Table 2 shall be checked in order to determine whether the Normal status applies. If it is not the case, the on-board shall immediately set the supervision status to the relevant value, applying a transition from the Normal status according to Table 4.
REQ-3.13.10.3.6	The Indication status is not used in ceiling speed monitoring. However, in case the ceiling speed monitoring is entered and the supervision status was previously set to Indication, the on-board equipment shall immediately execute one of the transitions from the Indication status, as described in Table 4.

Table 2. Triggering of Train Interface commands and supervision statuses in ceiling speed monitoring (from [9, Table 5]).

id	TC	Estimated speed	TI	SSE
REQ-3.13.10.3.3.t1	t1	$V_{est} \leq V_{MRSP}$	—	Normal Status
REQ-3.13.10.3.3.t2	t2	$V_{est} > V_{MRSP}$	—	Overspeed Status
REQ-3.13.10.3.3.t3	t3	$V_{est} > V_{MRSP} + \mathrm{dV}_{\mathrm{warning}}$	—	Warning Status
REQ-3.13.10.3.3.t4	t4	$V_{est} > V_{MRSP} + \mathrm{dV}_{\mathrm{sbi}}$	SB	Intervention Status
REQ-3.13.10.3.3.t5	t5	$V_{est} > V_{MRSP} + \mathrm{dV}_{\mathrm{ebi}}$	EB	Intervention Status

TC: trigger condition
TI: command triggered on train interface to brakes
SB: trigger service brake command (if available, otherwise trigger emergency brake)
EB: trigger emergency brake command SSE: supervision status entered

Table 3. Revocation of Train Interface commands and supervision statuses in ceiling speed monitoring (from [9, Table 6]).

id	RC	Estimated Speed	TICR	SSR
REQ-3.13.10.3.3.r0	r0	Standstill	EB	Intervention Status
REQ-3.13.10.3.3.r1	r1	$V_{est} \leq V_{MRSP}$	SB	Indication Status
			EB*	Overspeed Status
				Warning Status
				Intervention Status (if SBI)
				Intervention Status (if EB*)

* Only if allowRevokeEB = 1.

RC: revocation condition
TICR: command revoked on train interface to brakes
SSR: supervision status revoked

Table 4. Transitions between supervision statuses in ceiling speed monitoring (from [9, Table 7]).

Normal	$< r1$	$< r1$	$< r1$	$< r0, r1$
	Indication			
$t2 >$	$t2 >$	Overspeed		
$t3 >$	$t3 >$	$t3 >$	Warning	
$t4, t5 >$	$t4, t5 >$	$t4, t5 >$	$t4, t5 >$	Intervention

The sub-requirements IDs associated with each cell in the transition table are of the form REQ-3.13.10.3.4.rXcY where X and Y are the row and column indexes, respectively.

2.3 Test Model Semantics

SysML test models are structured using blocks. At the top-level, the model is decomposed into a block representing the system under test (SUT) and another one representing the test environment (TE); Fig. 1 shows this decomposition for the CSM. Depending on the complexity of the model, blocks can be further decomposed into lower-level block diagrams, until leaf blocks are reached that are associated with behaviour. In our test models this behaviour is specified by sequential hierarchic SysML state machines. Blocks execute concurrently and in a synchronous way, so that transitions of concurrent state machines that are enabled in the same model state execute simultaneously.

The whole model executes according to the *run-to-completion* semantics defined for state machines. The model is in a *quiescent* (or stable) state, if no transition can be executed without an input change or time passing.

In a quiescent model state, inputs may be changed. If these changes enable a transition, the latter is executed. Our SUT model is deterministic, this is typical for sequential safety-critical applications, but note that possible occurrences of non-deterministic models in a safety-critical context are discussed in Section 7. Due to determinism, there is no necessity to handle situations where several transitions are simultaneously enabled. The executed transition, however, may lead to a *transient* state, that is, to a state where another transition is enabled. In the run-to-completion semantics this new transition is also executed, and so forth until a quiescent state is reached. Conceptually, the consecutive execution of model transitions is executed in zero time, so that input changes cannot happen until the next quiescent state has been reached. Moreover, models admitting unbounded sequences of transitions between transient states are considered as illegal, and this situation is called a *livelock* failure.

2.4 Interfaces

The interfaces between SUT and its environment are specified in the internal block diagram displayed in Fig. 1. All interfaces are represented as flow ports. The environment writes to SUT input ports and reads from SUT output ports.

Ceiling speed monitoring is activated and de-activated by the speed and distance monitoring (SnD) coordination function that controls CSM, TSM, and RSM: on input interface ControlSUTCommand, variable csmSwitch specifies whether ceiling speed monitoring should be active (csmSwitch = startSUT) or passive, since target or release speed monitoring is being performed (csmSwitch = stopSUT).

The interface TrainInputs carries variable SBAvailable which has value 1, if the train is equipped with a service brake. This brake is then used for slowing down the train if it has exceeded the maximal speed allowed, but not yet reached the threshold for an emergency brake intervention. If SBAvailable = 0, the emergency brake shall be used for slowing down the train in this situation. Input SBAvailable is to be considered as a configuration parameter of the train, since it depends on the availability of the service brake hardware. Therefore this value can be freely selected at start-of-test, but must remain constant during test execution.

Input interface OdometrySerialInterface provides the current speed value estimated by the odometer equipment in variable V_{est}. Input interface SupervisionLimits provides the current maximal velocity defined by the most restrictive speed profile in variable V_{MRSP}. Input interface TracksideInput provides a control flag for the ceiling speed monitor: variable allowRevokeEB is

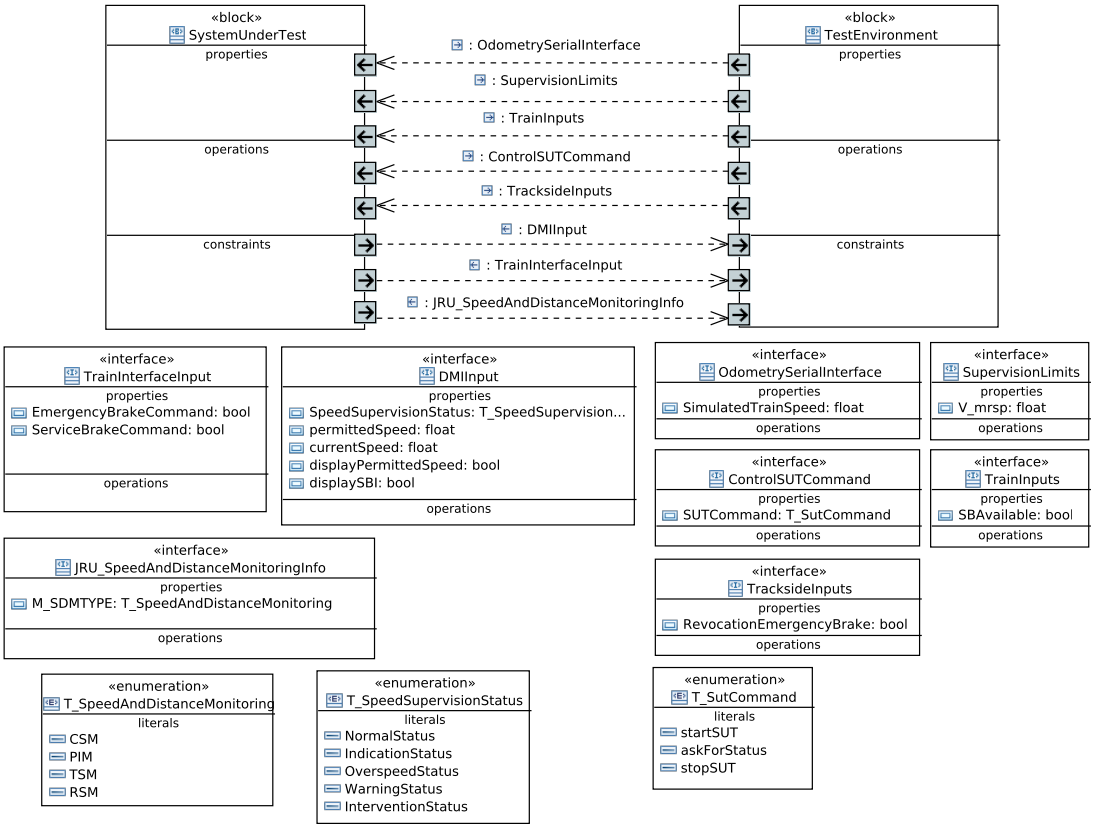


Figure 1. System interface of the ceiling speed monitor.

1, if after an emergency brake intervention the brake may be automatically released as soon as the estimated velocity of the train is again less or equal to the maximal speed allowed. Otherwise (allowRevokeEB = 0) the emergency brakes must only be released after the train has come to a standstill ($V_{est} = 0$). This input parameter is a part of the "national values" given by the tracks, it may change when a train crosses the boundaries between European countries, due to their local regulations.

Output interface DMIInput sends data from the SUT to the driver machine interface (DMI). It carries five variables. DMICmd is used to display the supervision status to the train engine driver: Value IndicationStatus may be initially present when CSM is activated, but will be immediately overridden by one of the values NormalStatus, OverspeedStatus, WarningStatus, or InterventionStatus, as soon as ceiling speed monitoring becomes active. Value NORMAL is written by the SUT to this variable as long as the ceiling speed is not violated by the current estimated speed. Value OverspeedStatus has to be set by the CSM as soon as condition $V_{MRSP} < V_{est}$ becomes true. If the speed increases further (the detailed conditions are described below), the indication changes from OverspeedStatus to WarningStatus, and from there to InterventionStatus. The latter value indicates that either the train is slowed down until it is back in the normal speed range, or the emergency brake has been triggered to stop the train. Furthermore, interface DMIInput contains several speed-related variables that are displayed on the DMI.

