OETCS/WP4/CSM – 01/00

openETCS

ITEA2 Project
Call 6 11025
2012 – 2015

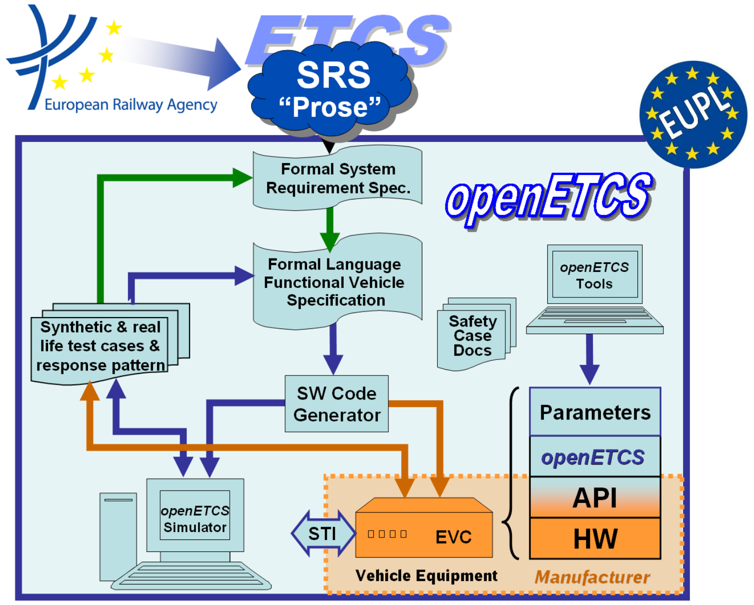
Work Package 4: “Verification & Validation Strategy”
Work Package 7: Tool Chain

Combination of Behavioral and Parametric Diagrams for Model-based Testing – Application to ETCS Target Speed Monitoring

Technical report

Felix Hübner, Christoph Hilken and Jan Peleska

2015-11-25



Funded by:

Federal Ministry of Education and Research
Ministère de l'Enseignement Supérieur et de la Recherche
Investissements d'Avenir
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 **2015-11-25**

Combination of Behavioral and Parametric Diagrams for Model-based Testing – Application to ETCS Target Speed Monitoring

Technical report

Felix Hübner, Christoph Hilken and Jan Peleska

University of 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. The technology presented here allows for the incorporation of braking curves and other time-continuous behaviours of trains into test models, from where test cases, concrete test data, and test procedures can be automatically generated. The proposed technology is currently implemented in the RT-Tester tool which has been made available for the openETCS project as part of WP7.

Model-based testing is a promising approach to cope with the complexity of today's train control systems. It shifts the effort from manual test case specification to the modeling of the system's behavior and enables automated test case generation. System behavior is often modeled by state machines. However, state machines are not appropriate to model all kinds of behavior, in particular time-continuous evolutions such as time-dependent position, speed, and acceleration of trains. This work combines state machines and parametric diagrams of the Systems Modeling Language (SysML) to enable the description of time-discrete and time-continuous control in test models, at the same time introducing a new concept for a higher degree of abstraction. This is shown by application to a real-word example: a safety function of the European Train Control System (ETCS), the so-called target speed monitoring function.

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

1 Introduction

Model-based testing

Model-based testing (MBT) has gained much attention during the last decade [17, 14, 3]. This is mainly due to the fact that MBT enables a high degree of automation, increasing the efficiency of test-related verification and validation activities in a considerable way. The main automation benefits are mechanized test case creation from the model, test data calculation by means of mathematical constraint solvers, test procedure generation using model-based code generation techniques, and compilation of traceability data relating testing artifacts to requirements by exploiting traceability mechanisms available in the modeling languages [13]. At the same time, MBT allows for the application of more complex test strategies. These provide higher test strength, but the test case generation algorithms involved can no longer be managed in a manual way; examples of these more complex strategies are given in [8, 15, 10].

For automated MBT, the modeling formalism applied needs to be associated with a formalized behavioral semantics describing how model states, inputs, and outputs evolve over time. For test models described in the SysML formalism [12], formalization options are described, for example, in [13, 9]. With these results at hand, model-based testing against concurrent real-time SysML models can be considered as a solved problem for continuous time/discrete control systems depending on notions of discrete or dense time, but producing discrete control outputs only. The system behavior over time is modeled, for example, by means of concurrent SysML state machines, whose trigger conditions depend on variable values and timer conditions. This is then formally specified by a transition relation describing how discrete control steps or time-delays are performed. Using, for example, an SMT solver that is also capable of floating point arithmetics, the possible transition steps can be calculated. Specific test objectives can be encoded as additional constraints used in conjunction with the transition relation, so that the solutions provided by the constraint solver describe at the same time valid state transitions of the model and suitable candidates for the test objective under consideration.

