Complete Agent-driven Model-based System Testing for Autonomous Systems – Technical Report[*]

Kerstin I. Eder
Department of Computer Science, University of Bristol, United Kingdom
Kerstin.Eder@bristol.ac.uk

Wen-ling Huang
Department of Mathematics & Computer Science, University of Bremen, Germany
huang@uni-bremen.de

Jan Peleska
Department of Mathematics & Computer Science, University of Bremen, Germany
peleska@uni-bremen.de

This technical report is an extended version of the authors' submission to FMAS 2021. It is intended as a position paper, where we present a novel approach to testing complex autonomous transportation systems (ATS) in the automotive, avionic, and railway domains. It is well-suited to overcome the problems of verification and validation (V&V) effort which is known to become infeasible for complex ATS, when trying to perform V&V with conventional methods. The approach advocated here uses complete testing methods on module level, because these establish formal proofs for the logical correctness of the software. Having established logical correctness, system-level tests are performed in simulated cloud environments and on the target system. To give evidence that "sufficiently many" system tests have been performed with the target system, a formally justified coverage criterion is introduced. To optimise the execution of very large system test suites, we advocate an online testing approach where multiple tests are executed in parallel, and test steps are identified on-the-fly. The coordination and optimisation of these executions is achieved by an agent-based approach. Each aspect of the testing approach advocated here is shown to be consistent with existing standards for development and V&V of safety-critical transportation systems, or it is justified why they should become acceptable in future revisions of the applicable standards.

1 Introduction

Motivation

Autonomous transportation systems (ATS) in the automotive, avionic, or railway domains are highly complex and at the same time safety-critical. It is a well-known fact that the verification and validation (V&V) effort (including the test effort) for assuring an acceptable degree of safety and reliability in complex ATS will become so high that it cannot be performed with conventional methods anymore [27]. In particular, it cannot be expected that all the necessary system tests will be executable on the integrated target system (vehicle, train, or aircraft). Instead, a major portion of the tests needs to be executed concurrently in simulation environments, since otherwise test campaigns would last unacceptably long.

[*]This technical report is an extended version of the authors' submission to FMAS 2201 https://fmasworkshop.github.io/FMAS2021/. The second and third author are funded by the German Ministry of Economics, Grant Agreement 20X1908E.

Submitted in abridged form to:
FMAS 2021

This approach, however, needs special attention from the perspective of applicable safety-related standards and certification rules: for good reasons, it has to be justified why simulation environments are sufficiently trustworthy "replica" of the real target systems and their operational environments, so that certification credit can be obtained for these tests, though they have not been executed with the original equipment and the real operational environment.

Objectives

This technical report is a position paper: we outline a novel approach to testing complex ATS and justify each building block of this approach by references to existing theories from the field of formal methods or motivate them at least by means of illustrating examples. The novelty consists in a new combination of existing theories and technologies, and in a careful consideration of applicable standards and pre-standards in the automotive, railway, and avionic domains [6, 25, 26, 49].

(1) It is proposed to combine so-called *complete* software test strategies on module level with scenario-based system tests. A test suite generated according to a specific strategy is complete, if it guarantees under certain hypotheses that every correct implementation will pass all test cases and every faulty implementation will fail at least one test case. Correctness is either defined by means of a conformance relation (refinement, equivalence, or variants thereof) to a given reference model, or by means of a set of property specifications to be fulfilled by the implementation. Here, the model-based approach is used. On system level, test scenarios are created from more comprehensive system models whose behavioural semantics can be represented by *symbolic finite state machines (SFSM)* [42], extended by control state invariants for time-continuous and discrete variables. This extension of SFSMs can be interpreted as a restricted variant of *hybrid automata*, as introduced in [16]. The system-level models can be traced back to module-level models, and this relationship can be exploited to obtain meaningful coverage values for system tests.

(2) On the system level, tests are performed concurrently, following the *online testing* paradigm [30], where input data to the *system under test (SUT)* are calculated on the fly from a system model for each test step which is part of a test case. Also, the SUT reactions are checked in real-time against the system model. A portion of these concurrent system tests will be performed on the original equipment and the real operational environment, while the rest is executed in cloud-based simulation environments. To coordinate this concurrent effort, an agent-based approach is used – we use the term *agent-based system testing (ABST)*. As pointed out in [29], the main advantage of using agents in testing is their ability to perform autonomous actions. They can decide to "push" test executions into specific directions, pursuing different goals, such as coverage maximisation, prioritisation, or investigation of critical functional aspects of the SUT.

Main Contributions

To the best of our knowledge, the material discussed in this technical report considers the following aspects for the first time (see also the discussion of related work in Section 5).

1. The combination of complete testing strategies on module level with complementary system tests whose degree of completeness can also be measured.
2. The agent-based approach to maximise system test coverage during online testing.
3. The investigation of the impact of applicable existing and future standards on the admissibility of cloud-based tests for the purpose of achieving certification credit.

4. The exploitation of test models created during module testing for the purpose of system test coverage assessment.

Overview

In Section 2, an example is presented, modelling an autonomous freight train controller. This example will be used in subsequent sections to illustrate the aspects of the comprehensive test approach advocated in this technical report. In Section 3, we discuss complete test methods and their applicability to module tests in the cloud. We present our approach to agent-based system testing in the cloud and on the original target systems in Section 4 and show how results from module testing can be used to calculate coverage. In each of these sections, the certification-related aspects are discussed where appropriate. In Section 5, we discuss related work; conclusions and plans for future work are presented in Section 6.

2 Running Example – Autonomous Freight Train Control System

System Description.

Consider a control system for an autonomous freight train, with interfaces as depicted in Fig. 1. The controller is only active when powered ($\texttt{pwr} = 1$). Resetting the controller is performed by switching the power off and on again. The controller acts on the train engine with a simplified interface a carrying three commands a_- (negative acceleration, brake the train), 0 (no acceleration, keep current velocity – this state is called *coasting*), and a_+ (accelerate the train). For the sake of simplicity, only one deceleration and one acceleration value is considered. The decisions about braking or accelerating depend on several inputs. The *radio block centre (RBC)* sends a *movement authority (MA)* to proceed up to track coordinate x_B which is greater than the train's starting position x_A. The train is expected to proceed until x_B and stop there.[1] Conversely, the train transmits its current position estimate x to the RBS. Depending on this position, the RBC sends the actual maximum speed allowed (v_{Max}) to the train. An obstacle detection sensor sets controller input ω to 1 if an obstacle is detected on the track. In this situation, the train is expected to brake until it has come to a halt and/or until the obstacle has been removed. Three position sensors[2] provide their actual location estimates $x_i \in [x_A, x_B],\ i = 1,2,3$, together with confidence values $c_i \in [0,1]$. The train controller calculates a fourth location estimate based on its last location estimate, last velocity, and last acceleration. We assume that this estimate is associated with a constant confidence value c_4.

The train controller operates in processing cycles of constant duration Δt (a typical value would be $\Delta t = 0.1\ \texttt{s}$). It manages a state tuple whose components are named $(x, c, x_4, v, a, x_{\text{Stop}}, x_B)$. In this tuple, x denotes the actual aggregated position estimate with its overall confidence value c. Variable x_4 is the current position estimate derived from the physical equation (4) below. Variable v is the current velocity derived from the physical motion equation (6). Output variable a carries the current acceleration value in $\{a_-, 0, a_+\}$. Variable x_{Stop} stores the current forecast where the train would come to a standstill, if braking would be started in the next processing cycle. Finally, x_B stores the current movement authority value. The state tuple is initialised by $(x = x_A, c = 1, x_4 = x_A, v = 0, a = 0, x_{\text{Stop}} = x_A, x_B = 0)$. As soon as a movement authority $x_B > x_A + \alpha$ is received, the train controller accelerates the train by setting

[1] This part is slightly simplified: according to the ETCS standard [11], new movement authorities $x'_B > x_B$ may be received while the train is driving, so that a stop at x_B is not required.

[2] For our example, we assume three sensors obtaining location information, for example, from GPS, Balise, and distance radar.

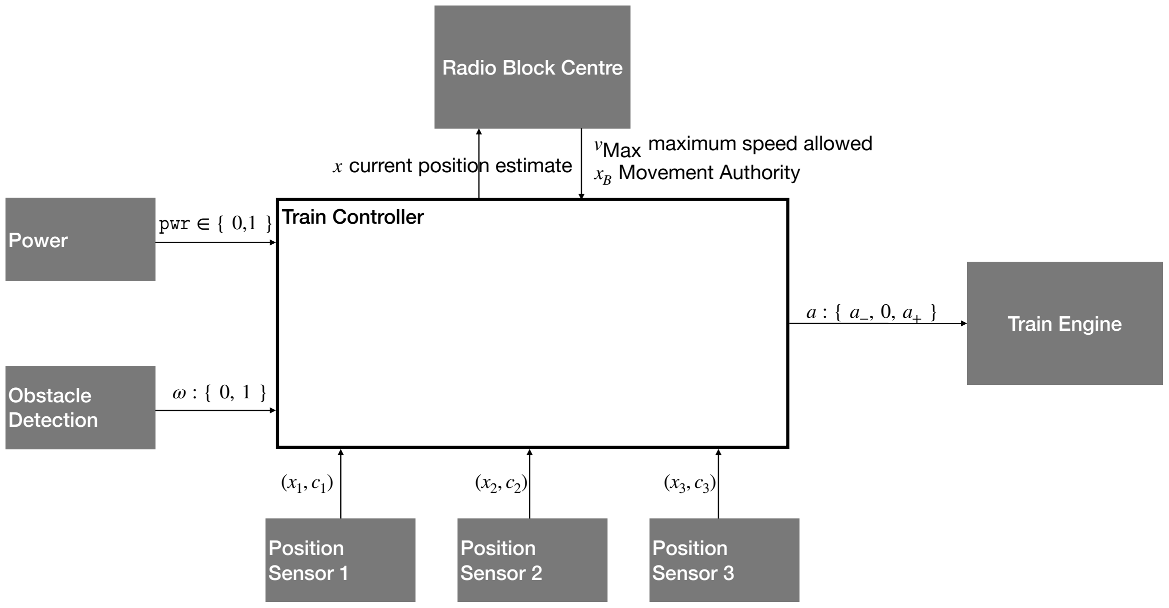

Figure 1: Train controller, location sensors, and engine interface.

$a = a_+$. The value α is a constant small distance representing the precision of a train's stopping position: if $|x_B - x| \leq \alpha$, the train is considered to be "close enough" to the position x_B and, therefore, loses its movement authority. When initialising the system, x_B is set to x_A, because no movement authority has been received yet. Conversely, condition $x_B > x_A + \alpha$ indicates that the distance to reach the authorised destination x_B is sufficently far away from starting point x_A, so that the train should be put in motion. After each processing cycle, the location estimate x_4 and the speed estimate v are updated according to the physical equations for motion with constant acceleration:

$$\Delta x = v \cdot \Delta t + \frac{a}{2} \cdot \Delta t^2, \tag{1}$$

$$\Delta v = a \cdot \Delta t, \tag{2}$$

which imply

$$v + \Delta v = 2 \cdot \frac{\Delta x}{\Delta t} - v. \tag{3}$$

Accordingly, the controller assigns new values to x_4 by

$$x_4 := x + v \cdot \Delta t + \frac{a}{2} \cdot \Delta t^2. \tag{4}$$

Next, the controller stores the old overall position estimate x in an auxiliary variable $\underline{x}$ and calculates the new position estimate x_4 according to Equation (4), with $\underline{x}$ replacing x in this formula.