Output interface TrainInterfaceInput specifies the train interface from the CSM to the brakes, using variables EmergencyBrakeCommand and ServiceBrakeCommand. If both equals 0, both service brakes (if existent) and emergency brakes are released. If ServiceBrakeCommand is acti-

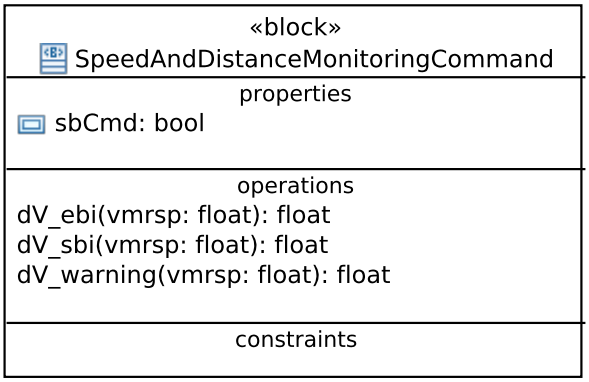


Figure 2. Block diagram with CSM (sequential behaviour).

vated and EmergencyBrakeCommand is not, the service brake is activated. If EmergencyBrakeCommand is activated, the emergency brake is triggered.

Output interface JRU_SpeedAndDistanceMoinitoringInfo carries the variable M_SDMTYPE that indicates which sub-function of the speed and distance monitoring is active. The Juridical Recording Unit (JRU) is used to record information during the test phase, we have used this interface as well as the DMI interface in compliance with the testing methodology defined by the European railway agency in [10] (see section 3.2).

2.5 SUT Attributes and Operations

The CSM executes sequentially; therefore the SUT block on the top-level interface diagram (Fig. 1) is refined to a single block representing the CSM, as shown in Fig. 2. There, the SUT uses a local attribute sbiCmd which carries value 1, if the service brake should be used for slowing down the train to the admissible speed. If the value 0 is assigned to sbiCmd, the emergency brake will be triggered in this situation.

Three supervision limits are computed to assist the driver in preventing automated service or emergency brake intervention by maintaining the speed within certain limits, namely Warning, Service brake intervention (sbi) and emergency brake intervention (ebi) limits. These limits depend on the MRSP, and they are calculated according to [9] as follows.

$$\mathrm{dV}_{\mathrm{warning}}(V_{MRSP}) = \begin{cases} \min\{\frac{1}{3} + \frac{1}{30} \cdot V_{MRSP}, 5\} & \text{if } V_{MRSP} > 110 \\ 4 & \text{if } V_{MRSP} \leq 110 \end{cases} \quad (1)$$

$$\mathrm{dV}_{\mathrm{sbi}}(V_{MRSP}) = \begin{cases} \min\{0.55 + 0.045 \cdot V_{MRSP}, 10\} & \text{if } V_{MRSP} > 110 \\ 5.5 & \text{if } V_{MRSP} \leq 110 \end{cases} \quad (2)$$

$$\mathrm{dV}_{\mathrm{ebi}}(V_{MRSP}) = \begin{cases} \min\{-0.75 + 0.075 \cdot V_{MRSP}, 15\} & \text{if } V_{MRSP} > 110 \\ 7.5 & \text{if } V_{MRSP} \leq 110 \end{cases} \quad (3)$$

2.6 CSM Behavioural Specification

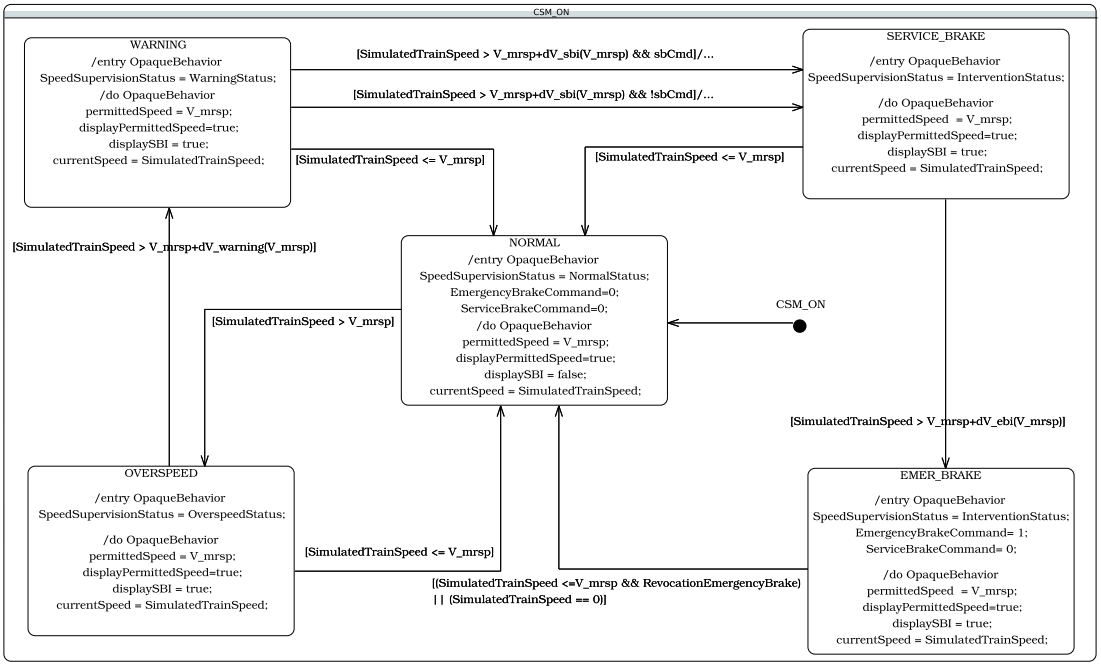


Figure 3. Ceiling speed monitoring state machine.

The behaviour of the ceiling speed monitor is modeled by a hierarchic state machine that is associated with the SUT block of Fig. 2. The top-level machine (not shown here) specifies the activation and de-activation of the CSM during the interplay between CSM, TSM, and RSM: state machine CSM_ON (this is the machine describing the behaviour of the CSM) is entered as soon as the value of input variable csmSwitch changes to startSUT, and it is left when its value changes to stopSUT. The lower-level state machine CSM_ON models the behaviour of the active CSM, as displayed in Fig. 3.

Its execution starts in basic state NORMAL, where the NormalStatus indication is displayed on the DMI and brakes are released (EmergencyBrakeCommand = 0 and ServiceBrakeCommand = 0). When the speed increases above the maximal speed allowed ($V_{est} > V_{MRSP}$), the state machine transits to basic state OVERSPEED, where the OverspeedStatus indication is displayed to the train engine driver. If the train continues over speeding until the warning threshold V_{MRSP} + $\mathrm{dV}_{\mathsf{warning}}(V_{MRSP})$ is exceeded, a transition into the WARNING state is performed, accompanied by an indication change on the DMI. Accelerating further until $V_{est} > V_{MRSP} + \mathrm{dV}_{\mathsf{sbi}}(V_{MRSP})$ leads to a transition into basic state SERVICE_BRAKE, where either the service brake or the emergency brake is triggered, depending on the value stored before in variable sbiCmd. The DMI display changes to InterventionStatus.

The intervention status is realized by two basic state machine states, SERVICE_BRAKE and EMER_BRAKE. From SERVICE_BRAKE it is still possible to return to NORMAL, as soon as the speed has been decreased below the over speeding threshold. When the train, however, continues its acceleration until the emergency braking threshold has been exceeded ($V_{est} > V_{MRSP} + \mathrm{dV}_{\mathsf{ebi}}(V_{MRSP})$), basic state EMER_BRAKE is entered. From there, a state machine transition to NORMAL is only possible if the train comes to a standstill, or if the national regulations (variable allowRevokeEB) allow to release the brakes as soon as over speeding has stopped.

Observe that the run-to-completion semantics of state machines also allows for zero-time transitions from, for example, NORMAL to EMER_BRAKE. If, while in basic state NORMAL, the

inputs change such that $V_{est} > V_{MRSP} + \mathrm{dV}_{\mathsf{ebi}}(V_{MRSP})$ becomes true[3], the state machine transition from NORMAL to OVERSPEED leads to a transient model state, because guard condition $V_{est} > V_{MRSP} + \mathrm{dV}_{\mathsf{warning}}(V_{MRSP})$ is already fulfilled, and the state machine transits to WARNING. Similarly, guards $V_{est} > V_{MRSP} + \mathrm{dV}_{\mathsf{sbi}}(V_{MRSP})$ and $V_{est} > V_{MRSP} + \mathrm{dV}_{\mathsf{ebi}}(V_{MRSP})$ also evaluate to true, so that the next quiescent state is reached in basic state EMER_BRAKE.

2.7 Tracing Requirements to the Model

SysML provides language elements for relating model elements to requirements, using the «satisfy» relationship from model elements to requirements symbols in arbitrary SysML diagrams [21, Section 16]. Alternatively, SysML models may be augmented by tables associating requirements and model elements. Exploiting this language feature supports both model validation and requirements-based testing. In the former case, missing requirements can be detected if they cannot be linked to structural or behavioural model elements in the appropriate way. In latter case, execution traces through the model covering a given structural or behavioural model element represent test cases contributing to the verification of all requirements related to the model element under consideration. For the CSM model described above, the system requirements specified in Section 2.2 can be traced to model elements as follows.

Tables 5 associates the SysML elements with the requirements they satisfiy. “Submachine State” and “Atomic State” refer to the top-level states of composite machines and to basic control states, respectively.

The complexity of «satisfy» relations between structural or behavioural model elements depends on the complexity of the requirement and the way it is reflected by the structural and behavioural model. Consider, for example (see Table 1), requirement

- REQ-3.13.10.2.1: The train speed indicated to the driver shall be identical to the speed used for the speed monitoring (i.e. the estimated speed V_{est}).

Every model trace where the CSM is activated is suitable for verifying this requirement, because the DMI variable speedToDriver is updated by the actual speed V_{est} via operation calc_speed_to_driver(), whenever the ceiling speed monitor is active, that is, in composite state CSM_ON. Therefore CSM_ON is linked to REQ-3.13.10.2.1 by the «satisfy» relation, as expressed in Table 5, row 1.

A more complex case of requirements tracing presents itself if one or more transitions are related to a given requirement. This is the case for the requirement

- REQ-3.13.10.2.2: Once a Train Interface command (traction cut-off, service brake or emergency brake) is triggered, the on-board shall apply it until its corresponding revocation condition is met.