The challenge

For real-time systems depending on mixed time-discrete and time-continuous evolutions of observables and/or control variables, no comprehensive MBT methodology exists yet. While the formal semantics of these so-called *hybrid systems* has been thoroughly investigated [7, 2], the automated calculation of suitable test data for practical MBT still remains a challenge. This is mainly due to the fact that best practices for specifying time-continuous evolutions in test models and creating associated concrete data by means of constraint solving are still subject to discussions.

Objectives and main contributions

In this paper, a novel approach to MBT in a hybrid systems context is presented, based on the SysML modeling language. The utilization of blocks and associated diagrams for decomposing the functionality of the system under test (SUT) and the use of state machines is "imported" from proven MBT technology for time-discrete systems. These description means are, however, combined with an abstraction technique and extended by *constraint blocks* and *parametric diagrams* [12, Section 10] for modeling time-continuous dependencies between inputs, outputs, and model variables. From this descriptions means our proposed approach is able to generate concrete test cases in a fully automatic way.

Our approach is illustrated and a proof of concept is given by application to a complex real-world system. We create a test model of a control problem from the *European Train Control System (ETCS)*, using the system requirement specification [16]. We describe how the expected EVC behavior can be modeled using the SysML subset indicated above, and the computational effort needed for automated test generation by means of an SMT solver is evaluated.

2 Background

In order to keep this paper self contained relevant parts of the European Train Control System and the Systems Modeling Language are introduced.

2.1 European Train Control System

The European Train Control System (ETCS) shall replace existing national train control systems by a modern unified system in the European Union. The system requirement specification is publicly available [16].

The onboard computer of ETCS trains (the so-called *European Vital Computer (EVC)*) performs – among other safety-relevant control tasks – *target speed monitoring (TSM)*: the speed of the train, while approaching a target (for example, a train station or a level crossing), is monitored by the EVC, so that the ability to brake the train in time for the target is always ensured [16]. To this end, the speed-dependent *braking curves* of the train have to be taken into account, so that the braking distance required is correctly calculated.

In the remainder, the following variables are used: v_{est} is a system input describing the current train speed. The maximum allowed speed on the track section of the train is denoted by v_{mrsp}. The value of v_{mrsp} should remain constant over time, while v_{est} should change according to the current acceleration a. All track related locations, including the train position, are measured as track relative distances D, starting from zero (start location) and ending at the target location D_{Target}. The train position is given by variable D_{front} and should always have values from the range $[0, D_{\text{Target}}]$. D_{Target} is constant during runtime, while D_{front} might change over time as the train moves forward towards the target location.

2.2 Systems modeling language

OMG's *Systems Modeling Language SysML* [12] is a semi-formal language to specify a model of the system's structure and behavior. In addition, corresponding diagrams offer a graphical representation of parts of the model. Therefore, a system description is composed of different kinds of diagrams, for example block definition diagrams and internal block diagrams to describe the structure, complemented by activity and state machine diagrams to describe the behavior of the system. Our approach is working on state machines and constraint blocks. Therefore, to focus on the objectives of this paper, in the remainder only parametric diagrams and state machine diagrams are used.

Constraint blocks

Constraint blocks are used to express general dependencies between observables, such as physical laws. Parametric diagrams are used to bind the generic names of these observables to the concrete model variables that are restricted by these constraints. Using these syntactic entities, the boundary conditions restricting time-continuous inputs or outputs of the SUT can be suitably specified.

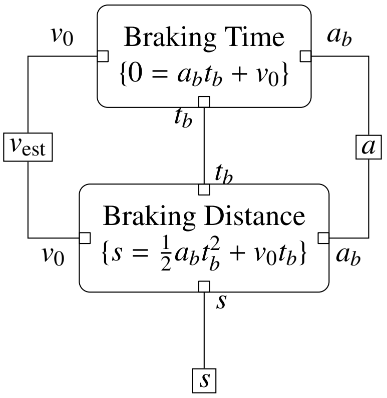


Figure 1. Parametric diagram for the calculation of the braking distance

Example 1 *For the TSM function the braking distance has to be calculated. The motion of a train can be expressed by the equations for a translational acceleration, which are $v = at + v_o$ and $s = \frac{1}{2}at^2 + v_0 t + s_0$. In Figure 1 both expressions are combined to calculate the braking distance s. The constraints are bound to the system inputs (v_{est} and a) in the parametric diagram.*