The overall position estimate is calculated from the current position sensor values x_1, x_2, x_3 and x_4, taking into account their different confidence values by assigning:

$$x := \frac{\Sigma_{i=1}^{4} c_i \cdot x_i}{\Sigma_{i=1}^{4} c_i}. \tag{5}$$

The new speed estimate is than calculated according to Equation (3) with $\Delta x = x - \underline{x}$, and assigned to v as

$$v := 2 \cdot \frac{x - \underline{x}}{\Delta t} - \underline{v}, \tag{6}$$

where $\underline{v}$ denotes the previous speed estimate.

The overall confidence value for the position value obtained according to Equation (5) is

$$c = \frac{\Sigma_{i=1}^{4} c_i}{4} \tag{7}$$

If the confidence value is acceptable ($c \geq c_{\text{Min}}$), then the train is accelerated until the maximal speed v_{Max} has been reached. After that, the train starts coasting. If the confidence value is too low ($c < c_{\text{Min}}$), the train is slowed down until it has reached a safe lower speed v_{Safe} until the position confidence is acceptable again.

With the values for position and speed at hand, the earliest possible stopping position x_{Stop} is calculated under the assumption that the train would keep its current acceleration in the actual processing cycle and start braking in the next cycle. For this calculation, the following assignment can be applied, using an auxiliary variable Δ_{Stop} indicating the duration until the train comes to a stop.

$$\Delta_{\text{Stop}} := -\frac{v + a \cdot \Delta t}{a_-}, \tag{8}$$

$$x_{\text{Stop}} := x + v \cdot \Delta t + \frac{a}{2} \cdot \Delta t^2 + (v + a \cdot \Delta t) \cdot \Delta_{\text{Stop}} + \frac{a_-}{2} \cdot \Delta_{\text{Stop}}^2 \tag{9}$$

$$= x + v \cdot \Delta t + \frac{a}{2} \cdot \Delta t^2 - \frac{a_-}{2} \cdot \Delta_{\text{Stop}}^2. \tag{10}$$

To understand assignment (8), recall from Equation (2) that the speed changes to $v' = v + a \cdot \Delta t$ in the current processing cycle, where a is the current acceleration. From the next cycle on, the constant deceleration a_- will be applied, so the speed changes according to $\Delta v = a_- \cdot \tau$ over duration τ. Resolving formula $v' + \Delta v(\tau) = 0$ to τ, yields $\tau = \Delta_{\text{Stop}}$ from assignment (8) for the duration to come to a halt from velocity v'. For understanding assignment (10), recall from Equation (1) that the position x changes to $x' = v \cdot \Delta t + \frac{a}{2} \cdot \Delta t^2$ in the current processing cycle. After that, the speed v' has been reached, and deceleration a_- is applied. Applying Equation (1) with $v = v'$, $a = a_-$, and for the duration Δ_{Stop}, yields the stopping position used in assignment (9). Observing that $(v + a \cdot \Delta t) = -\Delta_{\text{Stop}} \cdot a_-$, yields Equation (10).

When the estimated stopping position is only δ meters away from the destination x_B, further acceleration is forbidden. As soon as forecast x_{Stop} reaches the value of x_B, the train brakes until a very low speed v_{Min} has been reached, from where the train can stop "immediately", that is, within less than $\alpha = 0.6\ \mathtt{m}$. If the train has closed in on x_B within $0.6\ \mathtt{m}$, it is braked again until it stops. Then the train waits for a new movement authority to continue its journey.

Typical values for the constants and boundary variables mentioned in the requirements above are

$$\begin{array}{rclrclrclrcl}
a_+ & = & 1\ \mathtt{m/s}^2, & a_- & = & -1\ \mathtt{m/s}^2, & \Delta t & = & 0.1\ \mathtt{s}, & c_{\text{Min}} & = & 0.9 \\
v_{\text{Safe}} & = & 8\ \mathtt{m/s}, & v_{\text{Max}} & = & 22\ \mathtt{m/s}, & v_{\text{Min}} & = & 1\ \mathtt{m/s}, & \delta & = & 200\ \mathtt{m} \\
\alpha & = & 0.6\ \mathtt{m} & & & & & & & & &
\end{array}$$

It can be assumed that v_{Max} is always greater than v_{Safe}.

Formal Controller Model.

The informal system description given above is now modelled using UML state machines [35]. The formal model semantics can be specified, for example, by associating a variant of Kripke structures with state machines, as described in [21]. Alternatively, the model can be flattened and transformed into a symbolic finite state machine, whose interpretation is slightly simpler, because SFSMs can be regarded as a simpler sub-class of these Kripke structures. In the following, we explain the behaviour formalised with these machines in an intuitive way.

The expected controller behaviour is modelled by two state machines. The first updates the variable vector $(x, c, x_4, v, a, x_{\text{Stop}}, x_B)$ defined above every Δt according to the equations listed above. This state machine is not shown, since it consists of a single state with a self loop triggered every Δt with an action that just performs the necessary assignments specified above.

Concurrently, the hierarchic state machine TRAIN CONTROLLER with its top-level machine specified in Fig. 2 is executed. Its normal behaviour is to transit after power on into state ACTIVE described by the lower-level state machine in Fig. 3. In the special situation where the controller is started with an obstacle in front (condition $\omega == 1$), it transits into submachine state OBSTACLE PRESENT. If the train is still driving, it will brake in state BRAKE FOR OBSTACLE until it has come to a standstill and state HALTED is entered. State ACTIVE will always be left when an obstacle is detected. It is visited again as soon as the obstacle has been removed ($\omega == 0$).

When submachine ACTIVE is entered, one or more choice states will be visited to decide which stable state should be chosen.

1. If no movement authority is available ($x_B - x \leq \alpha$), state WAIT FOR MA is entered. There, a moving train is braked to stop (see sub-machine in Fig. 4). State WAIT FOR MA is left as soon as a movement authority is available ($x_B - x > \alpha$).
2. If a movement authority exists and the train is still far enough from its destination ($x_B - x_{\text{Stop}} > \delta$), the controller branches into submachine DRIVING. There, the train will be accelerated to its maximal speed v_{Max}, as shown in state machine Fig 5. The train will be slowed down if it is too fast and accelerated if it is slower than the maximal speed allowed. If the position confidence value c is too low, the controller transits to state SAFE DRIVING, where the train is kept at velocity v_{Safe} until the position confidence is acceptable again (Fig. 6).
3. If the predicted stopping location comes as close as δ to x_B (change condition $x_B - x_{\text{Stop}} \in (0, \delta]$), the train is not allowed to accelerate further. It will be kept at a constant velocity $v_{\text{const}} \leq v_{\text{Max}}$ in state NO ACCEL (see sub-machine in Fig. 7).
4. When it is time to brake ($x_B - x_{\text{Stop}} \leq 0$), state BRAKE TO TARGET is entered (Fig. 8), where the train is slowed down to a positive speed value v_{Min}.
5. This positive speed value is maintained until the train is very close to its destination ($x_B - x \leq \alpha$). Then state STOP TRAIN is entered, where the train is slowed down to a halt.
6. The train automatically loses its movement authority when coming close to its destination ($x_B - x \leq \alpha$). Therefore, after having come to a standstill, the control will transit from STOP TRAIN to WAIT FOR MA where it stays until a new movement authority arrives.

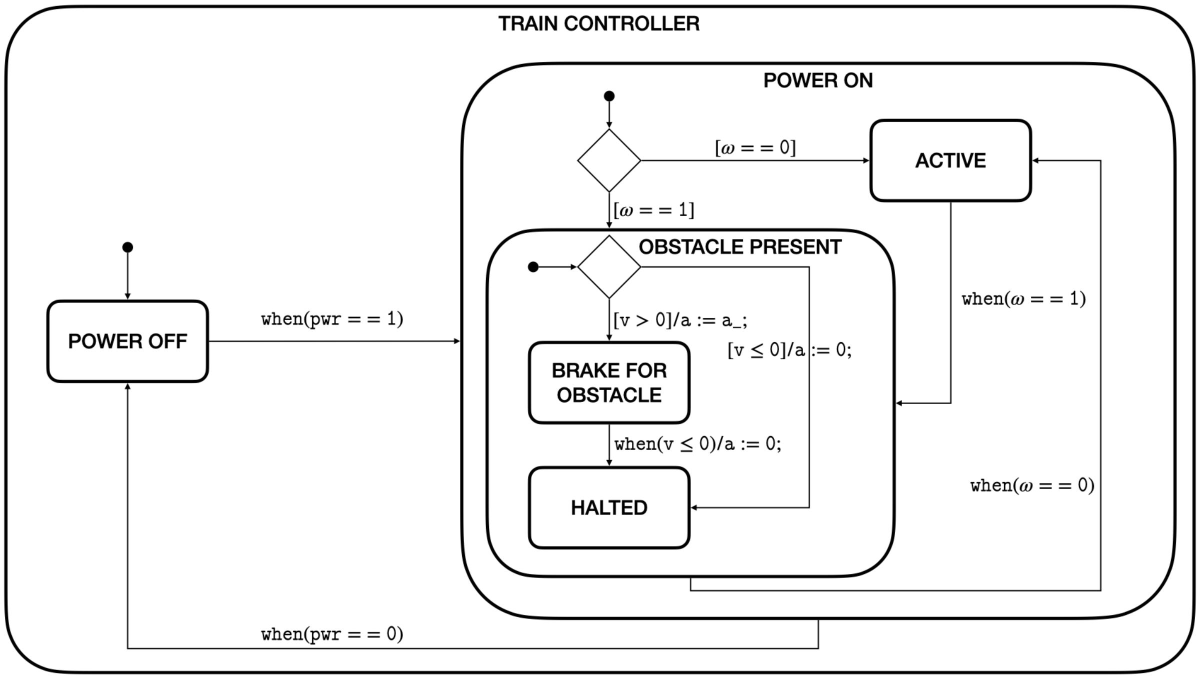

Figure 2: Train controller behaviour – top-level state machine.