As modeled in Fig. 3, we have two revocation conditions; one is reflected by the transition from basic state SERVICE_BRAKE to NORMAL, the other from EMER_BRAKE to NORMAL. Therefore both transitions are related to REQ-3.13.10.2.2, as specified in row 2 of Table 5. In the most complex case we have to handle situations where requirements are reflected by traces

[3] This would be an exceptional behaviour situation, caused, for example, by temporary unavailability of odometry data, so that a “sudden jump” of V_{est} would be observed by the CSM.

Table 5. Requirements links to the SysML Elements

No.	Requirement	⟵ «satisfy»
1	REQ-3.13.10.2.1	«Composite State» CSM_ON
2	REQ-3.13.10.2.2	«Transition» [EMER_BRAKE - NORMAL]
		«Transition» [SERVICE_BRAKE - NORMAL]
3	REQ-3.13.10.2.3	«Transition» [WARNING - SERVICE_BRAKE] (!sbCmd)
		«Basic State» SERVICE_BRAKE
4	REQ-3.13.10.2.4	«Transition» [EMER_BRAKE - NORMAL]
		«Transition» [SERVICE_BRAKE - NORMAL]
		«Transition» [WARNING - SERVICE_BRAKE] (!sbCmd)
5	REQ-3.13.10.2.5	«Transition» [EMER_BRAKE - NORMAL]
		«Transition» [SERVICE_BRAKE - NORMAL]
6	REQ-3.13.10.3.1	«Submachine State» CSM_ON
7	REQ-3.13.10.3.2	«Basic State» OVERSPEED
		«Basic State» SERVICE_BRAKE
		«Basic State» WARNING
		«Basic State» EMER_BRAKE
8	REQ-3.13.10.3.3 .r0	«Transition» [EMER_BRAKE - NORMAL]
	.r1	«Transition» [OVERSPEED - NORMAL]
		«Transition» [SERVICE_BRAKE - NORMAL]
		«Transition» [WARNING - NORMAL]
		«Transition» [EMER_BRAKE - NORMAL]
	.t1	«Basic State» NORMAL
	.t2	«Basic State» OVERSPEED
	.t3	«Basic State» WARNING
	.t4	«Basic State» SERVICE_BRAKE
	.t5	«Basic State» EMER_BRAKE
9	REQ-3.13.10.3.4 .r1c2	«Constraint» constraint_14
	.r1c3	«Transition» [OVERSPEED - NORMAL]
	.r1c4	«Transition» [WARNING - NORMAL]
	.r1c5	«Transition» [EMER_BRAKE - NORMAL]
		«Transition» [SERVICE_BRAKE - NORMAL]
	.r3c1	«Transition» [NORMAL - OVERSPEED]
	.r3c2	«Constraint» constraint_15
	.r4c1	«Constraint» constraint_08
	.r4c2	«Constraint» constraint_16
	.r4c3	«Transition» [OVERSPEED - WARNING]
	.r5c1	«Constraint» constraint_10
		«Constraint» constraint_09
	.r5c2	«Constraint» constraint_17
	.r5c3	«Constraint» constraint_12
		«Constraint» constraint_11
	.r5c4	«Transition» [WARNING - SERVICE_BRAKE]
		«Transition» [SERVICE_BRAKE - EMER_BRAKE]
		«Constraint» constraint_13
10	REQ-3.13.10.3.5	«Constraint» constraint_05
		«Constraint» constraint_06
		«Constraint» constraint_07
		«Basic State» NORMAL
		«Constraint» constraint_04
11	REQ-3.13.10.3.6	«Constraint» constraint_05
		«Constraint» constraint_06
		«Constraint» constraint_07
		«Constraint» constraint_04

visiting model state *vectors*[4] fulfilling certain constraints, and these model state vectors have to be visited by the traces in a specific order. Such a situation is reflected, for example, by

- REQ-3.13.10.3.4: The on-board equipment shall execute the transitions between the different supervision statuses as described in Table 4 (see [9, 4.6.1] for details about the symbols). This table takes into account the order of precedence between the supervision statuses and the possible updates of the MRSP while in ceiling speed monitoring (e.g. when a TSR is revoked; TSR = Temporary Speed Restriction).

This requirement may be decomposed into atomic sub-requirements one for each pertinent cells of table 3. Some of these sub-requirements are again reflected by transitions. Sub-requirement REQ-3.13.10.3.4.r5c1, however, specifies the possibility to directly transit from NORMAL to SERVICE_BRAKE or EMER_BRAKE. This cannot be specified by simply linking a behavioural model element to the sub-requirement, because we have avoided to draw direct state machine transitions from NORMAL to SERVICE_BRAKE or EMER_BRAKE, since those transitions are implicitly realized by the run-to-completion semantics, as explained above. In the formalization of requirements tracing described in Section 3.1 we show how these situations can be covered by using traceability specifications by means of temporal logic formulas.

3 Requirements Driven Approach

3.1 Formal Requirements Tracing in SysML Models

Consider, for example, the zero-time transition NORMAL $\longrightarrow$ SERVICE_BRAKE. To cover this situation, we need to

1. Enter NORMAL in a quiescent model state – this is specified by
$$[\mathsf{NORMAL} \wedge V_{est} \leq V_{MRSP}]$$
2. Stay there until the speed exceeds $V_{MRSP} + \mathrm{dV}_{\mathsf{sbi}}(V_{MRSP})$ but remains less or equal to $V_{MRSP} + \mathrm{dV}_{\mathsf{ebi}}$.

Formally this is expressed in LTL by

$$\begin{aligned}\mathsf{Finally}([&\mathsf{NORMAL} \wedge V_{est} \leq V_{MRSP}] \wedge \\ &([\mathsf{NORMAL} \wedge V_{est} \leq V_{MRSP}] \\ &\ \mathsf{Until} \\ &[\mathsf{NORMAL} \wedge V_{est} > V_{MRSP} + \mathrm{dV}_{\mathsf{sbi}}(V_{MRSP}) \wedge \\ &V_{est} \leq V_{MRSP} + \mathrm{dV}_{\mathsf{ebi}}(V_{MRSP})]))\end{aligned}$$

This is expressed by constraint_09 linked to REQ-3.13.10.3.4.r5c1 in Table 5. Similarly, covering the zero-time transition NORMAL $\longrightarrow$ EMER_BRAKE requires a trace satisfying

$$\begin{aligned}\mathsf{Finally}([&\mathsf{NORMAL} \wedge V_{est} \leq V_{MRSP}] \wedge \\ &([\mathsf{NORMAL} \wedge V_{est} \leq V_{MRSP}] \\ &\ \mathsf{Until} \\ &[\mathsf{NORMAL} \wedge V_{est} > V_{MRSP} + \mathrm{dV}_{\mathsf{ebi}}(V_{MRSP})]))\end{aligned}$$

[4] A model state vector consists of valuations of inputs, outputs, and internal model variables, as well as of variable valuations indicating the basic state machine states currently active.

Similar constraints are specified for REQ-3.13.10.3.4.r1c2, REQ-3.13.10.3.4.r3c2, REQ-3.13.10.3.4.r4c1, REQ-3.13.10.3.4.r4c2, REQ-3.13.10.3.4.r5c2, REQ-3.13.10.3.4.r5c3 and REQ-3.13.10.3.4.r5c4 as well as for requirements REQ-3.13.10.3.5 and REQ-3.13.10.3.6.

3.2 Manually Defined Tests

The European Railway agency provides a test specification [10] along with the requirements of the EVC. The tests aim at verifying the conformity and the functionality of the onboard subsystems against the system requirement specification (SRS) [11]. The SRS has been decomposed into a set of features such that a feature groups a set of requirements that can be tested at the available interfaces. The test cases have been designed to ensure that a test of a given feature is independent of all other feature. Moreover tests are only described as a reaction to a given stimulation at the interface of the feature.

The test specification formalises the test cases description, each test case is composed of

- a unique identifier,
- a short description of the target of the test,
- a list of covered requirement,
- an initial state of the test e.g. initial assignment to the internal test variables,
- the test steps: inputs change and expected outputs, and
- a final test state e.g. initial assignment to the internal test variables.

Table 6 summarizes the available tests. The second column describes the objective of each test. The column “Covers Requirements” shows the list of requirements covered by the tests. We have refined the list provided by the standard. We first refer to the sub-requirement when needed and secondly, we add some requirements that are covered by side effect to be able to provide more accurate requirement coverage.

The ceiling speed monitoring feature is associated with 8 test cases in [10]. They cover the change of the speed supervision status and the brake commands depending on the train speed. We only use seven of them (TC-CSM-[1-7]), the one missing covers a requirement that we assume, is “delegated” to the surrounding software of the CSM. Eight test cases cover the general requirement of the speed and distance monitoring feature in [10]. Four of those are test cases outside the scope of the ceiling speed monitoring and therefore not included in our benchmark. The four others cover the requirements REQ-3.13.10.2.3 and REQ-3.13.10.2.4 but only in the target speed monitoring section. We have adapted them to fit into the collection of CSM tests. Moreover, we grouped them in pairs (TC-GR-[1-2]) . Each pair is dealing with two different ways of receiving inputs depending on the ERTMS mode that is not relevant for testing the CSM function.