SysML does not define a language to express constraints. In the remainder the natural mathematical notation is used, which is interpreted by our tools. In addition, it is possible to add conditions under which constraints are valid. Therefore, pseudocode is used. In particular, such conditioned constraints can be used to specify the so-called *flow conditions* of *hybrid automata* associated with control modes specified in state machines: following [7], these flow conditions describe the restrictions to be observed by time-continuous variables, while the system resides in the given control mode.

State machines and testing strategies

UML's state machines have been adopted for SysML without changes. Hence, SysML state machines are a variant of Harel's statechart formalism [6]. Therefore, many MBT-tools are using state machines as modeling formalism, e.g.[1, 4] and many testing strategies have been developed in respect to state machines. This includes conventional strategies based on coverage criteria, but also complete test strategies, that give a guarantee that certain types of faults are revealed, given an assumption that only pre-defined types of faults can occur. In addition, it was shown in [11] that test suites generated by a complete test strategy can have a very high test strength even if the assumption for completeness does not hold.

3 Problem Formulation and General Idea

As mentioned before, state machines have been shown suitable to model the behavior of reactive systems in the past and MBT approaches chose them as description means for their input models. With respect to upcoming cyberphysical-systems state machines are not convenient, especially, due to the integration of components, such as sensors and actuators in today's embedded systems. Such hybrid systems can be described by a combination of state machines and conditioned constraint blocks. Constraint blocks and parametric diagrams are well suited to describe mathematical expressions, as for example physical laws, due to their declarative manner. State machines on the opposite are an imperative, state-based description means and, hence, are not convenient to specify physical properties of the system. There are many approaches in the field of control theory and simulation to handle physical properties, but in the field of MBT physical constraints are usually neglected or simplified and current test strategies do not take

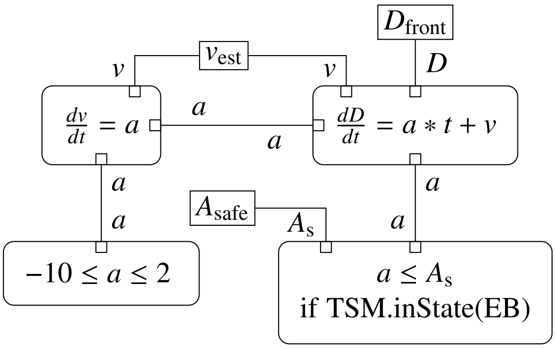


Figure 2. Parametric diagram describing the temporal evolution of the TSM

them into account. As a consequence, test cases generated using current tools and strategies may be unrealistic and violate physical constraints.

Example 2 *Assume test cases shall be generated for the TSM function. Without considering physical laws, a concrete test case might be generated that contains a sequence of train speed values that have sudden jumps from very low to very high values. Such a behavior is unrealistic, since the acceleration of a train is bounded by the train engine and train brakes. Figure 2 describes the physical constraints, that model how the speed and position of a train should evolve over time. Considering this kind of physical constraints leads to test cases that are executable in a real environment.*

To overcome this situation and to allow the utilization of parametric diagrams as well as constraint blocks, we propose the following approach:
In a first step using an abstracted version of the state machine, abstract test cases are generated. Such an abstracted version is a so-called *simulation* of the concrete system [5]; the former represents an over-approximation of the latter. The simulation can easily be obtained by using Boolean variables as guard condition for every transition in the state machine. These Boolean variables are not restricted on the state machine level but bound to concrete system variables in the parametric diagrams.[1] Abstract test cases are constructed from this simulation model using an input equivalence class partitioning test strategy with proven error detection capabilities [10]. [2] The inputs associated with an abstract test case are sequences of input equivalence classes.

In the second step, the abstract input sequences are resolved to sequences of concrete model variable valuations, using a mathematical constraint solver. For this step, the bindings of abstract Boolean condition variables to concrete model variables defined by parametric diagrams are taken into account. Additional physical constraints describing the temporal evolution of system variables are considered in this step as well, which leads to realistic concrete test cases.

4 Two-step Approach

Step 1. System abstraction

In the first step, the system behavior is abstracted. The idea is to use an overapproximation of the specified system's behavior: all traces possible in the concrete system based on the specification, considering all physical and system related constraints, are also observable in the abstract model.