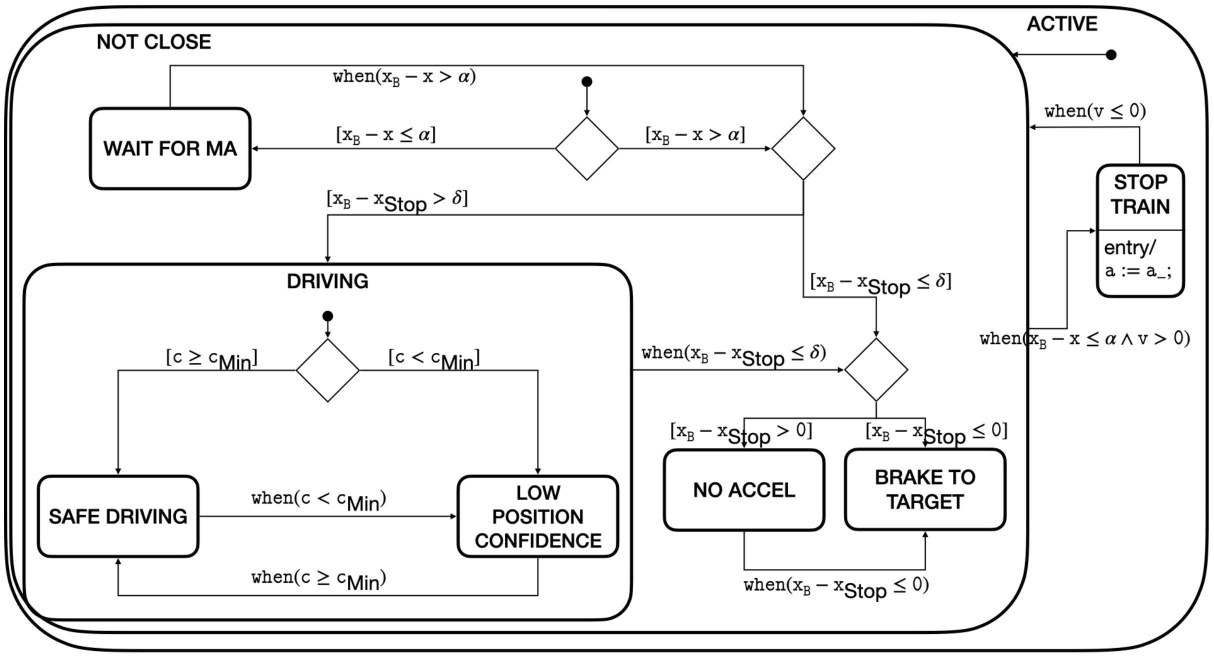

Figure 3: Train controller behaviour – active states.

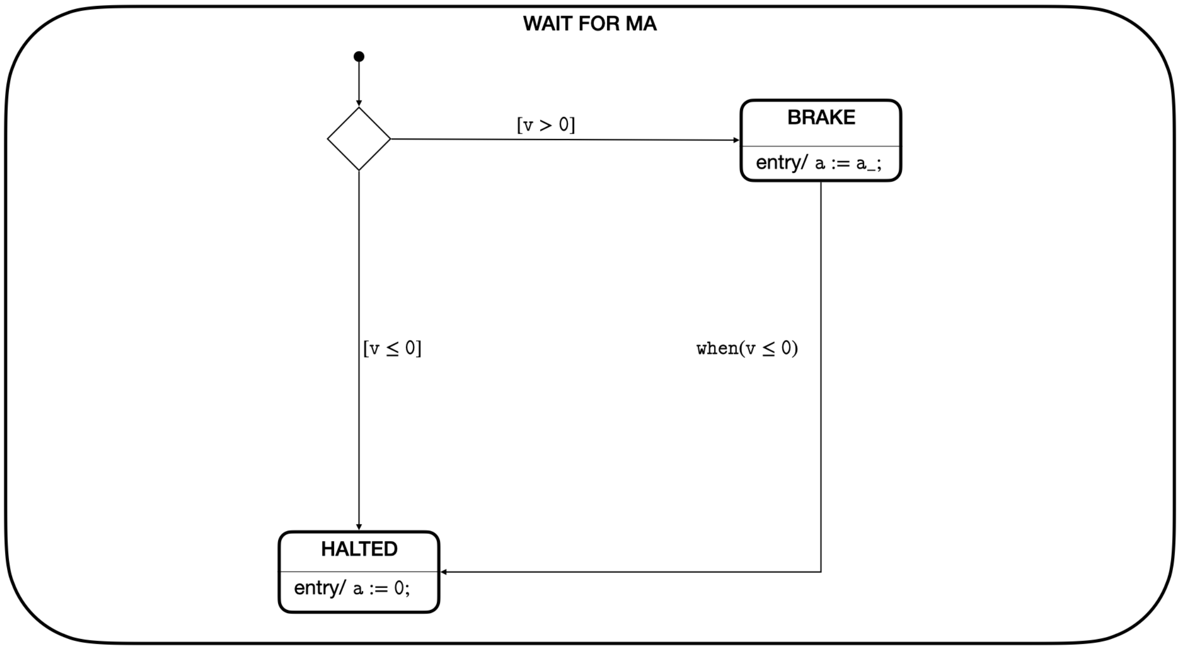

Figure 4: Train controller behaviour – sub-machine of state WAIT FOR MA.

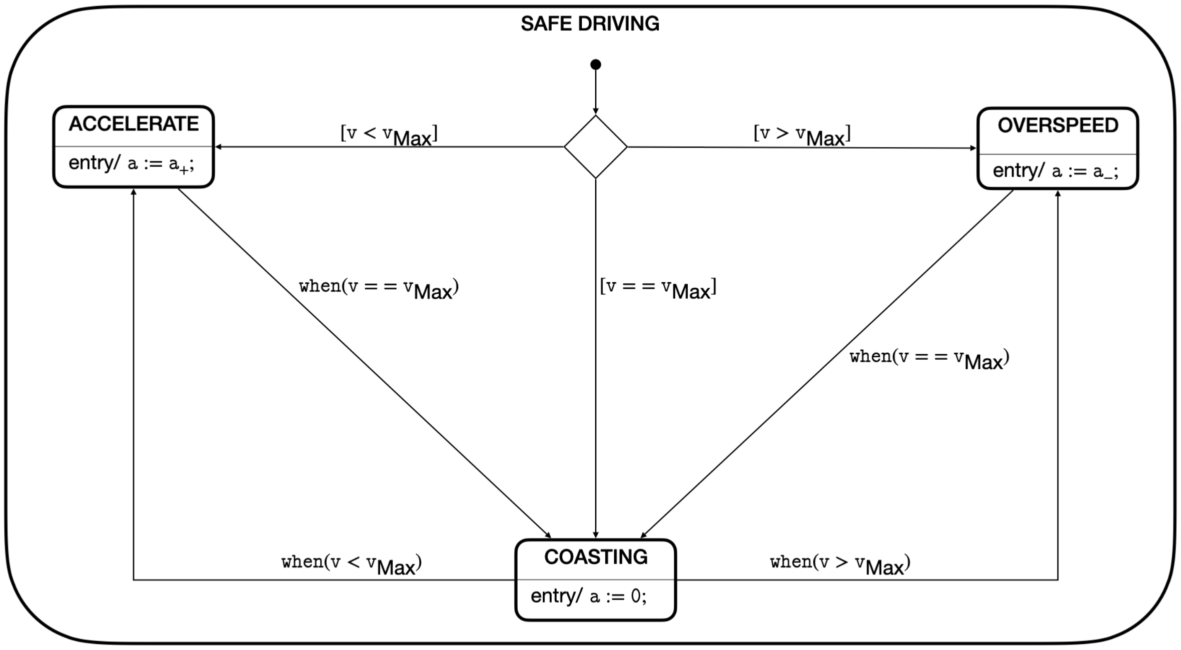

Figure 5: Train controller behaviour – safe driving state.

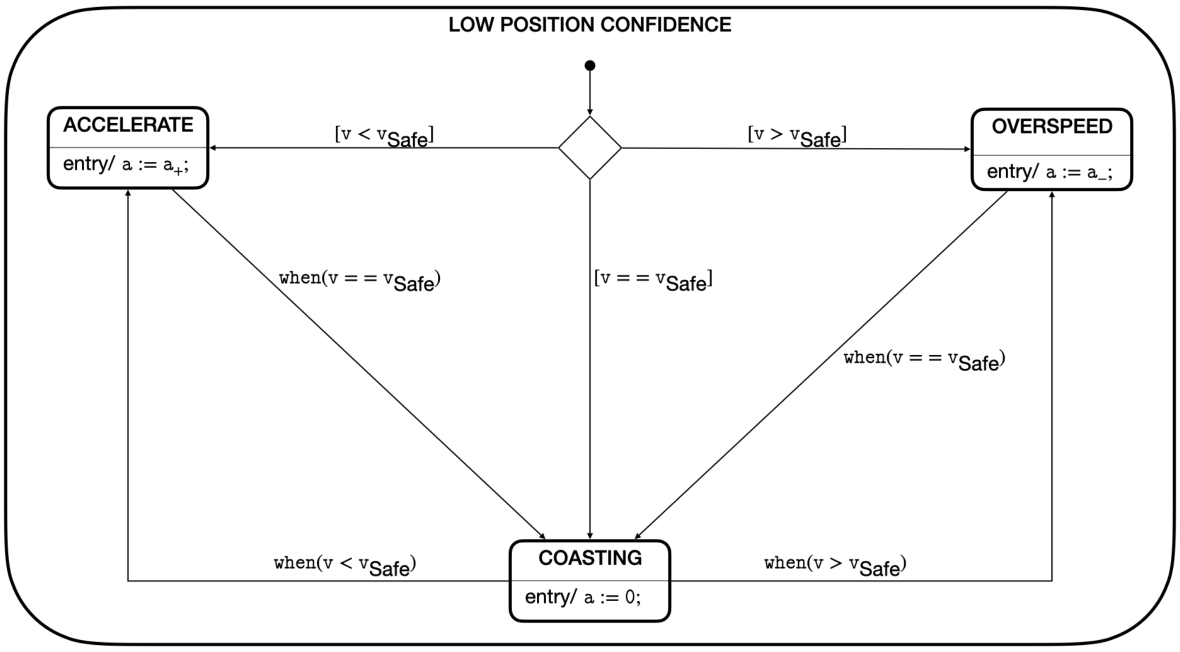

Figure 6: Train controller behaviour – driving with low position confidence.

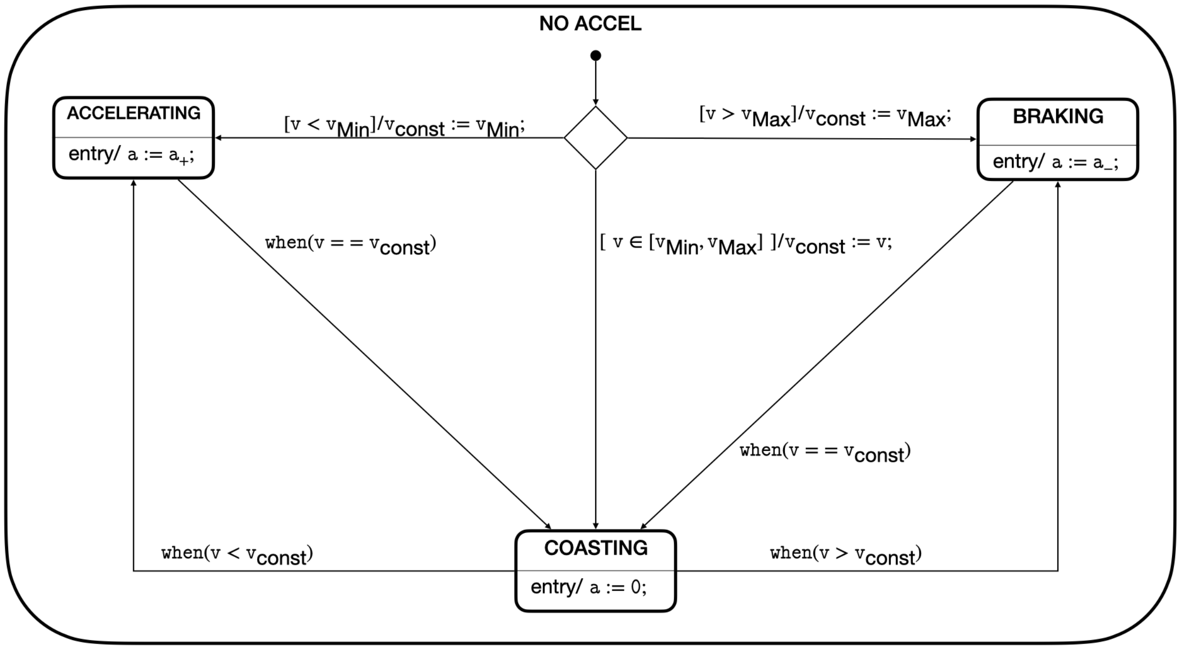

Figure 7: Train controller behaviour – sub-machine of state NO ACCEL.