These 9 test cases has been translated into LTL formula to fit our experiments platform. This translation is performed by representing the steps by a sequence of nested until operator. Let the test sequence be of the following form: $i_0, i_1, o_0, i_2, o_1, o_2$ where i_k (o_k) represents an input (resp. output) assignment to an interface variables. The associated LTL formula will be in the following form:

$$\mathsf{Finally}([i_0 \wedge i_1 \wedge o_0] \wedge ([i_0 \wedge i_1 \wedge o_0] \ \mathsf{Until}\ [i_2 \wedge \mathsf{Finally}([o_1 \wedge o_2]))$$

Table 6. Test cases from the Subset 076

Identifier	Target of the test	Covered Requirements
TC-CSM-1	When the train runs in CSM and does not exceed the the permitted speed, no intervention is triggered and the NormalStatus is displayed.	REQ-3.13.10.3.1, REQ-3.13.10.3.3, REQ-3.13.10.3.5
TC-CSM-2	When the train runs in CSM and the speed is between the permitted speed and the Warning limit, no intervention is triggered and the OverspeedStatus is displayed. Once the train speed is below the permitted speed the NormalStatus is displayed.	REQ-3.13.10.3.1, REQ-3.13.10.3.2, REQ-3.13.10.3.3, REQ-3.13.10.3.4
TC-CSM-3	When the train runs in CSM and the speed is between the Warning limit speed and the SBI supervision limit, no intervention is triggered and the WarningStatus is displayed. Once the train speed is below the permitted speed the NormalStatus is displayed.	REQ-3.13.10.3.1, REQ-3.13.10.3.2, REQ-3.13.10.3.3, REQ.3.13.10.3.4.r4
TC-CSM-4	When the train runs in CSM and the speed is between the SBI supervision limit and the EBI supervision limit , the service brake intervention is triggered and the InterventionStatus is displayed. Once the train speed is below the permitted speed the NormalStatus is displayed.	REQ-3.13.10.2.2, REQ-3.13.10.3.1, REQ-3.13.10.3.2, REQ-3.13.10.3.3, REQ-3.13.10.3.4
TC-CSM-5	When the train runs in CSM and the speed is greater then the EBI supervision limit , the emergency brake intervention is triggered and the InterventionStatus is displayed. Once the train reaches standstill the NormalStatus is displayed.	REQ-3.13.10.2.2, REQ-3.13.10.2.5, REQ-3.13.10.3.1, REQ-3.13.10.3.2, REQ-3.13.10.3.3, REQ-3.13.10.3.4
TC-CSM-6	When entering CSM mode the Indication status is overwritten	REQ-3.13.10.3.1, REQ-3.13.10.3.3, REQ-3.13.10.3.4, REQ-3.13.10.3.6
TC-CSM-7	When the train is between the permitted speed and the Warning limit the SBI supervision limit also referred as the FLOI (First Line Of Intervention) is displayed. Once the train speed is below the permitted speed the NormalStatus is displayed	REQ-3.13.10.3.1, REQ-3.13.10.3.2, REQ-3.13.10.3.3, REQ-3.13.10.3.4, REQ-3.13.10.3.5
TC-GR-1	When the use of service brake is not allowed the emergency brake command shall be triggered instead. The emergency brake is then revoked according to the service brake revocation.	REQ-3.13.10.2.1, REQ-3.13.10.2.3, REQ-3.13.10.2.4, REQ-3.13.10.3.3, REQ-3.13.10.3.4
TC-GR-2	The use of service brake is not allowed and the train exceeds the EBI supervision limit, the emergency brake command shall be triggered. The emergency brake is then revoked only when the train reaches standstill.	REQ-3.13.10.2.1, REQ-3.13.10.2.3, REQ-3.13.10.2.4, REQ-3.13.10.3.3, REQ-3.13.10.3.4

```
1  TC-CSM-2;
2   Finally ([currentSpeed > 0 &&
3             currentSpeed < V_mrsp &&
4             SpeedSupervisionStatus == NormalStatus &&
5             displayPermittedSpeed == 1 ]
6            &&
7            ([currentSpeed > 0 &&
8              currentSpeed < V_mrsp &&
9              SpeedSupervisionStatus == NormalStatus &&
10             displayPermittedSpeed == 1 ]
11           Until
12           (([currentSpeed > V_mrsp &&
13              currentSpeed <= V_mrsp + dV_Waring(V_mrsp)]  &&
14              Finally[
15                      SpeedSupervisionStatus ==  OverspeedStatus &&
16                      displayPermittedSpeed == 1 &&
17                      displaySBI == 1 &&
18                      ServiceBrakeCommand == 0 ])
19               &&
20               ([currentSpeed > V_mrsp  &&
21                 currentSpeed <= V_mrsp + dV_Waring(V_mrsp) ]
22               Until
23               ([currentSpeed > 0 && currentSpeed < V_mrsp ] &&
24                 Finally[SpeedSupervisionStatus == NormalStatus ])))));
25
```

Figure 4. Example of test case TC-CSM-2 as LTL formula

Intuitively the inputs are set and do not change until a new input configuration is reached and implies new output values.

Figure 4 shows the LTL formula used to represent the test case TC-CSM-2.

- lines 2-5 : The train is moving within the permitted speed and the DMI aspects is coherent with the normal status.
- lines 7-13 : The train stays in the initial configuration until the speed is greater than permitted speed but below the Warning limit.
- lines 14-21 : The DMI information changes according to the new speed.
- line 23 : The train speed is now back below the permitted.
- line 24 : The Normal status is now displayed.

3.3 Extended Manually Defined Tests

The test specification provided by the European Railway agency ([10]) contains test suites for the different ETCS features. The sub-requirements REQ-3.13.10.3.4.r1c2, REQ-3.13.10.3.4.r3c2, REQ-3.13.10.3.4.r4c2 and REQ-3.13.10.3.4.r5c2 describe a transition from supervision status *Indication* to *Normal*, *Overspeed*, *Warning* and *Intervention*. Supervision status *Indication* is not used by CSM, but requirement REQ-3.13.10.3.6 states, that if CSM is ativated with supervision

Table 7. Additional test cases to cover the remaining sub-requirements.

Identifier	Target of the test	Covered Requirements
TC-CSM-8	When entering CSM mode the Indication status is overwritten	REQ-3.13.10.3.4.r1c2, REQ-3.13.10.3.4.r3c2, REQ-3.13.10.3.4.r4c2, REQ-3.13.10.3.4.r5c2, REQ-3.13.10.3.6
TC-CSM-9	Switching from *Overspeed* to *Intervention* and *Warning* to *Intervention*	REQ-3.13.10.3.4.r5c3, REQ-3.13.10.3.4.r5c4

status being *Indication*, the state changes shall occur as defined in the sub-requirements above. The manual test case for REQ-3.13.10.3.6 defined in [10] only covers the state transition from *Indication* to *Normal*. Therefore the test manual test suite does not cover the sub-requirements REQ-3.13.10.3.4.r3c2, REQ-3.13.10.3.4.r4c2 and REQ-3.13.10.3.4.r5c2.

Our analysis shows that the sub-requirements REQ-3.13.10.3.4.r5c3 and REQ-3.13.10.3.4.r5c4 are not covered by the tests defined in [10]. These two sub-requirements define the behaviour of CSM with supervision status being *Overspeed* and the trigger conditions require to directly change to *Intervention* (REQ-3.13.10.3.4.r5c3) and supervision status being *Warning* and the trigger conditions require to change to *Intervention* (REQ-3.13.10.3.4.r5c4).

To be able to compare the test manually defined suite with automatically generated test suites presented in 3.4 and the equivalence class testing approach presented in 4, additional test procedure have manually been added to cover the missing requirements[5].

3.4 Automatically Defined Tests

The CSM model described in Section 2 contains requirement tracing information. Instead of manually translating the test cases specified for the CSM feature into LTL formulas [6], the requirement tracing information from the model can be used to automatically generate test procedures covering these requirements. In this section, we describe how the requirement tracing information defined in the model can be used to automatically generate test procedures for full requirements coverage from the CSM model. Note that the test procedures in section 3.2 also are automatically generated test procedures providing requirements coverage, but they have been generated from a manually developed formal test case specification. The approach presented here uses the requirement tracing information defined in the model to automatically generate the formal test case specification and in a second step to automatically generate the test procedures covering these test cases. Through this, a test suite is generated in a fully automated way that covers all requirements represented in the model.

The test generation tool from our experiments platform automatically generates test cases for different kinds of model coverage strategies from a test model[7]:

- **Basic Control State Coverage (BCS)**
  This type of behavioural coverage aims at covering each basic control state of each state machine at least once. No additional objectives are made about concurrent control states or accompanying variable valuations when reaching the control state under consideration.
- **Transition Coverage (TR)**
  Transition coverage aims at covering each transition of every state machine in the model.

[5] See table 7

[6] as described in the previous section 3.2

[7] More detailed explanation on the coverage criteria may be found in [20].

Table 8. Model Derived Requirements Coverage Tests

Test Procedure Name	Requirements	Number of Test Cases
TP-REQ-3.13.10.2.1	REQ-3.13.10.2.1	1
TP-REQ-3.13.10.2.2	REQ-3.13.10.2.2	2
TP-REQ-3.13.10.2.3	REQ-3.13.10.2.3	10
TP-REQ-3.13.10.2.4	REQ-3.13.10.2.4	3
TP-REQ-3.13.10.2.5	REQ-3.13.10.2.5	2
TP-REQ-3.13.10.3.1	REQ-3.13.10.3.1	2
TP-REQ-3.13.10.3.2	REQ-3.13.10.3.2	18
TP-REQ-3.13.10.3.3	REQ-3.13.10.3.3	24
TP-REQ-3.13.10.3.4	REQ-3.13.10.3.4	16
TP-REQ-3.13.10.3.5	REQ-3.13.10.3.5	6
TP-REQ-3.13.10.3.6	REQ-3.13.10.3.6	4

Again, no restrictions are made regarding variable valuations, control states and concurrent transitions to be performed when the one under consideration is triggered.

- **MC/DC Transition Coverage (MCDC)**
  Modified condition/decision (MC/DC) coverage is a variant of transition coverage, where non-atomic guard conditions are evaluated in a systematic manner.

- **Hierarchic Transition Coverage (HITR)**
  For a transitions emanating from higher-level control states, different underlying basic control states can be active when the transition is triggered. Hierarchic transition coverage aims at exercising these transitions once for every underlying basic control state being active.

- **Basic Control States Pair Coverage (BCSPAIR)**
  For concurrent state machines pairs of states of two different state machines have to be active simultaneously. This strategy does not apply to the test model under consideration, because it does not contain concurrent state machines. [8]

The test model contains requirements together with satisfy relations linking them to model elements. These requirements are taken directly from the ETCS specification ([11]). The requirements 3.13.10.3.3 and 3.13.10.3.4 have been refined into sub-requirements to allow better tracing to model elements and to be able to define the requirements coverage in more detail. Satisfy relations are defined in the model that link transitions, basic control states or LTL-formulas to requirements. The test generation tool automatically generates test cases that cover states, transitions or user defined LTL constraints with the different test strategies described above. The satisfy relations defined in the model are used to determine the set of automatically generated test cases that satisfy a requirement. A requirement is covered if the respective set of the automatically generated test cases (generated from the model) has been covered.