[1] Given a concrete system description by means of a state machine, this simulation can as well be obtained automatically. But in most cases, the a priori modeling of an abstract state machine description of the system might be more natural and reduce the modeling effort, as will be shown in section 5.

[2] Apart from that, the proposed approach allows the application of every other state machine based testing strategy

Whether a trace in the abstract model has a concrete solution in the real world, depends on the abstraction and constraints. This is a conservative approach, which guarantees that every system behavior is covered by the model, while unrealistic traces might remain. If in the second step, no concrete solution for an abstract test exists, it can be deduced, that this behavior is not possible (assuming that the constraints are correct), and therefore it is safe to neglect this test case.

Example 3 *Consider the following complex boolean expression, taken from the ETCS specification [16]:*

$$V_{est} > V_{mrsp} + DV_{ebi}(V_{mrsp}) \vee D_{front} > D_{EBI} \tag{1}$$

This boolean expression describes a trigger condition for the onboard controller of the train to automatically activate the emergency brakes. Variable V_{est} describes the current train speed. If this speed exceeds the maximum allowed speed V_{mrsp} by more than an offset DV_{ebi}, the emergency brakes are activated. Alternatively, if the front of the train D_{front} is too close to the target location (this is expressed by $D_{front} > D_{EBI}$), the emergency brakes are activated as well.

Instead of explicitly describing this trigger condition by concrete model variables, an abstract input variable t_{13} is introduced. The state machine transition that uses this abstract input variable looks like this: $\boxed{L1} \xrightarrow{[t_{13}]} \boxed{L2}$. In this case we introduced a new free input variable. It can be assumed, that the value of this variable can change arbitrarily over time. This is a safe overapproximation of the real system.

The replacement of a complex guard-condition by a new boolean input variable is always possible, but n replacements introduce n new inputs. In the worst case this results in 2^n new input equivalence classes of the SUT [10]: each feasible conjunction of the positive or negated Boolean expressions associated with each of the abstraction variables t_i specifies an input equivalence class. For realistic models, not all combinations of valuations of the complex conditions are possible. Using an SMT solver, impossible input combinations can be dropped a priori. These combinations are ignored in the test generation.

With this abstract description of the system behavior at hand, we are able to apply the test generation strategy presented in [10] and create a test suite, whose inputs are represented by abstract equivalence classes.

Example 4 *For our running example, the TSM function, we introduced nine abstract input variables. From the $2^9 = 512$ possible combinations, only eight combinations were valid. Two of the combinations can be considered as equivalent [10]. Thus, seven input equivalence classes were calculated from our abstract model.*

The result of the first step in our test case generation is a set of abstract test cases. An abstract test case is a sequence of system states $\langle \bar{s}_0, \ldots, \bar{s}_n \rangle$, where $\bar{s}_i : V_{\text{abstract}} \to D$ is the valuation function, mapping the (abstract) Boolean system variables t_k to their values $\bar{s}_i(t_k)$ in the i-th state.

Step 2. Concretization

While the first step defined abstract test cases that are related to computations in the abstract test model, the second step aims at constructing concrete test data.

A concrete test case is a sequence of concrete system states $\langle s_0, \ldots, s_n \rangle$. Each s_i is a valuation function $s_i : V \cup V_{\text{abstract}} \to D$, where the concrete system variables v, including the inputs and outputs, as well as the abstraction variable introduced before, are mapped to concrete values $s_i(v) \in D$.

For the concretization of the test cases, parametric diagrams were added as an additional description means. The abstract variables introduced in Step 1 have to be related to concrete system variables by parametric diagrams. Two types of constraints can be defined: *state invariants* and *temporal evolution*.

State invariants constrain the concrete system variables in every state and, therefore, in every test step of the test case, these invariants have to be fulfilled. In a parametric diagram a state invariant can be described by a constraint property. A parametric diagram with m constraint properties that define state invariants $(\text{INV}_1, \ldots, \text{INV}_m)$ yields the invariant:

$\text{INV} \equiv \bigwedge_{j=1}^{m} \text{INV}_j$

Invariants in the concrete system are also used to bind the values of abstract system variables to the concrete guard conditions they are abstracting.