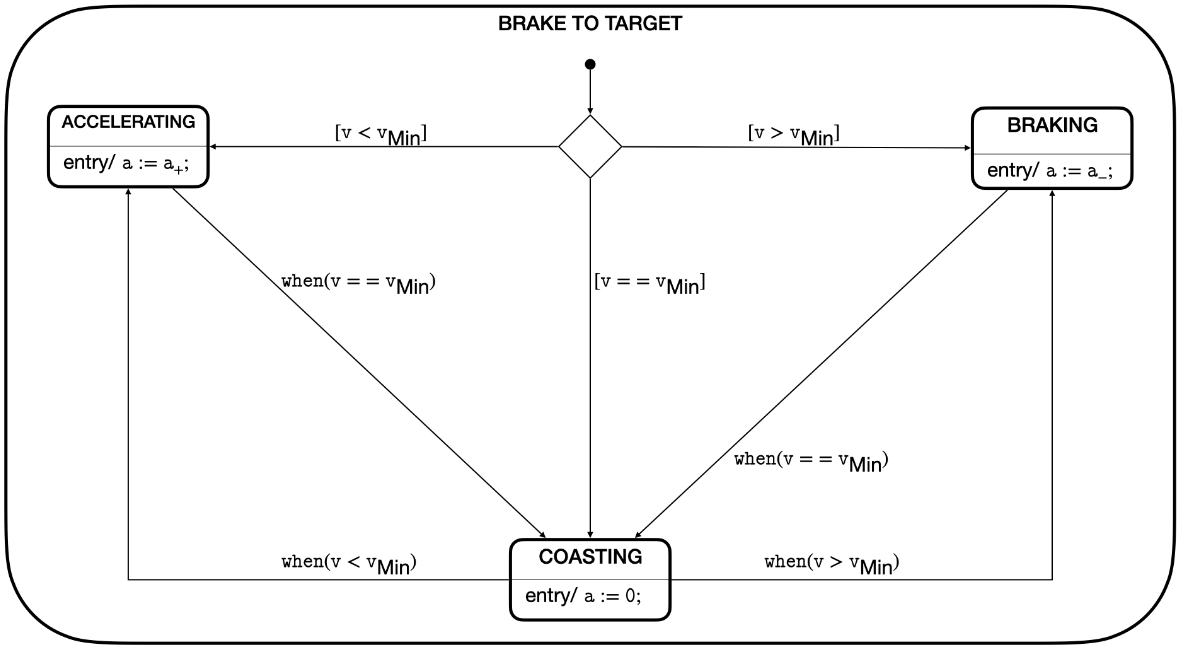

Figure 8: Train controller behaviour – braking to target destination.

3 Testing on Module Level – Complete Methods

Complete Test Suites and Formal Methods.

Since 2011, the standard RTCA DO-178C [49] which is applicable for the development and V&V of safety-critical avionic systems contains guidelines for the application of formal methods, these are described in the supplement RTCA DO-333 [47]. This supplement lists model checking as one accepted formal verification method among others (e.g. theorem proving). The complete testing strategies advocated in this technical report for module-level software verification can be regarded as model checking of code in two variants: (1) for *conformance checking*, *complete test suites* proving model conformance can be used. (2) For *property checking*, requirements for the given module need to be formalised (typically in a temporal logic, e.g. LTL). Then it can be verified again by means of complete test suites that the code fulfils the given formula, just as the reference model does. Completeness of a test suite asserts that every violation of model conformance or of the specified property will be uncovered, and that correct implementations will pass the test suite. Frequently, it suffices to apply *exhaustive* test suites. These guarantee to detect errors but may also reject correct implementations in certain situations.

In black-box testing, completeness or exhaustiveness can only be guaranteed under certain hypotheses, typically about the maximal number of states implemented in the SUT, and about the possible mutations of branching conditions and output expressions. For safety-critical systems development, however, software source code must be available during V&V for analysis. Therefore, the hypotheses about the SUT can be checked by means of various forms of static analysis. Consequently, complete or exhaustive test suites correspond to "real" code-level model checkers in the sense that the absence of failed test executions *proves* software correctness. Different from model checkers, the verification is performed by dynamic execution of the test cases against the code, and not by static code analysis methods.

The autonomous train controller example introduced in Section 2 has typical characteristics occurring in many ATV control applications: Inputs may be associated with conceptually infinite domains, such as real values for train position, but outputs are discrete, as the accelerate/coast/brake outputs to the train engine.[3] For this class of systems, complete test suites can be constructed in 4 steps [21, 22].

1. First the input equivalence classes are constructed.
2. In the second step, the system model is abstracted to a finite state machine (FSM) in Mealy style. The inputs of this FSM correspond to the input equivalence classes of the original system model, and the FSM outputs correspond to the discrete system outputs.
3. In step 3, a complete FSM test strategy is applied, resulting in a test suite with input sequences of equivalence class identifiers and expected outputs from the domain of system outputs.
4. In the last step, this FSM test suite is translated into a test suite for the original system, by selecting concrete representatives for each of the classes referenced in an FSM test case. The resulting test suite has the analogous completeness properties as the FSM test suite (this is the main result shown in [21, 22]).

Example 1. For the autonomous train controller, we consider two modules: the cyclic update machine providing new position and speed values on every Δt cycle and the proper controller exercising the acceleration commands to the engine according as specified in the SysML state machines shown in Fig. 2 and subsequent sub-machine models.

For the latter, the SysML state machine model is associated with a formal behavioural model by means of a translation to a flattened symbolic finite state machine (SFSM) [28, 42]. These can be considered as a simpler subset of the more general Kripke structures used in [21, 22] for the elaboration of the complete testing theory. The SFSM representation has the advantage that the input equivalence classes can be simply calculated by enumerating all positive and negated conjunctions of the SFSM guards, as long as these have a model. For the train controller module shown in Fig. 2 and its sub-machines, this results in 28 input equivalence classes $c_i,\ i = 1, \dots, 28$, of which we show several examples here:

$$\begin{aligned}
c_1 &\equiv \texttt{pwr} = 0 \\
c_3 &\equiv \texttt{pwr} = 1 \land \omega = 1 \land 0 < v \\
c_5 &\equiv \texttt{pwr} = 1 \land \omega = 0 \land x_B - x > \alpha \land x_B - x_{\text{Stop}} > \delta \land c \geq c_{\text{Min}} \land 0 < v < v_{\text{Min}} \\
c_9 &\equiv \texttt{pwr} = 1 \land \omega = 0 \land x_B - x > \alpha \land x_B - x_{\text{Stop}} > \delta \land c \geq c_{\text{Min}} \land v \geq v_{\text{Max}} \\
c_{27} &\equiv \texttt{pwr} = 1 \land \omega = 0 \land x_B - x \leq \alpha \land x_B - x_{\text{Stop}} \leq 0 \land v = 0
\end{aligned}$$

Applying the well-known complete H-Method [9] for proving equivalence of I/O behaviour between an FSM and an implementation, results in approx. 600 test cases[4], under the hypothesis that the number of internal control states of the implementation is less or equal to the number of states in the FSM reference model. Assuming that the source code of the implementation has been directly created from the SFSM justifies this hypothesis, and it can be verified by static analysis of the created code.

[3]Other example of systems of this kind are airbag controllers, thrust reversal controllers in aircraft, or safety controllers for robots interacting with humans.

[4]The number of test cases needed usually varies with the method implementation. We have used here the open source library `libfsmtest` for model-based testing with FSMs which is available on `https://bitbucket.org/JanPeleska/libfsmtest`.

As an example of the test cases on FSM level, consider the following case consisting of two steps.

$$(c_{27}/\mathtt{a} := \mathtt{0}).(c_5/\mathtt{a} := \mathtt{a}_+)$$

Translation of this FSM test case to a concrete test case against the target system requires to select representatives from input classes c_{27} and c_5 that have been specified above. An example for such a concrete test case is

$$(\mathtt{pwr} = 1 \wedge \omega = 0 \wedge x_B = 0 \wedge x = 0 \wedge x_{\text{Stop}} = 0 \wedge v = 0/\mathtt{a} := \mathtt{0}).$$
$$(\mathtt{pwr} = 1 \wedge \omega = 0 \wedge x_B = 10000 \wedge x = 0 x_{\text{Stop}} = 0 \wedge c = 0.95 \wedge v = 0.01/\mathtt{a} := \mathtt{a}_+)$$

In the first step, the train controller is powered, and no obstacle is present. The speed of the train is zero, and there is no movement authority available ($x_B = x = 0$). Thus it is expected that the requested acceleration is 0. In the second test step, a movement authority arrives ($x_B = 10000$), so it is expected that the train starts to accelerate.

The longest test cases of this complete test suite have only 4 steps: every state of the minimised FSM can be reached in at most 2 steps. From every state, every input needs to be exercised; this adds 1 further step to the test case. Then the target state reached needs to be distinguished from other states. This can be performed for the train controller by traces of length one. This adds up to test cases of maximal length 4. □

It has been criticised in the past that complete test suites turn out to be too big to be applied in practise to systems of realistic complexity. We agree that complete test suites should not be applied to system level testing. However, for software components of a complex system, these test suites turn out to be feasible from today's perspective, for the following reasons: (1) As shown in [23, 41], the construction of equivalence classes reduces the test suite size significantly without impairing the suite's completeness properties. (2) For the application of complete test suites as described here, we need not formulate hypotheses about potential additional control states in the implemented software, because the number of these states can be extracted from the code by static analysis. Consequently, the test suite size only depends polynomially on the number of states [8] (the size would grow exponentially in the potential number of *additional* control states present in the implementation). (3) The execution of software tests can be performed fully automatic in the cloud, so that many test cases can be executed in parallel. For these reasons, the original criticism about complete test suites no longer applies today.

Summarising, the complete test suites advocated here are suitable for verifying the logical correctness of the source code, but they are not intended for receiving certification credits regarding the correctness of the integrated HW/SW system. In the approach advocated here, this integration aspect is shifted to the system test level.