For each of the requirements from the ETCS specification, one test procedure has been generated that covers all automatically generated test cases that satisfy the requirement (or any of its sub-requirements). Whether it is possible to completely cover a requirement in a single test procedure and how well requirement can be represented in a test model highly depends on the requirements and on the test model. In this case, we were able to link all requirements [9] from the ETCS specification to test model elements and to automatically generate a test suite that provides full requirements coverage. Table 8 also provides the number of test cases that are covered during

[8] This coverage criteria does not apply to the CSM test model described in chapter 2, as it does not include concurrent state machines.

[9] All requirements under consideration that are relevant for ceiling speed monitoring

the test procedure and that are necessary to test the requirement. Naturally test cases can occur in multiple test procedures.

4 Improving Tests through Equivalence Class Testing Strategy

In [14], two of the authors have presented a novel complete input equivalence class partition (IECP) testing strategy. In [6] this approach has first been applied to the Ceiling Speed Monitor and the analysis of manually created mutants indicated that this strategy is superior to model derived test cases relying on structural criteria of the state machine. In [15] the approach was evaluated with automatically generated mutants from a Java implementation. Here different refinements of the strategy were compared and evaluated. For the Ceiling Speed Monitor a mutation score of 100 % could be achieved using a combination of equivalence class testing and boundary value tests.

4.1 Semantic Domain.

The novel equivalence class partition testing strategy presented in [14] is applicable to deterministic, livelock-free systems with conceptually infinite input domains and finite internal state and output domains. "Conceptually infinite" means that the domains are too large to be explicitly enumerated for test purposes. This includes physical models with real-valued inputs, but can also apply to finite but very large data types such as 64 bit integers or doubles as used in typical programming languages or modelling formalisms. As pointed out in [14, 6], this class of systems is quite significant in the embedded systems domain: typical candidates are controllers processing analogue inputs and deriving discrete control decisions from these inputs, such as thrust reversal controllers in aircrafts, or the speed monitors and airbag controllers described in this paper.

The strategy has been proven to be complete on the semantic domain of *Reactive Input Output State Transition Systems (RIOSTS)* $\mathcal{S} = (S, s_0, R, V, D)$. These systems have state spaces S, initial state $s_0 \in S$, and transition relations $R \subseteq S \times S$. Their state spaces consist of valuation functions $s : V \to D$, where V is a set of variable symbols and D is the union of all variable domains. The variable symbols can be partitioned into $V = I \cup M \cup O$, where I comprises input variables, M (internal) model variables, and O output variables. RIOSTS distinguish between *quiescent* states $s \in S_Q$ and *transient* states $s' \in S_T$, such that $S_Q \cup S_T$ partitions the state space S. Transitions from quiescent states only change input valuations, while internal model variables and output variables remain unchanged. The resulting post-states may be quiescent or transient. Transitions from transient states always have uniquely determined quiescent post-states (so we only allow deterministic RIOSTS here), and the associated transitions leave the inputs unchanged. This concept represents a natural abstraction of timed formalisms, where delay transitions allow for time to pass and inputs to be changed, while discrete transitions produce output and change internal state, but are executed in zero time [3, p. 687].

By associating atomic propositions AP with free variables in V, any RIOSTS can be extended to a Kripke Structure [8] $K(\mathcal{S}) = (S, s_0, R, V, D, L, AP)$. The labelling function $L : S \to 2^{AP}$ maps $s \in S$ to the set of all atomic propositions $p \in AP$ that evaluate to `true`, when replacing every free variable v of p by its valuation $s(v)$ in state s.

Notation.

In the exposition below, variable symbols are enumerated with the naming conventions $I = \{x_1, \ldots, x_k\}$, $M = \{m_1, \ldots, m_p\}$, $O = \{y_1, \ldots, y_q\}$. We use notation $\vec{x} = (x_1, \ldots, x_k)$ for input variable vectors, and their valuation in state s is written as $s(\vec{x}) = (s(x_1), \ldots, s(x_k))$. $D_I =$

$D_{x_1} \times \cdots \times D_{x_k}$ denotes the Cartesian product of the input variable domains. Tuples $\vec{m}, \vec{y}$ and D_M and D_O are defined over model variables and outputs in an analogous way. By $s \oplus \{\vec{x} \mapsto \vec{c}\}$, $\vec{c} \in D_I$ we denote the state s' which coincides with s on all variables from $M \cup O$, but maps the input vector to valuation $s'(\vec{x}) = \vec{c}$. For $(s_1, s_2) \in R$ we also use the shorter expression $R(s_1, s_2)$. Restricting a state s to variable symbols from a set $U \subseteq V$ is denoted by $s|_U$. This function has domain U and coincides with s on this domain.

4.2 Application to Concrete Modelling Formalisms.

The test strategy described below is elaborated on the semantic domain of RIOSTS. Every concrete modelling formalism whose behavioural semantics can be represented by RIOSTS is automatically equipped with such a test strategy: the concrete model M is translated into its corresponding RIOSTS $\mathcal{S}$. Then the test strategy is applied to $\mathcal{S}$, and this results in a set of test cases, each case represented by a finite sequence of inputs to the SUT. When executing the test cases, the transition relation of $\mathcal{S}$ is used to determine whether the SUT's reactions to these input sequences are adequate. In this article, concrete models are expressed by SysML state machines, and these can be associated with RIOSTS semantics which is consistent with the semi-formal specification of state machine behaviour in the UML/SysML standards [22, 21].

4.3 Equivalence Classes.

We use the term *trace* to denote finite sequences of states, input vectors, or output vectors. Applying a trace $\iota = \vec{c}_1 \ldots \vec{c}_n$ of input vectors $\vec{c}_i \in D_I$ to an RIOSTS $\mathcal{S} = (S, s_0, R, V, D)$ residing in some quiescent state $s \in S$ stimulates a sequence of state transitions, each pair of consecutive states connected by the transition relation R, and with associated output changes as triggered by these inputs. Restricting this sequence to quiescent states, this results in a trace of states $\tau = s_1.s_2 \ldots s_n$ such that $s_i(\vec{x}) = \vec{c}_i, i = 1, \ldots, n$, and $s_i(\vec{y})$ is the last STS output resulting from application of $\vec{c}_1 \ldots \vec{c}_i$ to state s.[10] This trace τ is denoted by s/ι. The restriction of s/ι to output variables is denoted by the trace $(s/\iota)|_O$. Since transient states have unique quiescent post-states, $(s/\iota)|_O$ is a uniquely determined output trace. Two quiescent states s, s' are *I/O-equivalent*, written $s \sim s'$, if every non-empty input trace ι, when applied to s and s', results in the same outputs, that is, $(s/\iota)|_O = (s'/\iota)|_O$. Two STS $\mathcal{S}, \mathcal{S}'$ with the same input domain are I/O-equivalent, if their initial states are I/O-equivalent. Note that $s \sim s'$ asserts equivalent I/O-behaviour *in the future*, while it still admits that states s and s' show different output valuations, i.e. $s|_O \neq s'|_O$.

Since I/O-equivalence $\sim$ is an equivalence relation on quiescent states, we can factorise S_Q with respect to $\sim$. The *initial input equivalence class partitioning (IECP)* $\mathcal{I} \subseteq \mathbb{P}(D_I)$ associated with $S_Q/_\sim$ is the coarsest partitioning of D_I such that for all $\mathbf{q} \in S_Q/_\sim$, $X \in \mathcal{I}$, there exists a uniquely determined I/O-equivalence class $\delta(\mathbf{q}, X) \in S_Q/_\sim$, such that

$$\forall s \in \mathbf{q}, \vec{c} \in X : s/\vec{c} \in \delta(\mathbf{q}, X) \quad \textbf{(4)}$$

and there exists a well-defined output $\omega(\mathbf{q}, X) \in D_O$, such that

$$\forall s \in \mathbf{q}, \vec{c} \in X : (s/\vec{c})|_O = \omega(\mathbf{q}, X) \quad \textbf{(5)}$$

It is shown in [14] that $S_Q/_\sim$ is finite if the RIOSTS $\mathcal{S}$ has finite internal state domains and finite output domains, while the input domains may be infinite. Moreover, the coarsest partitioning $\mathcal{I}$ exists, and it is finite and uniquely determined under these prerequisites. For these RIOSTS,

[10] Observe that the restriction to quiescent states does not result in a loss of information. Every transient state has the internal and output variable valuations coinciding with its quiescent pre-state, and its input valuation is identical to that of its quiescent post-state.

properties (4) and (5) induce an abstraction to DFSMs with state space $S_Q/_\sim$, input alphabet $\mathcal{I}$, and output alphabet D_O: (4) specifies a well-defined total transition function $\delta : S_Q/_\sim \times \mathcal{I} \to S_Q/_\sim$, and (5) a well-defined output function $\omega : S_Q/_\sim \times \mathcal{I} \to D_O$. When partitioning $\mathcal{I}$ further to a refined IECP $\overline{\mathcal{I}}$, the characteristic properties (4),(5) are preserved.

A finite sequence $X_1 \dots X_k, X_i \in \mathcal{I}$ is called an *abstract test case*: concrete test input vectors $\vec{c}_i$ can be selected from each X_i, and, when applied to the initial state s_0, this selection induces a trace $s_1 \dots s_k$ of quiescent states, such that

$$\exists \mathbf{q}_1, \dots, \mathbf{q}_k \in S_Q/_\sim : \forall i \in \{1, \dots, k\} : s_i \in \mathbf{q}_i \wedge \mathbf{q}_i = \delta(\mathbf{q}_{i-1}, X_i)$$

The IECP properties imply that the *expected results* associated with this test case are then specified by the output trace $\omega(\mathbf{q}_{i-1}, X_i), i = 1, \dots, k$.

In [14] an algorithm for calculating $S_Q/_\sim$ and $\mathcal{I}$ is given. This algorithm produces propositions over variables from V, specifying the members of $S_Q/_\sim$ and $\mathcal{I}$, respectively. Making use of an SMT solver, the algorithm allows for identifying the reachable I/O-equivalence classes $\mathbf{q} \in S_Q/_\sim$. As a consequence, every proposition characterising an abstract test case $X_1 \dots X_k$ is actually feasible: this means that we can find concrete traces in $\mathcal{S}$ such that, after deleting the transient states, the resulting quiescent state sequence $s_0.s_1 \dots s_k$ fulfils $s_i \in \mathbf{q}_i$ for $i = 0, \dots, k$ and $s(\vec{x}) \in X_i$ for $i = 1, \dots, k$.