Example 5 *Considering trigger condition t_{13} from example 3 the corresponding parametric diagram is shown in Figure 4. In this diagram t_{13} gets bound to its concrete guard condition.*

Given this invariant, and given an abstract test case $\langle \bar{s}_0, \ldots, \bar{s}_n \rangle$, we want to calculate a concrete test case $\langle s_0, \ldots, s_n \rangle$ that uses the same valuations of the abstract system variables – that is, the same equivalence classes – as prescribed by the abstract test case. This is encoded by the following formula, where $\phi[t/x]$ denotes the formula derived from ϕ by exchanging every free occurrence of x by the term t.

$$\bigwedge_{i=0}^{n} (\bigwedge_{t \in V_{\text{abstract}}} s_i(t) = \bar{s}_i(t) \wedge \quad (2)$$

$$\text{INV}[s_i(v)/v, v \in V \cup V_{\text{abstract}}]) \quad (3)$$

The above formula can be solved by an SMT solver. The result is a concrete mapping of variables from V to concrete values. Using these concrete values, a concrete test case $\langle s_0, \ldots, s_n \rangle$ is calculated.

Apart from that, the temporal evolution of system variables is described by means of parametric diagrams. In contrast to an invariant, such constraints describe the change of values between two test steps. For example it may necessary to constrain the change of velocity and location due to the physical laws of acceleration. Figure 2 gives an example for this kind of constraints. This kind of constraints can best be described by differential equations.

In our approach we support the definition of linear differential equations. These equations can be discretized and translated to expressions relating pre and post states. These expressions contain

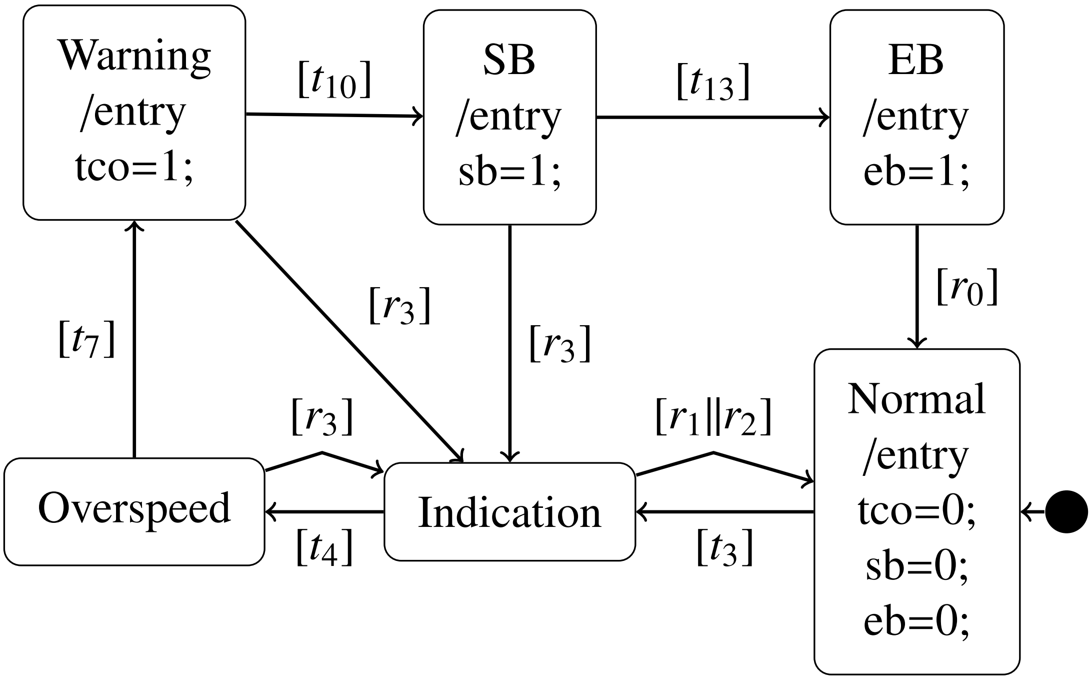


Figure 3. State Machine of the TSM

unprimed variable symbols V describing the variable in the pre-state s_i, and primed variable symbols V' describing the post-state s_{i+1}. For example the differential equation $\frac{dv}{dt} = a$ in Figure 2 can be expressed by the constraint: $t' = t + \Delta t \wedge {V_{\text{est}}}' = V_{\text{est}} + a \cdot \Delta t$.