Model-Conformance vs. Requirements Satisfaction – Traceability Issues

The supplement to RTCA DO-178C on model-based development and V&V states [48, MB.6.7]: *"Model coverage analysis does not eliminate the need for traceability analysis between requirements from which the model was developed and the model."*. This has an implication on the model-based testing approach which – to the best of our knowledge – has not received sufficient attention in the research communities: traceability demands that requirements are related to the model portions realising these requirements. Then the code has to be related to the model parts it implements. Finally, the test cases checking the code need to trace back to the requirements they intend to verify.[5] It does *not* suffice to argue that the

[5]This can be done either indirectly through the traceability chain *test case* $\longrightarrow$ *code* $\longrightarrow$ *model* $\longrightarrow$ *requirement* or directly from test case to requirement.

formal model (in our case, the SysML state machines) has been reviewed (or even formally verified) and found to reflect all requirements correctly and then conclude that, since the code has been shown by complete test suites to be equivalent to the model, it must implement all requirements "somehow" in the proper way.

For modelling with SysML, traceability between model elements and requirements can be directly represented by means of the so-called «satisfy» relationship linking behavioural or structural model elements to requirements [34]. It has been shown in [39] how this information can be exploited to create formal representations of requirements in a temporal logic like LTL. The atomic propositions of the LTL formulas are quantifier-free first-order expressions over model variables. The validity of a formula can be decided by abstracting model computations to the sequences of sets containing the formula's atomic propositions valid in the respective computation step [4, 3.2.2]. In [28] it has been shown that exhaustive test suites for given sets P of atomic propositions can be constructed that frequently require fewer test cases than needed for establishing full model equivalence. Exhaustiveness in this situation means that the test suite is guaranteed to fail for at least one test case, if the computations of the implementation, when abstracted to traces over sets of elements from P, contain traces that are not produced by the reference model. Therefore, passing the test suite implies that the implementation fulfils all LTL properties over atomic propositions from P that are also fulfilled by the reference model.

We conclude that also the requirements-based verification tasks imposed by the RTCA DO-178C standard can be handled by means of exhaustive test suites. We suggests that on module-level, two complete test suites should be executed: the first with objective to prove model equivalence enables us to calculate a meaningful test coverage measure for system tests, as explained in the next section. The second, requirements-driven test suite is essential to satisfy the traceability requirements of the standard.

4 System-Level Testing

Meaningful System Tests.

Besides checking the correctness of the overall system integration, a major objective of system testing is to verify the complete workflow of services and applications, potentially across several communicating controllers and the respective interfaces involved. Tests designed for this purpose are usually called *end-to-end (E2E) tests*. Analysing the module tests discussed in the previous section, it becomes apparent that they do *not* represent meaningful end-to-end tests.

Example 2. Consider one of the module tests of maximal length 4 from the complete test suite discussed in Section 3. We present here the symbolic version with input equivalence classes, and not the concrete version, where atomic propositions have been resolved by selecting concrete values.

$$\begin{aligned}
&(\texttt{pwr} = 1 \land \omega = 0 \land x_B - x > \alpha \land x_B - x_{\text{Stop}} > \delta \land c \geq c_{\text{Min}} \land v = 0/\texttt{a} := \texttt{a}_+).\\
&\qquad(\texttt{pwr} = 1 \land \omega = 0 \land x_B - x \leq \alpha \land 0 < v/\texttt{a} := \texttt{a}_-).\\
&\qquad(\texttt{pwr} = 1 \land \omega = 0 \land x_B - x > \alpha \land x_B - x_{\text{Stop}} > \delta \land c \geq c_{\text{Min}} \land v = 0/\texttt{a} := \texttt{a}_+).\\
&\qquad(\texttt{pwr} = 1 \land \omega = 0 \land x_B - x > \alpha \land x_B - x_{\text{Stop}} > \delta \land c \geq c_{\text{Min}} \land v == v_{\text{Min}}/\texttt{a} := \texttt{a}_+)
\end{aligned}$$

The first test step initiates a power up in a situation where movement authority is already available ($x_B - x > \alpha$). The train is expected to accelerate. The controller transits into sub-states of SAFE DRIVING, because the confidence value c is high enough. Step 2, however, initiates a robustness transition from SAFE DRIVING to STOP TRAIN in the controller, because the train location appears to be close to the destination ($x_B - x \leq \alpha$), without having first fulfilled condition $x_B - x_{\text{Stop}} \leq \delta$ which would be expected in a “physically reasonable” execution. The train is expected to brake. While this transition is contained in the controller model to ensure its robustness, its occurrence is highly unlikely, so that it would not be selected for a “realistic” E2E test, but only for a robustness test. It would suffice, however, to check this aspect of robustness at module level. In Step 3, a situation is described again where the train is further away from its destination – this expressed by conditions $x_B - x > \alpha \wedge x_B - x_{\text{Stop}} > \delta$. If we just selected a smaller value of x to fulfil these conditions, this would again result in a robustness transition: the position sensors provide an improved position value, while the previous one for Step 2 was too high. Alternatively, this step could be realised by increasing the value of x_B. This would correspond to a realistic system test step, where a new movement authority is obtained by increasing the destination value x_B. Obviously, the distinction between a robustness test step and a normal behaviour test step suitable for an E2E test cannot be made on the basis of the equivalence class formulas alone. While this distinction is not relevant for module testing, it has high significance for the design of meaningful E2E tests. In the last step, the input is just updated according to the expected change of velocity: if the speed was zero before and the train keeps accelerating, the velocity will increase to v_{Min}.

The test case now ends while the train is still accelerating. For a suitable E2E test, we would expect a continuation with further test steps, leading the train to its destination x_B. Such continuations, however, do not exist in the module test suite, because every test case there starts from the initial controller state by powering the system.

Obviously, the physical model connecting acceleration, velocity, and position (see Section 2) needs to be considered for constructing meaningful system tests in a simulation environment: Step 3, for example, can only be executed when the train has come to a standstill ($v = 0$), and the point in time when this happens depends on the effect of the negative acceleration a_- and the actual speed value when the brakes had been triggered in Step 2.

For system tests on target hardware and in the real operational environment, these simulations of the physical world are unnecessary, but the trigger conditions for each new test step need to be monitored. Here, Step 3 can only be triggered *after having observed* that the train has come to a standstill. Then the new value of x_B can be provided to the SUT, so that $x_B - x > \alpha \wedge x_B - x_{\text{Stop}} > \delta$ is fulfilled. Moreover, we observe that system testing with the target system and its operational environment needs to be *intrusive*, if different confidence values of the position sensors and erroneous position values need to be tested. ‘Intrusive’ means that the behaviour of the position sensors can be changed for test purposes, for example, by corrupting position values and lowering the actual confidence values calculated by the sensors. Finally, the occurrence and removal of obstacles has to be controlled (for example, remotely controlled vehicles could be moved on the track and removed at a later point in time). □

Summarising the findings highlighted by this example, we conclude that

1. Module test suites are not well-suited to be turned into meaningful E2E tests.
2. The formula representations of input equivalence classes do not provide sufficient information about which concrete solutions would give meaningful test data for E2E testing.
3. Test environments for the simulated execution of system tests require components observing the applicable physical laws during test execution.
4. System tests with the “real” target in its operational environment require intrusive test equipment for some input interfaces to the SUT.

Finding 1 can be justified more formally. When testing for model conformance, the complete strategies applied usually strive to reach every state of the SUT on the shortest path possible, and then to execute every possible input from this state. Consequently, breadth-first-search algorithms are applied, and transitions that should only be performed in exceptional cases from the users' perspective are frequently taken if they reside on the shortest path to such a state. Moreover, the objective to exercise every possible input (or input class) in a given state necessarily leads to frequent resets, leading to the start of a new test case. In contrast to this, end-to-end tests need to be identified by a depth-first search through a system model; this leads to longer test cases starting and ending in system states that are meaningful from the perspective of the tested application.

Example 3. For the train controller example, meaningful end-to-end tests would start at the initial location x_A and end at the destination x_B. If a reaction to an obstacle detected during this journey is checked during an E2E test, the test should not only check the proper braking procedure, but also the continuation of the journey at normal speed after removal of the obstacle which is indicated by ω falling back to zero. □

As a consequence of these observations, we see that it is generally impossible to "glue several module tests together and come out with a meaningful end-to-end test". We further observe that the optimal error detection capabilities of complete test suites are unrelated to the end-to-end test quality.

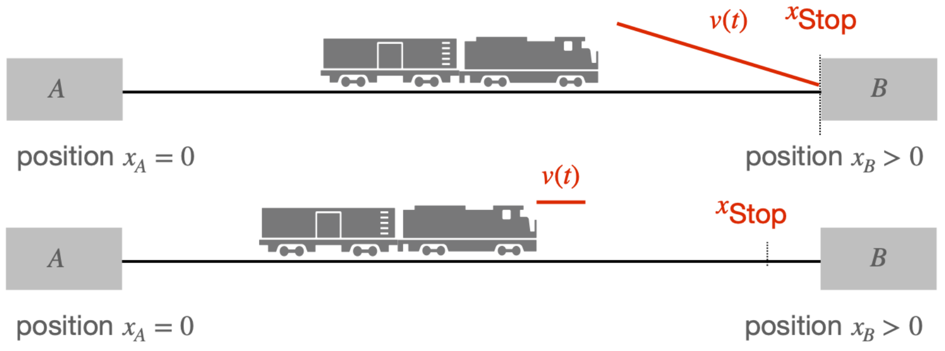

Figure 9: System test track for autonomous train.

System Test Scenarios.

System tests of autonomous trains need to be executed in different *scenarios*, where each scenario is specified by a route through the rail network. This concept is similar to scenario-based testing of autonomous road vehicles [24], but considerably simpler, since the train movements are restricted by the rail network, and the absence of collision hazards involving other trains is already ensured by the interlocking system. One of these scenarios is sketched in Fig. 9, where the train has to travel along a pre-defined route, starting at station A and ending at station B. Along this route, different speed restrictions (v_{Max}) may apply, and obstacles may be present and subsequently removed, either after the train has come to a standstill, or while the train is still approaching the obstacle.

Other scenarios might involve stops at railroad crossings or multiple stops at different destinations. Note that the comprehensiveness of the system test scenario catalogue to be checked is outside the scope of this technical report, but has been investigated, for example, in [14].

To allow for automated test case derivation for E2E system tests, we introduce a data structure called *symbolic scenario test tree (SSTT)*. These trees are interpreted as a *hybrid systems* [16] which are simplified in the sense that our outputs are always concrete assignments instead of being implicitly specified through jump conditions. The use of these trees is exemplified with the tree fragment shown in Fig. 10.