For the model described in section 2, input equivalence classes are unions of convex subsets of $\mathbb{R}^n$. It should be noted, however, that the notion of I/O-equivalence and IECPs introduced here is far more general, since arbitrary propositional specifications of I/O-equivalence classes can be handled by the underlying theory. The input equivalence classes identified in [14, Example 1], for example, contain members z specified by conditions $z \bmod m = n$.

4.4 Fault Models.

For the semantic domain of RIOSTS, the fault models $\mathcal{F} = (\mathcal{S}, \sim, \mathcal{D}(\mathcal{S}, m, \overline{\mathcal{I}}))$ are specified as follows. The reference models $\mathcal{S}$ are semantic RIOSTS representations of models elaborated in concrete formalisms, such that the expected behaviour of the SUT is specified by $\mathcal{S}$ up to I/O-equivalence. We use I/O-equivalence as conformance relation.

Positive integer m fulfils $m \geq n$, where n is the number of I/O-equivalence classes of $\mathcal{S}$. IECP $\overline{\mathcal{I}}$ is a refinement of the initial coarsest IECP $\mathcal{I}$ associated with $\mathcal{S}$. Then the members $\mathcal{S}'$ of the fault domain $\mathcal{D}(\mathcal{S}, m, \overline{\mathcal{I}})$ are RIOSTS specified as follows.

1. The states of $\mathcal{S}'$ are defined over the same I/O variable space $I \cup O$ as defined for the model $\mathcal{S}$.
2. Initial state s'_0 of $\mathcal{S}'$ coincides with initial state s_0 of $\mathcal{S}$ on $I \cup O$.
3. $\mathcal{S}'$ generates only finitely many different output values.
4. $\mathcal{S}'$ has a well-defined reset operation allowing to re-start the system from its initial state.
5. The number of I/O-equivalence classes of $\mathcal{S}'$ is less or equal m.
6. If $\mathcal{I}$, $\mathcal{I}'$ are the initial coarsest IECP of $\mathcal{S}$, $\mathcal{S}'$, respectively, fulfilling the characteristic properties (4), (5), then $\overline{\mathcal{I}}$ fulfils the following *adequacy condition*:

$$\forall X \in \mathcal{I}, X' \in \mathcal{I}' : (X \cap X' \neq \emptyset \Rightarrow \exists \overline{X} \in \overline{\mathcal{I}} : \overline{X} \subseteq X \cap X') \quad \textbf{(6)}$$

The intuition behind the adequacy condition 6 is as follows. Every possible behaviour of a fault domain member $\mathcal{S}'$ can be exercised by visiting a state in some I/O-equivalence class $\mathbf{q}'$ and applying an input of some IECP member $X' \in \mathcal{I}'$ to this state. Using the refined IECP $\overline{\mathcal{I}}$ in the test suite as described below, ensures that an input from $\overline{X} \subseteq X' \in \mathcal{I}'$ will be selected when $\mathcal{S}'$ resides in $\mathbf{q}'$, so the behaviour associated with $(\mathbf{q}', X')$ will be stimulated in at least one of the test cases. If, when in a state of $\mathbf{q}'$, $\mathcal{S}'$ conforms to the behaviour of $\mathcal{S}$ for all inputs from $X \setminus X'$, but fails for inputs from $X \cap X'$, inputs selected from $\overline{X} \subseteq X \cap X'$ will uncover this error.

Conversely, suppose now that the reference model $\mathcal{S}$ behaves differently, when IECs $X_1, X_2 \in \mathcal{I}$ are applied in some state $\mathbf{q}$. Suppose further that $\mathcal{S}'$ fails to make this case distinction in a corresponding state $\mathbf{q}'$. Then there exists $X' \in \mathcal{I}'$ such that $\mathcal{S}'$ shows the same behaviour for all $\vec{c} \in X'$, but $X_1 \cap X' \neq \emptyset$ and $X_2 \cap X' \neq \emptyset$, so two different behaviours should be visible according to the reference model. Now the adequacy condition guarantees that there exist two IEC $\overline{X}_1, \overline{X}_2 \in \overline{\mathcal{I}}$, such that $\overline{X}_1 \subseteq X_1 \cap X'$ and $\overline{X}_2 \subseteq X_2 \cap X'$. As a consequence, if inputs from every input class of $\overline{\mathcal{I}}$ are exercised, the behavioural differences for inputs from $X_1 \cap X'$ and $X_2 \cap X'$ will be revealed. Summarising, the adequacy condition ensures that the IECP $\overline{\mathcal{I}}$ from where input data to the SUT is selected is fine-grained enough to stimulate every possibly deviating behaviour of $\mathcal{S}$ and $\mathcal{S}'$. These facts are exploited in the complete test strategy described next.

4.5 Complete Finite Test Suite.

The complete DFSM abstraction $\mathcal{M}$ of $\mathcal{S}$ with states $S_{Q/\sim}$, input alphabet $\overline{\mathcal{I}}$, transition function and output function as characterised in (4), (5), allows for application of finite complete DFSM testing strategies, such as the *W-Method* introduced in [7, 32]. The general form of a W-Method test suite is

$$\mathcal{W} = P.(\bigcup_{i=0}^{m-n} \overline{\mathcal{I}}^i.W) \tag{7}$$

where P is the state transition cover, $\overline{\mathcal{I}}^i$ denotes the input trace segments of length i, and W is the characterisation set. Every test of $\mathcal{W}$ consists of a (possibly empty) input trace from P, concatenated with an arbitrary input trace of length zero up to $m - n$, and terminated by an input trace from the characterisation set. P is the union of a *state cover* C and a *transition cover* $C.\overline{\mathcal{I}}$: C contains the empty trace ε, and for any state $\mathbf{q}$ of $\mathcal{M}$, there exists an input trace in C which, when applied to the initial state, ends at $\mathbf{q}$. The transition cover is defined by $C.\overline{\mathcal{I}} = \{\iota.X \mid \iota \in C, X \in \overline{\mathcal{I}}\}$. Summarising, the input sequences of a state transition cover ensure that (1) every state of the reference DFSM $\mathcal{M}$ associated with the reference model $\mathcal{S}$ is visited, and (2) every transition from every state is exercised. A characterisation set is a set of input traces distinguishing each pair of states in a minimal DFSM. Using minimisation algorithms such as the one specified in [12], characterisation sets can be constructed as a by-product of the minimisation process.

The test suite generated according to (7) is called an *abstract test suite*, because its elements are abstract test cases as defined above: the inputs to be used in each test case are not yet represented by concrete input vectors $\vec{c}$, but by input equivalence classes $\overline{X} \in \overline{\mathcal{I}}$. For creating an executable test suite, inputs $\vec{c} \in \overline{X}$ have to be selected for every $\overline{X} \in \overline{\mathcal{I}}$.

The W-Method is complete for the fault model of all DFSM over the same input and output alphabet and with at most m states. It is shown in [14] that the associated test suites with concrete inputs $\vec{c} \in \overline{X}$ are also complete for $\mathcal{F} = (\mathcal{S}, \sim, \mathcal{D}(\mathcal{S}, m, \overline{\mathcal{I}}))$. This completeness result is independent on the choice of concrete input data selected from each input equivalence class $X \in \overline{\mathcal{I}}$.

The fault domain $\mathcal{D}(\mathcal{S}, m, \overline{\mathcal{I}})$ introduced above can be extended by increasing m or by refining $\overline{\mathcal{I}}$. Increasing m increases the maximal length of input sequences in test cases in a linear way. This affects the size of the test suite exponentially, but allows for fault domain members with higher *recurrence diameters* r [4]: this is the length of the longest loop-free path in a Kripke structure. Erroneous SUT behaviour that only occurs at the end of such a longest loop free path may only be detected if the test cases use input sequences that are long enough to traverse the SUT state up to the length of the recurrence diameter.

Refining $\overline{\mathcal{I}}$ increases the size of the IECP, and this size increases the number of test cases in a polynomial way. It has to be noted, however, that uniformly refining all members of $\overline{\mathcal{I}}$ – for example, by using a sub-paving strategy as it is well known from interval analysis [16] – increases the size of the IECP exponentially with each new refinement step. The resulting fault domain contains members $\mathcal{S}'$ possessing narrower *trapdoors*: these are refined input guard conditions $g \wedge \delta$ applicable in certain $\mathcal{S}'$-states, where $\mathcal{S}'$ should behave uniformly for all inputs satisfying g. The true behaviour of $\mathcal{S}'$, however, conforms to the expected behaviour modelled by $\mathcal{S}$ only for inputs fulfilling $g \wedge \neg\delta$, while erroneous behaviour is revealed for inputs satisfying $g \wedge \delta$.

4.6 Refinement and Randomisation of Equivalence Partition Tests

As we have seen above, the fault domain $\mathcal{D}(\mathcal{S}, m, \overline{\mathcal{I}})$ can be enlarged via m or $\overline{\mathcal{I}}$. As a drawback this seriously affects the size of the resulting complete test suite $\mathcal{W}$. Therefore a refinement of the input equivalence class partitioning $\overline{\mathcal{I}}$ should be used which effectively increases the resulting test suites ability to uncover faults. In [15] two refinement heuristics were investigated and it could be shown that these heuristics are able to improve the test strength of the resulting test suite. Basically these two improvements work as follows:

1. An input partitioning $\overline{\mathcal{I}}$ is used. $\overline{\mathcal{I}}$ is a refinement of the initial coarsest IECP and reflects all case distinctions visible in guard conditions.
2. Additionally all boundary value conditions can be used in order to generate boundary value tests of the original equivalence classes [11].

In [15] another improvement of the equivalence class approach has been presented. This strategy aims at increasing the test strength of $\mathcal{W}$ for SUTs $\mathcal{S}''$ whose true behaviour is reflected by RIOSTS *outside* $\mathcal{D}(\mathcal{S}, m, \overline{\mathcal{I}})$. For obvious reasons it is assumed that these SUTs still fulfil the RIOSTS compatibility requirements 1 – 4 of the fault domain definition. This means that $\mathcal{S}''$ may have more than m I/O-equivalence classes and may need an IECP that is more fine-grained than $\overline{\mathcal{I}}$, but it is still assumed that $\mathcal{S}''$ is an RIOSTS using the same I/O variables and possessing the same visible initial state and fulfilling a reset condition.