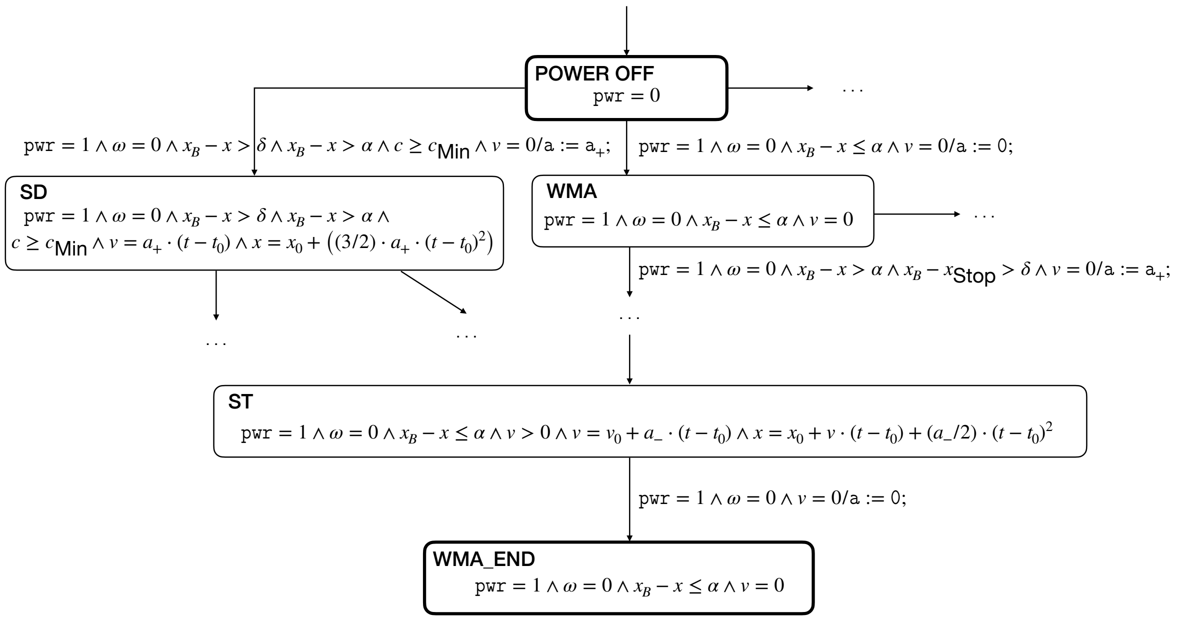
POWER OFF
pwr = 0
pwr = 1 ∧ ω = 0 ∧ x_B − x > δ ∧ x_B − x > α ∧ c ≥ c_Min ∧ v = 0/a := a_+;
pwr = 1 ∧ ω = 0 ∧ x_B − x ≤ α ∧ v = 0/a := 0;
SD
pwr = 1 ∧ ω = 0 ∧ x_B − x > δ ∧ x_B − x > α ∧ c ≥ c_Min ∧ v = a_+ · (t − t_0) ∧ x = x_0 + ((3/2) · a_+ · (t − t_0)^2)
WMA
pwr = 1 ∧ ω = 0 ∧ x_B − x ≤ α ∧ v = 0
pwr = 1 ∧ ω = 0 ∧ x_B − x > α ∧ x_B − x_Stop > δ ∧ v = 0/a := a_+;
ST
pwr = 1 ∧ ω = 0 ∧ x_B − x ≤ α ∧ v > 0 ∧ v = v_0 + a_− · (t − t_0) ∧ x = x_0 + v · (t − t_0) + (a_−/2) · (t − t_0)^2
pwr = 1 ∧ ω = 0 ∧ v = 0/a := 0;
WMA_END
pwr = 1 ∧ ω = 0 ∧ x_B − x ≤ α ∧ v = 0

Figure 10: Partial representation of a symbolic scenario test tree (SSTT) for the autonomous train control system.

Just as the control modes of hybrid system models, the nodes of an SSTT represent invariants over both discrete and time-continuous variables. The edges are labelled with guard conditions and output assignments, as in symbolic finite state machines. In Fig. 10, the root of the tree is represented by node POWER OFF, which only has the invariant $\texttt{pwr} = 0$. From there, edges lead to different child nodes, depending on the guard conditions labelling these edges. The edge to node WMA is taken when the power is switched on, no obstacle is present, the train is without movement authority ($x_B - x \leq \alpha$), and it is not moving. Therefore, the acceleration can be set to 0, and the system resides in state WMA until a movement authority arrives. There are several situations to distinguish: the movement authority might arrive with an obstacle present, or it might arrive in a position that is already quite close to the destination. The downward edge shown in Fig. 10 specifies the situation where no obstacle is present, and it is not yet required to brake for the target destination ($x_B - x_{\text{Stop}} > \delta$). The edge from node POWER OFF to node SD specifies the situation where the train is powered, while a movement authority is already available, so that the train is allowed to accelerate, directly leaving its standstill position. If the invariants need to refer to variable values at node entry, these are denoted by $< \texttt{variable} >_0$. In node SD, for example, $(t - t_0)$ denotes the time which has passed since entering the node. Towards the leaves of the tree, the nodes describe the situations where the train approaches the destination x_B and brakes until it comes to a standstill. This is exemplified here by nodes ST ("stop train") and WMA_END ("wait for next movement authority").

As can be seen in this example, the SSTT contains the necessary information about physical laws to be applied. These were missing in the module test models. Moreover, the SSTT is created in such a way that robustness transition sequences that are better tested on module level are left out, and that the leaves of the tree represent suitable target states from the perspective of E2E testing. The example also indicates that these trees can become quite large. In part, they can be automatically derived from a hybrid system model, but the information about sequences of transitions that are only taken in robustness situations need to be provided by system experts. Moreover, the suitable termination states for E2E testing need to be manually provided. In [39], methods for automated elaboration of requirements formulas from SysML models have been presented. With these (LTL-) formulas at hand, it is at least possible to check automatically whether the SSTT covers a certain requirement. This, however, does not yet imply that the respective path through the tree also represents a meaningful E2E test. For these reasons, we expect that SSTT creation can only be partially automated and will always require inputs from human experts.

Cloud-based Tests vs. Tests on Target System – System Test Coverage.

Cloud-based testing significantly increases the availability of hardware resources for test campaigns, and improves the possibilities to extend or shrink these resources according to the demands of such campaigns. It has to be discussed, however, how certification credit may be obtained for tests performed in the cloud, since cloud platforms do not allow for execution of SUTs with their original hardware. The standard supplement RTCA DO-333 regulating the use of formal methods [47, FM.6.3.1.c] states that formal verification results can also verify aspects of compatibility of software and hardware, if the underlying formal models also cover the hardware platform. For specific verification objectives like worst-case execution time analysis, this possibility has already been successfully exploited [51]. For system testing in the cloud, more general hardware models are needed, so that control applications can be executed on these models with their original machine code and address maps, as used on the real target. This type of models has recently been developed; they are called *virtual prototypes* [17] and represent more advanced variants of the simulators that have been used for quite some time. We expect that certification credit for cloud-based system tests or HW/SW integration tests can be achieved if the software is executed in such a variant of virtual prototypes. This option, however, requires a new version of the standards and would certainly require *tool qualification* [50] for the virtual prototypes, to justify that they really represent a faithful model of the target HW.

For good reasons, we can expect that also in the future at least some tests will be required to be performed on the “real” target system with its integrated hardware and software: RTCA DO-333 states [47, FM.6.7] *“Tests executed in target hardware are always required to ensure that the software in the target computer will satisfy the high-level requirements . . . ”*. We expect that system-level tests performed on the target system will be regarded as more valuable than module tests on target HW, since system tests exercise more aspects of drivers and firmware than module tests that often only cover software interfaces and do not stimulate drivers and firmware at all. Since the current standard ISO 26262 applicable in the automotive domain refers to RTCA DO-178 when suggesting V&V methods, we expect that the verification approaches accepted for the avionic domain will also remain admissible for the automotive domain in the future. In the railway domain, the applicable standard EN 50128 [6] also regulates (even requires) the use of formal methods, but is less explicit than RTCA DO-178C with respect to formal hardware models and the amount of testing to be performed on the target system. With these standard-related facts in mind, we conclude that the most specific guidelines for a formal approach to testing in simulated and in target environments are given by the avionic standards.

According to these standards, system requirements shall all be covered by system-level tests. As discussed in Section 3, requirements can be transformed into LTL formulas, and the symbolic scenario test tree introduced above contains sufficient information about valid atomic propositions along each path to decide with an automated procedure which formulas are fulfilled on each path.[6] Uncovered requirements may both lead to new branches to be added into a scenario test tree and to the creation of new scenarios, if the existing ones cannot accommodate paths where a given formula can be checked non-vacuously.

The new aspect not covered by today's standards is the fact that a major portion of the system tests will be executed in the cloud, and not on the target with its operational environment. We expect that future revisions of the standards will demand that "a reasonable portion" of these tests should be executed on the target. Then the question remains to be answered is which sub-collection of system tests represents such a reasonable portion. When the approach advocated in this technical report is followed, a sound answer to this question can be given: system testing on the target should cover the normal behaviour transitions of the SFSM models used for module-level testing. This is justified since we have shown on that the software conforms to its SFSM models, so SFSM transition coverage implies software transition coverage. Consequently, transition coverage of SFSMs implies that all relevant transitions of the software have been exercised *on the target system*. This is a very comprehensive check of the HW/SW integration correctness. We suggest to exempt robustness transitions from this coverage requirement, since many robustness aspects can be better tested on module level, avoiding too many variants of intrusive tests on system level.

The coordinator test agent can monitor the transition coverage achieved so far and delegate the respective test steps suitable for covering missing transitions to the agents executing tests on the target and its operation environment. Note that this process requires constraint solving, since the system input sequences and timing conditions need to be calculated that are suitable to cover a given transition in a given module. This however, has been solved in modern model-based testing tools, such as [30, 39].

Agent-based Approach and Online Testing.

The preceding discussion of system tests suggests a testing environment where multiple concurrent components are deployed that are responsible for the different aspects identified above.

1. Coverage analysis and coordination of concurrent test executions.
2. Control of an individual system test execution.
3. Simulation of the physical environment in real time or simulated time (for cloud-based system tests).
4. Interfacing to the target system in its operational environment (transmission of movement authorities, intrusive influence of the position sensors, and obstacle creation and removal).

For realising these testing environments, we apply agent-based system testing (ABST). To this end, a multi-agent system (MAS) is realised according to the paradigm of *mixed initiative control*. As stated in [13], this form of agent collaboration is characterised by "allowing a supervisor and group of searchers to jointly decide the correct level of autonomy for a given situation". In [13], this paradigm was intended to regulate the collaboration between multiple robots and a human operator. In our context, test agents collaborate without any human interaction, but some agents have primarily coordination tasks, while others focus on optimising the execution of individual system tests. It has been shown already in [7],

[6] Note that this procedure requires a *vacuity analysis* [5]: a requirement of the form $\mathbf{G}(\varphi \Rightarrow \psi)$ may be satisfied on a path π of the tree just because φ never becomes true. In this case, the path π would not be a suitable witness for testing this requirement.

how MAS technology helps to increase the test strength of system test cases by letting agents execute strategies for selecting test data which is most effective in the current situation of a system test execution.

In [30], the authors introduced the term *online testing* (or *on-the-fly testing*) for model-based tests combining the generation of test steps of a test case with the actual execution and checking of expected results.[7] Online testing is very well-suited for the objectives outlined in this technical report, because in the presence of potentially millions of system test cases to be executed, a separate test generation phase preceding the test execution might take too much time, whereas online testing delivers information about passed and failed test steps right from the start.

5 Related Work

The existence of complete test suites, that is, test suites with guaranteed error detection capabilities, has been first observed in the fields of process algebras [15] and finite state machines [8]. This research area is still very active: focal points concern complete test strategies for new types of conformance relations [20], and the reduction of test suite sizes while preserving completeness properties [40, 45].

Agent-based systems are frequently applied for developing autonomous systems or sub-systems thereof [36, 44, 52]. Since the end of the last century, a parallel line of research has received increasing attention [29]: the use of agent-based testing strategies and testing environments, with special emphasis on testing (potentially also agent-based) autonomous systems.

This new line of research has been motivated for example in [33], where it is stated that *"The specific nature of software agents, which are designed to be distributed, autonomous, and deliberative makes it difficult to apply existing software testing techniques to them"*. The recent study [29] confirms the growing interest in ABST, but it covers only the software testing aspect, without considering system testing of cyber-physical, potentially autonomous systems. Also with a focus on software testing (in particular, regression testing), the authors of [10] point out that static test scripts with "hard-coded" checks of expected results have insufficient flexibility in presence of changing software. They show how agent-based testing environments facilitate the update of test purposes (including the check of expected results). It should be pointed out that this problem has already been investigated very early in the field of process algebras, where test cases were already considered as (one or several) concurrent processes interacting with the SUT [15] (a more recent result introducing complete test suites for the CSP process algebra has been published in [40]). In the field of model-based testing against finite states machines, the term *adaptive tests* has been introduced for test cases that are able to react on different, possibly nondeterministic, SUT behaviours in a flexible way [18]. The results of [10], however, carry these existing ideas further, by (a) suggesting to equip test agents with learning capabilities, to adapt, for example, test oracles to intentional changes of the software under test, and (b) add prioritisation capabilities to the test agents, so that the most critical test cases are executed first. The approach from [10] significantly differs from the approach elaborated here in that the former do not consider tests of autonomous HW/SW systems and that they use the adaptive autonomy paradigm for inter-agent collaboration. This paradigm would not be suitable for the objectives described here. In [43], coordinator agents and agents acting as *runners* have been introduced. This is similar to the two categories of test agents discussed in this technical report. Our system test execution agents, however, have a higher degree of autonomy than the runners discussed there.

[7]This concept has already been used in 1996 in the VVT-RT Tool [37] whose commercial version is called RT-Tester today `www.verified.de`.

In [32], software system tests are investigated from the test management perspective. Similar to [10] the authors emphasise the importance of prioritisation in presence of very large test suites allowing only for sub-sets of test cases to be executed during a test campaign. They suggest to derive the priority from the importance of the SUT component (called 'test unit' in [32]) covered by the test case and several other factors. We have adopted these ideas here to distinguish transitions of different criticality in the reference model.

The authors of [12] explicitly address the problem of system testing, also advocating a MAS approach. The main focus is on testing distributed SUTs, and the authors suggest a design pattern for a distributed MAS interacting with the SUT. More formalised work on testing distributed systems has been published, for example, in [19, 31]. In our work presented in this technical report, we could neglect the distribution effects, since the SUT consists of an autonomous train controller with its on-board sensors, actuators, and sub-ordinate systems. Extending this example, the problem of test distribution would arise if we had to design, for example, a system test of the IXL, its associated RBC, and several autonomous trains driving across the rail network controlled by IXL and RBC. This is beyond the scope of this technical report.

For testing integrated autonomous HW/SW systems or simulations thereof, applications in the automotive and robotics domains have been published in [1, 2, 3, 7]. This work demonstrates the capability of agent-based testing environments to increase the test strength of complex system test suites. It has inspired the new contributions presented in this technical report.

Summarising, none of the related work addresses (1) the main focus of this technical report, which is the combined module-level and system-level testing of complex autonomous systems, observing existing and potential future requirements of applicable standards, (2) the possibility to achieve higher test coverage by adaptively adding resources in the cloud (CPU, memory, system test execution containers), (3) the advantages of a model-based approach, or (4) the coordinated combination of complete module-level tests with system test achieving a justifiable degree of test coverage.

6 Conclusion and Future Work

In this technical report, we have proposed a solution for efficient module testing and system testing of autonomous transportation systems (vehicles, trains, aircrafts). Solutions of this kind are of considerable importance, because the number of tests to be performed on autonomous systems is so much higher in comparison to test suites for conventional systems. Therefore, these test campaigns cannot be performed in acceptable time with conventional methods, since the latter impose system tests to be executed with original equipment only. The key characteristics of the approach proposed here are (1) the use of complete test strategies on module level, (2) concurrent online systems tests executed on original equipment and in cloud-based simulation environments, and (3) the use of test agents to coordinate the system test effort. The consistency of this approach with applicable standards has been explained.

Two important aspects of testing autonomous systems were beyond the scope of this technical report. The first is the fact that many autonomous systems are so complex that they can no longer be represented by comprehensive models. Instead, they are specified in scenario libraries, and this leads to the questions of scenario completeness and consistency [14, 38]. The second aspect concerns the assurance to be provided for applications based on neural networks and multi-agent systems, because their behaviour cannot be specified and verified with conventional methods, and their behaviour may evolve over time, due to machine learning effects and on-the-fly modification or even creation of plans [46].

Future work will focus on the detailed evaluation of the approach advocated in this technical report.

At first, an in-depth analysis for the autonomous train control system presented here will be performed. Next, applications from the robotic domain (human-robot collaboration) and the avionic domain, such as autonomous taxiing, take-off, and landing or formation flight with the objective of fuel saving will be investigated. Future work will also cover distributed autonomous systems under test, based on existing results such as [19, 31].

References

[1] Dejanira Araiza-Illan, Anthony G. Pipe & Kerstin Eder (2016): *Intelligent Agent-Based Stimulation for Testing Robotic Software in Human-Robot Interactions*. In: *Proceedings of the 3rd Workshop on Model-Driven Robot Software Engineering*, MORSE '16, Association for Computing Machinery, New York, NY, USA, p. 9–16, doi:10.1145/3022099.3022101. Available at `https://doi.org/10.1145/3022099.3022101`.

[2] Dejanira Araiza-Illan, Anthony G. Pipe & Kerstin Eder (2016): *Model-based Test Generation for Robotic Software: Automata versus Belief-Desire-Intention Agents*. *CoRR* abs/1609.08439. Available at `http://arxiv.org/abs/1609.08439`. _eprint: 1609.08439.

[3] Dejanira Araiza-Illan, Tony Pipe & Kerstin Eder (2016): *Model-Based Testing, Using Belief-Desire-Intentions Agents, of Control Code for Robots in Collaborative Human-Robot Interactions*. *CoRR* abs/1603.00656. Available at `http://arxiv.org/abs/1603.00656`.

[4] Christel Baier & Joost-Pieter Katoen (2008): *Principles of model checking*. MIT Press.

[5] Thomas Ball & Orna Kupferman (2008): *Vacuity in Testing*. In Bernhard Beckert & Reiner Hähnle, editors: *Tests and Proofs*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 4–17.

[6] CENELEC (2011): *EN 50128:2011 Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems*.

[7] Greg Chance, Abanoub Ghobrial, Séverin Lemaignan, Tony Pipe & Kerstin Eder (2020): *An Agency-Directed Approach to Test Generation for Simulation-based Autonomous Vehicle Verification*. In: *IEEE International Conference On Artificial Intelligence Testing, AITest 2020, Oxford, UK, August 3-6, 2020*, IEEE, pp. 31–38, doi:10.1109/AITEST49225.2020.00012. Available at `https://doi.org/10.1109/AITEST49225.2020.00012`.

[8] Tsun S. Chow (1978): *Testing Software Design Modeled by Finite-State Machines*. *IEEE Transactions on Software Engineering* SE-4(3), pp. 178–186.

[9] Khaled El-Fakih, Rita Dorofeeva, Nina Yevtushenko & Gregor von Bochmann (2012): *FSM-based testing from user defined faults adapted to incremental and mutation testing*. *Program. Comput. Softw.* 38(4), pp. 201–209, doi:10.1134/S0361768812040019. Available at `https://doi.org/10.1134/S0361768812040019`.

[10] Eduard Enoiu & Mirgita Frasheri (2019): *Test Agents: The Next Generation of Test Cases*. In: *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 305–308, doi:10.1109/ICSTW.2019.00070.

[11] European Railway Agency (2012): *ERTMS – System Requirements Specification – UNISIG SUBSET-026*. Available under http://www.era.europa.eu/Document-Register/Pages/Set-2-System-Requirements-Specification.aspx.

[12] Giovanni Farias, Ayla Dantas, Raquel Lopes & Dalton Guerrero (2012): *Distributed Test Agents: A Pattern for the Development of Automatic System Tests for Distributed Applications*. In: *Proceedings of the 9th Latin-American Conference on Pattern Languages of Programming*, SugarLoafPLoP '12, Association for Computing Machinery, New York, NY, USA, pp. 1–11, doi:10.1145/2591028.2600817. Available at `https://doi.org/10.1145/2591028.2600817`.

[13] Benjamin Hardin & Michael A. Goodrich (2009): *On Using Mixed-Initiative Control: A Perspective for Managing Large-Scale Robotic Teams*. In: *Proceedings of the 4th ACM/IEEE International Conference on Human Robot Interaction*, HRI '09, Association for Computing Machinery, New York, NY, USA, p. 165–172, doi:10.1145/1514095.1514126. Available at `https://doi.org/10.1145/1514095.1514126`.

[14] Florian Hauer, Tabea Schmidt, Bernd Holzmüller & Alexander Pretschner (2019): *Did We Test All Scenarios for Automated and Autonomous Driving Systems?* In: *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019, Auckland, New Zealand, October 27-30, 2019*, IEEE, pp. 2950–2955, doi:10.1109/ITSC.2019.8917326. Available at `https://doi.org/10.1109/ITSC.2019.8917326`.

[15] Matthew Hennessy (1988): *Algebraic Theory of Processes*. MIT Press, Cambridge, MA, USA.

[16] T.A. Henzinger (1996): *The theory of hybrid automata*. In: *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, IEEE Computer Society Press, pp. 278–292.

[17] Vladimir Herdt, Daniel Große, Pascal Pieper & Rolf Drechsler (2020): *RISC-V based virtual prototype: An extensible and configurable platform for the system-level*. *J. Syst. Archit.* 109, p. 101756, doi:10.1016/j.sysarc.2020.101756. Available at `https://doi.org/10.1016/j.sysarc.2020.101756`.

[18] Robert M. Hierons (2004): *Testing from a Nondeterministic Finite State Machine Using Adaptive State Counting*. *IEEE Trans. Computers* 53(10), pp. 1330–1342, doi:10.1109/TC.2004.85. Available at `http://doi.ieeecomputersociety.org/10.1109/TC.2004.85`.

[19] Robert M. Hierons (2016): *A More Precise Implementation Relation for Distributed Testing*. *Comput. J.* 59(1), pp. 33–46, doi:10.1093/comjnl/bxv057. Available at `https://doi.org/10.1093/comjnl/bxv057`.

[20] Robert M. Hierons (2019): *FSM quasi-equivalence testing via reduction and observing absences*. *Sci. Comput. Program.* 177, pp. 1–18, doi:10.1016/j.scico.2019.03.004. Available at `https://doi.org/10.1016/j.scico.2019.03.004`.