To this end, we observe that the completeness property of the test suites introduced above does not depend on the concrete values selected from each input equivalence class $\overline{X} \in \overline{\mathcal{I}}$. For members $\mathcal{S}' \in \mathcal{D}(\mathcal{S}, m, \overline{\mathcal{I}})$ it would suffice to fix one input vector $\vec{c}_{\overline{X}}$ for every $\overline{X} \in \overline{\mathcal{I}}$. Alternatively, we could also choose different members at random, each time an input from some class $\overline{X}$ is required according to the abstract test suite definition. While this alternative would not affect the

[11] This can be done by adding extra constraints defining the boundaries of the original equivalence classes. As an example consider the guard condition $x \leq n$ restricting the input variable x in input equivalence class $\overline{X} \in \overline{\mathcal{I}}$ to values less or equal n. In this case the additional constraint γ defined as $x = n$ yields two new equivalence classes $\overline{X}_\gamma \subset \overline{X}$ and $\overline{X}_{\neg\gamma} \subset \overline{X}$ with $\overline{X}_\gamma \cap \overline{X}_{\neg\gamma} = \emptyset$. $\overline{X}_\gamma$ is the set of input vectors $\vec{c}_{\overline{X}_\gamma}$ that fulfill $x = n$ and thus are the boundaries of the original equivalence class $\overline{X}$ containing all input vectors, where $x < n$ or $x = n$.

Table 9. Requirements Coverage and Model Coverage Overview.

	Manually Defined Tests (A)	Extended Manual Tests (B)	Automatically Defined Tests (C)	Equivalence Class Testing (D)
Requirements Coverage (ETCS specification)	24 of 29 (82,76%)	29 of 29 (100%)	29 of 29 (100%)	29 of 29 (100%)
Test Cases	9	11	45	5524
Model Coverage States	9 of 9 (100%)	9 of 9 (100%)	9 of 9 (100%)	9 of 9 (100%)
Model Coverage Transitions	10 of 10 (100%)	10 of 10 (100%)	10 of 10 (100%)	10 of 10 (90%)
Model Coverage MCDC	8 of 10 (80%)	8 of 10 (80%)	10 of 10 (100%)	10 of 10 (100%)
Model Coverage HITR	1 of 5 (20%)	3 of 5 (60%)	5 of 5 (100%)	5 of 5 (100%)

suite's completeness property when applied against members of $\mathcal{D}(\mathcal{S}, m, \overline{\mathcal{I}})$, it favourably affects the test strength against RIOSTS outside $\mathcal{D}(\mathcal{S}, m, \overline{\mathcal{I}})$: the chances for uncovering trapdoors are obviously increased. This approach results in an *adaptive random testing strategy*, where the selection of input data is no longer performed uniformly over the complete input domain, but selectively for each input equivalence class $\overline{X} \in \overline{\mathcal{I}}$. Moreover, the random values from such an $\overline{X}$ are only applied when an $\overline{X}$-input is required according to the abstract test suite constructed from Equation (7). The experimental results in [15] indicate, that the randomisation of the input selection is able to improve the mutation score for erroneous implementations that are *outside* the fault domain without increasing the size of the resulting test suite and without nullifying the completeness result for implementations inside the fault domain.

5 Experimental Results

This section compares the generated test suites presented in the previous chapters in terms of requirements coverage, model coverage and test strength. Test suites with the same requirements coverage can provide different model coverage. Higher model coverage indicates higher test strength, but test suites with the same model coverage can still be of different test strength. Mutations of the System Under Test are used to measure the test strength of the generated test suites.

We will refer to the different test suites as *Test Suite A* (test procedures generated automatically from manually defined test case specifications) [12], *Test Suite B* (test procedures generated automatically from manually defined test case specifications including additional test cases to cover all remaining sub-requirements) [13], *Test Suite C* (test procedures generated automatically from automatically generated test case specifications) [14] and *Test Suite D* (test procedures generated using the *Equivalence Class Testing* strategy). [15] Section 5.1 presents the requirement and model coverage results and section 5.2 compares the test strength of the different test suites.

5.1 Requirements and Model Coverage

[12] explained in section 3.2.
[13] explained in section 3.3.
[14] explained in section 3.4.
[15] as explained in section 4 including the refinements and randomisation as presented in section 4.6.

In Table 9 we compare the requirements and the model coverage of the test suites. Although all test suites cover all states and transitions of the model at least once, *Test Suite A* does not cover all requirements completely (not all sub-requirements are covered). Each of the other three test suites covers all requirements of ETCS defined in [10] including the sub-requirements from our analysis.

Though *Test Suite B* does cover all requirements, it does not cover all test cases that are automatically generated from the test model. Fewer combinations for guard conditions and high level transitions are executed with this test suite.

The fully automatically generated *Test Suite C* provides full requirements coverage which is expected, as this is the goal of the test suite. It also provides full model coverage which is an indication for testable requirements, a model that has been developed for testing purposes and for carefully defined satisfy relations from model elements to requirements.

The test traces that are generated for *Test Suite D* have been evaluated on the test model as well and through this, a test case coverage and requirements coverage has been calculated using the tracing information from the test model. *Test Suite C* and *Test Suite D* both achieve the same requirements and model coverage, but *Test Suite D* uses more test cases to reach this coverage. In the following section, we will evaluate if the additional test cases result in an increased test strength of this test suite.

5.2 Test Strength Comparison

Experimental Setup.

The strength of a test suite is the ability to detect real faults. [18] shows that mutants are a valid substitute for real faults when comparing generated test suites. Mutants are modifications of the SUT with (automatically) seeded single faults. Mutants in large numbers can be generated by mutation operators, which apply slight syntactical changes to the original code. To compare the test strength of the generated test suites we measured the mutation score of each test suite. The mutation score is the ratio of mutants that were “killed” by a test suite to the total number of *real* mutants[16]. A mutant is killed, if at least one test case of the test suite does not pass.

For the experimental evaluation we generated a correct Java implementation from the model shown in section 2. The java implementation was performed by hand in a straight forward way, resulting in 176 lines of code. Next, mutants were automatically generated from each implementation with the Major mutation framework [17]. All applicable mutation operators were executed to generate single-fault mutants. For our concrete implementations these operators were as follows: arithmetic operator replacement (AOR), conditional operator replacement (COR), relational operator replacement (ROR), statement deletion (STD) and literal value replacement (LVR). Note that the mutation tool is unaware of any conformance relation. Therefore we manually investigated the generated mutants. Discarding I/O-equivalent mutants resulted in a collection of 186 erroneous implementations. Finally, the three test suites specified above were executed against all mutants to measure the mutation score of the test suites.

Experimental Results.

[16] Only non-I/O-equivalent mutants are considered, since not every syntactical modification results in a mutant showing deviating observable behaviour.

Table 10. Mutation Score Overview.

	Manually Defined Tests	Extended Manual Tests	Automatically Defined Tests	Equivalence Class Testing
Number of Test Cases	9	11	45	5524
Mutation Score	108 of 186 (58 %)	116 of 186 (62 %)	126 of 186 (68 %)	186 of 186 (100 %)

Table 10 shows the mutation scores of the test suites. The manually defined test cases achieve the lowest mutation score of all examined strategies. This seems to correlate with the low model coverage shown in the previous section. Further, the mutation score of the model derived tests is superior to the manually defined tests, as already expected by the higher (complete) model coverage. The last column of Table 10 confirms, that for our model the equivalence class testing strategy behaves best in revealing the faults injected by the mutation tool. The test suite revealed all but one mutation.[17] Although the model derived test cases yield a full coverage of the test model, this criterion is not sufficient to reveal a considerable number of faults. However, the equivalence class testing approach uses more rigorous criteria and this results in the detection of defects, which would not be revealed by test suites considering only model coverage criteria as described in section 3.4. Note that the large number of test procedures needed for the equivalence class tests is due to the detailed refinement and the use of boundary value tests. The equivalence class testing strategy can be applied with a coarser IECP for the cost of lower test strength. The reader may refer to [15] for a detailed evaluation of the different refinements and the impact on size and mutation score.

6 Related Work

The European Train Control System (ETCS) official specification includes a set of test cases to ensure the functional conformity, compatibility and interoperability of train onboard units. Railway domain experts designed these ETCS standard tests in the Subset 076 [10]. They are created manually from the System Requirement Specification (SRS), the requirement coverage could then be evaluated. The chapter [19] shows how the tests are actually used in their test environment execution to prove the technical interoperabilty on the onboard units.

The authors [1] point out the deficiency of the test specification for safety assessment. Furthermore, in [5] a method is proposed to generate new tests respecting the existing tests of the standard, using a formalism to improve the latter by automatically generating the tests from the specification.

In [29, 30] the authors propose to start from the ETCS standard tests to generate a functional behavior prototype of the onboard unit. The prototype can be a starting point for the architects or can be refined to a stronger test model, it can also be used as a model-in-the-loop software within a simulation environment. Their idea is to take advantage of the huge amount of domain specific knowledge that experts have put in designing the tests. Moreover, the tests cases are already linked to the requirements, hence requirements can be directly traced to the model.

[17] Note that this result differs slightly from the result in [15]. There all mutants were killed. The reason for this deviation results from a different model we use in this paper. This model has more inputs than the reduced version in [15].

7 Conclusions

Discussion of results

In this article, the ceiling speed monitoring function of the European Train Control System has been used as a case study for evaluating various model-based testing strategies and comparing their strength with the associated tests contained in the ETCS standard, the latter having been developed by domain experts in a manual way. A test model of the CSM has been presented. It is expressed in SysML and interpreted in a formal behavioural semantics based on Kripke structures. The MBT strategies comprised "standard" model coverage techniques (basic control state, transition, MC/DC coverage, and hierarchic transition coverage applicable to composite state machines), as well as a complete input equivalence class testing strategy. The comparison of the strategies involved focused on requirements coverage and test strength. It turned out that the manually developed test suite contained in the ETCS standard has some deficiencies regarding requirements coverage; as a result, their test strength is considerably lower than the strength of any of the MBT strategies.

The test strength of the different strategies has been evaluated using a CSM reference implementation programmed in Java, from which 368 mutations have been generated. The complete input equivalence class testing strategy turned out to have significantly higher test strength (mutation score of nearly 100%), when compared to any of the other strategies, and the test suite contained in the ETCS standard was significantly weaker than the other strategies. Suggestions were presented how the latter suite should be extended, resulting in moderate additional effort only.