[21] Wen-ling Huang & Jan Peleska (2016): *Complete model-based equivalence class testing*. *Software Tools for Technology Transfer* 18(3), pp. 265–283, doi:10.1007/s10009-014-0356-8. Available at `http://dx.doi.org/10.1007/s10009-014-0356-8`.

[22] Wen-ling Huang & Jan Peleska (2017): *Complete model-based equivalence class testing for nondeterministic systems*. *Formal Aspects of Computing* 29(2), pp. 335–364, doi:10.1007/s00165-016-0402-2. Available at `http://dx.doi.org/10.1007/s00165-016-0402-2`.

[23] Felix Hübner, Wen-ling Huang & Jan Peleska (2019): *Experimental evaluation of a novel equivalence class partition testing strategy*. *Software & Systems Modeling* 18(1), pp. 423–443, doi:10.1007/s10270-017-0595-8. Available at `https://doi.org/10.1007/s10270-017-0595-8`. Published online 2017.

[24] Hardi Hungar (2018): *Scenario-Based Validation of Automated Driving Systems*. In Tiziana Margaria & Bernhard Steffen, editors: *Leveraging Applications of Formal Methods, Verification and Validation. Distributed Systems*, Lecture Notes in Computer Science, Springer International Publishing, pp. 449–460.

[25] ISO (2021): *ISO/DIS 21448: Road vehicles — Safety of the intended functionality*. ICS: 43.040.10, Draft International Standard.

[26] ISO/DIS 26262-4 (2009): *Road vehicles – functional safety – Part 4: Product development: system level*. Technical Report, International Organization for Standardization.

[27] Nidhi Kalra & Susan M. Paddock (2016): *Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?* RAND Corporation, Santa Monica, CA, doi:10.7249/RR1478.

[28] Niklas Krafczyk & Jan Peleska (2021): *Exhaustive Property Oriented Model-based Testing With Symbolic Finite State Machines (Technical Report)*, doi:10.5281/zenodo.5151778. Available at `https://doi.org/10.5281/zenodo.5151778`. Funded by the Deutsche Forschungsgemeinschaft (DFG) – project number 407708394.

[29] Pavithra Perumal Kumaresen, Mirgita Frasheri & Eduard Paul Enoiu (2020): *Agent-Based Software Testing: A Definition and Systematic Mapping Study*. In: *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pp. 24–31, doi:10.1109/QRS-C51114.2020.00016.

[30] Kim Guldstrand Larsen, Marius Mikucionis & Brian Nielsen (2004): *Online Testing of Real-time Systems Using Uppaal*. In Jens Grabowski & Brian Nielsen, editors: *Formal Approaches to Software Testing, 4th International Workshop, FATES 2004, Linz, Austria, September 21, 2004, Revised Selected Papers*, *Lecture Notes in Computer Science* 3395, Springer, pp. 79–94, doi:10.1007/978-3-540-31848-4_6. Available at `https://doi.org/10.1007/978-3-540-31848-4_6`.

[31] Bruno Lima, João Pascoal Faria & Robert M. Hierons (2020): *Local Observability and Controllability Analysis and Enforcement in Distributed Testing With Time Constraints*. *IEEE Access* 8, pp. 167172–167191, doi:10.1109/ACCESS.2020.3021858. Available at `https://doi.org/10.1109/ACCESS.2020.3021858`.

[32] C. Malz & N. Jazdi (2010): *Agent-based test management for software system test*. In: *2010 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, 2, pp. 1–6, doi:10.1109/AQTR.2010.5520835.

[33] Cu D. Nguyen, Anna Perini, Carole Bernon, Juan Pavón & John Thangarajah (2011): *Testing in Multi-Agent Systems*. In Marie-Pierre Gleizes & Jorge J. Gomez-Sanz, editors: *Agent-Oriented Software Engineering X*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 180–190.

[34] Object Management Group (2015): *OMG Systems Modeling Language (OMG SysML), Version 1.4*. Technical Report, Object Management Group. Http://www.omg.org/spec/SysML/1.4.

[35] Object Management Group (2017): *OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.5.1*. Technical Report, OMG.

[36] Said Ouiazzane, Fatimazahra Barramou & Malika Addou (2020): *Towards a Multi-Agent based Network Intrusion Detection System for a Fleet of Drones*. *International Journal of Advanced Computer Science and Applications (IJACSA)* 11(10), doi:10.14569/IJACSA.2020.0111044.

Available at https://thesai.org/Publications/ViewPaper?Volume=11&Issue=10&Code=IJACSA&SerialNo=44. Number: 10 Publisher: The Science and Information (SAI) Organization Limited.

[37] Jan Peleska (1996): *Test Automation for Safety-Critical Systems: Industrial Application and Future Developments*. In Marie-Claude Gaudel & Jim Woodcock, editors: *FME '96: Industrial Benefit and Advances in Formal Methods, Third International Symposium of Formal Methods Europe, Co-Sponsored by IFIP WG 14.3, Oxford, UK, March 18-22, 1996, Proceedings*, *Lecture Notes in Computer Science* 1051, Springer, pp. 39–59, doi:10.1007/3-540-60973-3_79. Available at https://doi.org/10.1007/3-540-60973-3_79.

[38] Jan Peleska (2020): *New Distribution Paradigms for Railway Interlocking*. In Tiziana Margaria & Bernhard Steffen, editors: *Leveraging Applications of Formal Methods, Verification and Validation: Applications - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part III*, *Lecture Notes in Computer Science* 12478, Springer, pp. 434–448, doi:10.1007/978-3-030-61467-6_28. Available at https://doi.org/10.1007/978-3-030-61467-6_28.

[39] Jan Peleska, Jörg Brauer & Wen-ling Huang (2018): *Model-Based Testing for Avionic Systems Proven Benefits and Further Challenges*. In Tiziana Margaria & Bernhard Steffen, editors: *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice - 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part IV*, *Lecture Notes in Computer Science* 11247, Springer, pp. 82–103, doi:10.1007/978-3-030-03427-6_11. Available at https://doi.org/10.1007/978-3-030-03427-6_11.

[40] Jan Peleska, Wen-ling Huang & Ana Cavalcanti (2019): *Finite complete suites for CSP refinement testing*. *Science of Computer Programming* 179, pp. 1 – 23, doi:https://doi.org/10.1016/j.scico.2019.04.004. Available at http://www.sciencedirect.com/science/article/pii/S0167642319300620.

[41] Jan Peleska, Wen-ling Huang & Felix Hübner (2016): *A Novel Approach to HW/SW Integration Testing of Route-Based Interlocking System Controllers*. In Thierry Lecomte, Ralf Pinger & Alexander Romanovsky, editors: *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification - First International Conference, RSSRail 2016, Paris, France, June 28-30, 2016, Proceedings*, *Lecture Notes in Computer Science* 9707, Springer, pp. 32–49, doi:10.1007/978-3-319-33951-1_3. Available at http://dx.doi.org/10.1007/978-3-319-33951-1_3.

[42] Alexandre Petrenko (2016): *Checking Experiments for Symbolic Input/Output Finite State Machines*. In: *Ninth IEEE International Conference on Software Testing, Verification and Validation Workshops, ICST Workshops 2016, Chicago, IL, USA, April 11-15, 2016*, IEEE Computer Society, pp. 229–237, doi:10.1109/ICSTW.2016.9. Available at http://dx.doi.org/10.1109/ICSTW.2016.9.

[43] Anand S. Rao & Michael P. Georgeff (1995): *BDI Agents: From Theory to Practice*. In Victor R. Lesser & Les Gasser, editors: *Proceedings of the First International Conference on Multiagent Systems, June 12-14, 1995, San Francisco, California, USA*, The MIT Press, pp. 312–319.

[44] Martin Selecký, Michal Stolba, Tomás Meiser, Michal Cáp, Antonín Komenda, Milan Rollo, Jirí Vokrínek & Michal Pechoucek (2013): *Deployment of multi-agent algorithms for tactical operations on UAV hardware*. In Maria L. Gini, Onn Shehory, Takayuki Ito & Catholijn M.

Jonker, editors: *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13, Saint Paul, MN, USA, May 6-10, 2013*, IFAAMAS, pp. 1407–1408. Available at http://dl.acm.org/citation.cfm?id=2485248.

[45] Michal Soucha & Kirill Bogdanov (2018): *SPYH-Method: An Improvement in Testing of Finite-State Machines*. In: *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops, ICST Workshops, Västerås, Sweden, April 9-13, 2018*, IEEE Computer Society, pp. 194–203, doi:10.1109/ICSTW.2018.00050.

[46] Youcheng Sun, Hana Chockler, Xiaowei Huang & Daniel Kroening (2020): *Explaining Image Classifiers Using Statistical Fault Localization*. In Andrea Vedaldi, Horst Bischof, Thomas Brox & Jan-Michael Frahm, editors: *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XXVIII*, *Lecture Notes in Computer Science* 12373, Springer, pp. 391–406, doi:10.1007/978-3-030-58604-1_24. Available at https://doi.org/10.1007/978-3-030-58604-1_24.

[47] RTCA SC-205/EUROCAE WG-71 (2011): *Formal Methods Supplement to DO-178C and DO-278A*. Technical Report RTCA/DO-333, RTCA Inc, 1150 18^{th} Street, NW, Suite 910, Washington, D.C. 20036-3816 USA.

[48] RTCA SC-205/EUROCAE WG-71 (2011): *Model-Based Development and Verification Supplement to DO-178C and DO-278A*. Technical Report RTCA/DO-331, RTCA Inc, 1150 18^{th} Street, NW, Suite 910, Washington, D.C. 20036-3816 USA.

[49] RTCA SC-205/EUROCAE WG-71 (2011): *Software Considerations in Airborne Systems and Equipment Certification*. Technical Report RTCA/DO-178C, RTCA Inc, 1150 18^{th} Street, NW, Suite 910, Washington, D.C. 20036-3816 USA.

[50] RTCA SC-205/EUROCAE WG-71 (2011): *Software Tool Qualification Considerations*. Technical Report RTCA/DO-330, RTCA Inc, 1150 18^{th} Street, NW, Suite 910, Washington, D.C. 20036-3816 USA.

[51] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David B. Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter P. Puschner, Jan Staschulat & Per Stenström (2008): *The worst-case execution-time problem - overview of methods and survey of tools*. *ACM Trans. Embed. Comput. Syst.* 7(3), pp. 36:1–36:53, doi:10.1145/1347375.1347389. Available at https://doi.org/10.1145/1347375.1347389.

[52] Richard Williams, Boris Konev & Frans Coenen (2014): *Multi-agent Environment Exploration with AR.Drones*. In Michael N. Mistry, Ales Leonardis, Mark Witkowski & Chris Melhuish, editors: *Advances in Autonomous Robotics Systems - 15th Annual Conference, TAROS 2014, Birmingham, UK, September 1-3, 2014. Proceedings*, *Lecture Notes in Computer Science* 8717, Springer, pp. 60–71, doi:10.1007/978-3-319-10401-0_6. Available at https://doi.org/10.1007/978-3-319-10401-0_6.