The nearly complete mutation score achieved by the equivalence class method confirms the hypothesis that complete testing strategies are also of great practical value when applied against implementations that are outside the specified fault domain where *every* SUT error is detected by at least one test case. On the other hand, the complete strategy increased the test suite size by a factor of 100 and more in comparison to the suites resulting from the other strategies. This indicates that the complete strategies may be less suitable for applications outside the safety-critical systems domain, where lesser test strength may still be acceptable. It has been analysed why the non-complete MBT strategies still resulted in a mutation score of 85%, and it was pointed out that this is due to the simple decision structure in the CSM, where guard conditions are relatively wide and most control modes can be reached from any other mode in one step.

Ongoing work

The nature of the input equivalence classes used in the CSM case study is quite "conventional" in the sense that they are made up from intervals of real numbers. The underlying equivalence class testing theory, however, does not depend on this simple class structure: since the equivalence concept is based on equivalent behaviours derived from the behavioural model's transition relation, the theory covers classes of discrete points in the state space as well. Exploiting this fact, the construction of equivalence classes for train configurations in railway networks is currently investigated in collaboration with other authors [33], for the purpose of testing interlocking control systems with acceptable effort.

In this context, an extension of the equivalence class testing theory to nondeterministic models is currently being prepared for publication. This extension makes use of an abstraction of nondeterministic IOSTS with infinite input domains, but finite internal state and output domains to nondeterministic finite state machines (NFSMs). This abstraction enables us to transfer

complete test suites for NFSM as presented in [25, 26, 13] to complete test suites for IOSTS. The motivation of this theory extension to nondeterministic models is as follows: when performing MBT in a safety-critical context, the correctness and completeness of the test model is of utmost importance. Therefore formal verification techniques, in particular variants of model checking, are applied to verify the test model before using it for automated test suite generation. Typical verification methods that can be applied successfully for this purpose use over-approximation of the model in order to facilitate the verification process. As a consequence, the test model to be used as input for the automated testing campaign becomes nondeterministic. An example for a model verification in the interlocking control systems domain by means of bounded model checking in combination with k-induction is presented in [33].

Acknowledgements

Cécile Braunstein's work is funded by ITEA2 project openETCS under grant agreement 11025. Felix Hübner's research contribution is funded by Siemens AG in the context of the SyDE Graduate School on System Design.[18] The work of Wen-ling Huang and Jan Peleska is funded by the project *ITTCPS – Implementable Testing Theory for Cyber-physical Systems*[19] which has been granted in the context of the German Universities Excellence Initiative.[20]

References

[1] Almir Villaro Arriola, Jon Mendizabal Samper, and Juan Melédez Lagunilla. Fault injection for On-Board ERTMS/ETCS Safety assesement. In *Railway Safety, Reliability, and Security: Technologies and Systems Engineering: Technologies and Systems Engineering*, volume 1, pages 128–150. IGI Global, 2012.

[2] Saswat Anand, Edmund K. Burke, Tsong Yueh Chen, John A. Clark, Myra B. Cohen, Wolfgang Grieskamp, Mark Harman, Mary Jean Harrold, and Phil McMinn. An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software*, 86(8):1978–2001, 2013.

[3] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[4] Armin Biere, Keijo Heljanko, Tommi Junttila, Timo Latvala, and Viktor Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, 2(5), November 2006. arXiv: cs/0611029.

[5] Giuseppe Bonifacio, Pietro Marmo, Antonio Orazzo, Ida Petrone, Luigi Velardi, and Alessio Venticinque. Improvement of processes and methods in testing activities for safety-critical embedded systems. In *Computer Safety, Reliability, and Security*, pages 369–382. Springer, 2011.

[6] Cécile Braunstein, Anne E. Haxthausen, Wen-ling Huang, Felix Hübner, Jan Peleska, Uwe Schulze, and Linh Vu Hong. Complete model-based equivalence class testing for the ETCS ceiling speed monitor. In S. Merz and J. Pang, editors, *Proceedings of the ICFEM 2014*, number 8829 in Lecture Notes in Computer Science, pages 380–395. Springer, November 2014.

[7] Tsun S. Chow. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, SE-4(3):178–186, March 1978.

[8] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[18] http://www.informatik.uni-bremen.de/syde/index.php?home-en
[19] http://www.informatik.uni-bremen.de/agbs/projects/ittcps/index.html
[20] http://en.wikipedia.org/wiki/German_Universities_Excellence_Initiative

[9] European Railway Agency. *ERTMS/ETCS System Requirements Specification – Principles*. Volume Chapter 3 of SUBSET-026-3 [11], 2014. Issue 3.4.0.

[10] European Railway Agency. *ERTMS/ETCS Class 1, Test plan – SUBSET-076*. ERTMS, 2015. Version 3.0.0.

[11] European Railway Agency. *ERTMS/ETCS System Requirements Specification – SUBSET-026*. European Railway Agency, 2015. Version 3.4.0.

[12] Arthur Gill. *Introduction to the theory of finite-state machines*. McGraw-Hill, New York, 1962.

[13] Rob M. Hierons. Testing from a nondeterministic finite state machine using adaptive state counting. *IEEE Transactions on Computers*, 53(10):1330–1342, 2004.

[14] Wen-ling Huang and Jan Peleska. Complete model-based equivalence class testing. *International Journal on Software Tools for Technology Transfer*, pages 1–19, 2014.

[15] Felix Hübner, Wen-ling Huang, and Jan Peleska. Experimental evaluation of a novel equivalence class partition testing strategy. In *Proceedings of the 9th International Conference on Tests & Proofs TAP 2015*, number 9154 in Lecture Notes in Computer Science. Springer, 2015.

[16] Luc Jaulin, Michel Kieffer, Olivier Didrit, and Éric Walter. *Applied Interval Analysis*. Springer-Verlag, London, 2001.

[17] René Just. The Major mutation framework: Efficient and scalable mutation analysis for Java. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, pages 433–436, San Jose, CA, USA, July 23–25 2014.

[18] René Just, Darioush Jalali, Laura Inozemtseva, Michael D. Ernst, Reid Holmes, and Gordon Fraser. Are Mutants a Valid Substitute for Real Faults in Software Testing? In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 654–665, New York, NY, USA, 2014. ACM.

[19] Lars Ebrecht and Michael Meyer zu Hörste. Verification and validation of interoperability. In *Railway Safety, Reliability, and Security: Technologies and Systems Engineering: Technologies and Systems Engineering*, volume 1, pages 117–127. IGI Global, 2012.

[20] Wen ling Huang, Jan Peleska, and Uwe Schulze. Test automation support. Technical Report D34.1, COMPASS Comprehensive Modelling for Advanced Systems of Systems, 2013. http://www.compass-research.eu.

[21] Object Management Group. OMG Systems Modeling Language (OMG SysML^{TM}). Technical report, Object Management Group, 2010. OMG Document Number: formal/2010-06-02.

[22] Object Management Group. OMG Unified Modeling Language (OMG UML), superstructure, version 2.4.1. Technical report, OMG, 2011.

[23] Jan Peleska. Industrial-strength model-based testing - state of the art and current challenges. In Alexander K. Petrenko and Holger Schlingloff, editors, Proceedings Eighth Workshop on *Model-Based Testing,* Rome, Italy, 17th March 2013, volume 111 of *Electronic Proceedings in Theoretical Computer Science*, pages 3–28. Open Publishing Association, 2013.

[24] Jan Peleska, Artur Honisch, Florian Lapschies, Helge Löding, Hermann Schmid, Peer Smuda, Elena Vorobev, and Cornelia Zahlten. A real-world benchmark model for testing concurrent real-time systems in the automotive domain. In Burkhart Wolff and Fatiha Zaidi, editors, *Testing Software and Systems. Proceedings of the 23rd IFIP WG 6.1 International Conference, ICTSS 2011*, volume 7019 of *Lecture Notes in Computer Science*, pages 146–161, Heidelberg Dordrecht London New York, November 2011. IFIP WG 6.1, Springer.

[25] A. Petrenko and N. Yevtushenko. Adaptive testing of deterministic implementations specified by nondeterministic fsms. In *Testing Software and Systems*, number 7019 in Lecture Notes in Computer Science, pages 162–178, Berlin, Heidelberg, 2011. Springer.

[26] A. Petrenko and N. Yevtushenko. Adaptive testing of nondeterministic systems with fsm. In *2014 IEEE 15th International Symposium on High-Assurance Systems Engineering (HASE)*, pages 224–28, 2014.

[27] A. Petrenko, N. Yevtushenko, and G. v. Bochmann. Fault models for testing in context. In Reinhard Gotzhein and Jan Bredereke, editors, *Formal Description Techniques IX – Theory, application and tools*, volume 69, pages 163–177. Chapman&Hall, 1996.

[28] Alexandre Petrenko, Adenilso Simao, and José Carlos Maldonado. Model-based testing of software and systems: Recent advances and challenges. *Int. J. Softw. Tools Technol. Transf.*, 14(4):383–386, August 2012.

[29] Christoph Torens, Lars Ebrecht, and Karsten Lemmer. Inverse model based testing – generating behavior models from abstract test cases. In *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*, pages 75–78. IEEE, 2011.

[30] Christoph Torens, Lars Ebrecht, and Karsten Lemmer. Starting model-based testing based on existing test cases used for model creation. In *Computer and Information Technology (CIT), 2011 IEEE 11th International Conference on*, pages 320–327. IEEE, 2011.

[31] Jan Tretmans. Conformance testing with labelled transition systems: Implementation relations and test generation. *Computer Networks and ISDN Systems*, 29(1):49–79, 1996.

[32] M. P. Vasilevskii. Failure diagnosis of automata. *Kibernetika (Transl.)*, 4:98–108, July-August 1973.

[33] Linh H. Vu, Anne E. Haxthausen, and Jan Peleska. Formal modeling and verification of interlocking systems featuring sequential release. In Cyrille Artho and Peter Csaba Ölveczky, editors, *Formal Techniques for Safety-Critical Systems*, volume 476 of *Communications in Computer and Information Science*, pages 223–238. Springer International Publishing, 2015.