All constraint properties describing temporal evolution ($\text{TEMP}_1, \ldots, \text{TEMP}_p$) can be summarized:

$\text{TEMP} = \bigwedge_{k=1}^{p} \text{TEMP}_k$

We extend the SMT instance to generate concrete test cases respecting the time-continuous evolution defined by parametric diagrams as follows:

$$\bigwedge_{i=0}^{n}\Big(\bigwedge_{t \in V_{\text{abstract}}} s_i(t) = \bar{s}_i(t) \wedge \tag{4}$$

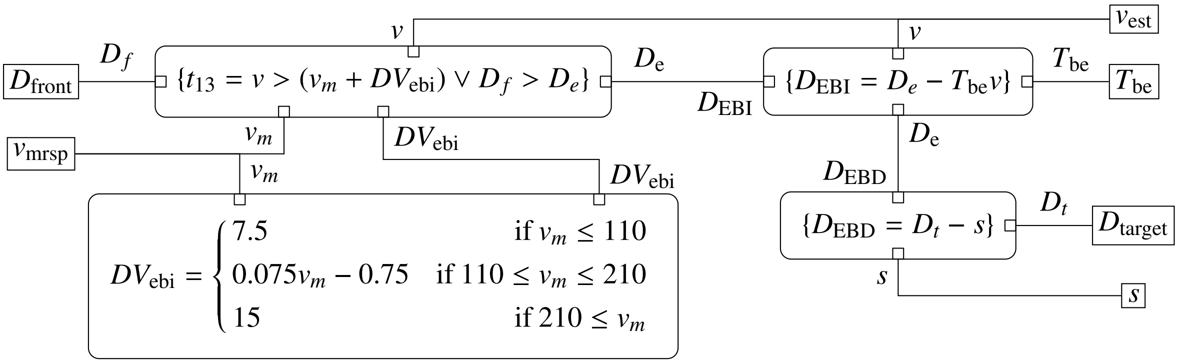
$$\text{INV}[s_i(v)/v, v \in V \cup V_{\text{abstract}}]\Big) \wedge \tag{5}$$

$$\bigwedge_{i=0}^{n-1} \text{TEMP}[s_i(v)/v, v \in V \cup V_{\text{abstract}}, \tag{6}$$

$$s_{i+1}(v)/v', v' \in V' \cup V'_{\text{abstract}}] \tag{7}$$

5 Evaluation

The approach described in the previous sections has been applied to the TSM function of the ETCS specification [16]. The specification describes the behavior in means of a transition table with the corresponding conditions for each transition. Such a specification can easily be modeled in terms of state machines and parametric diagrams: For every condition defined in the specification a Boolean variable $t_3, \ldots, t_{13}$, $r_0, \ldots, r_3$ was introduced, named according to the definitions in [16]. The state machine can easily be deduced from the transition table using the introduced Boolean variables. The corresponding state machine of the TSM function is shown in



$$DV_{\mathrm{ebi}} = \begin{cases} 7.5 & \text{if } v_m \leq 110 \\ 0.075v_m - 0.75 & \text{if } 110 \leq v_m \leq 210 \\ 15 & \text{if } 210 \leq v_m \end{cases}$$

Figure 4. Parametric Diagram of the TSM

Figure 3. The concrete conditions are described and the introduced Boolean variables are bound by means of parametric diagrams. An excerpt of the parametric diagram of the TSM function is given in Figure 4.

As a result, specifying the constraint blocks and creating the parametric diagram, as well as the state machine, was straightforward, where only the mathematical equations given in [16] had to be translated to constraint blocks and some notations had to be slightly modified to be parsed by our tools. Thus, the modeling effort for this example was very low and we assume that the modelling effort should scale very well if other sub functions of this specification were modeled with the proposed approach.

Model description

In the TSM function state changes occur as soon as trigger conditions are fulfilled. Under certain trigger conditions (t_3,t_4, ...) the system changes its state machine state corresponding to different levels of intervention. t_3 is a boolean expression that evaluates to true, if the current velocity V_{est} and the current train front position D_{front} exceed the line indicated by D_I in Figure 5. Similarly, t_4, t_7, t_{10} and t_{13} return true if and only if the train exceeds $D_{\mathrm{P}}, D_{\mathrm{W}}, D_{\mathrm{SBI}}$ or D_{EBI} respectively. When a new state machine state is entered, the *entry*-actions write to the output variables tco, sb and eb. These variables indicate the activation of the traction cut-off, the service brake and the emergency brake. The revocation conditions $r_0, \ldots, r_3$ describe the conditions necessary for the intervention states to be revoked.

The *state invariants* of our test model are described by means of a parametric diagram and as an example in Figure 4 the definition of trigger condition t_{13} is shown. For convenience the definition of the other triggering conditions is omitted.

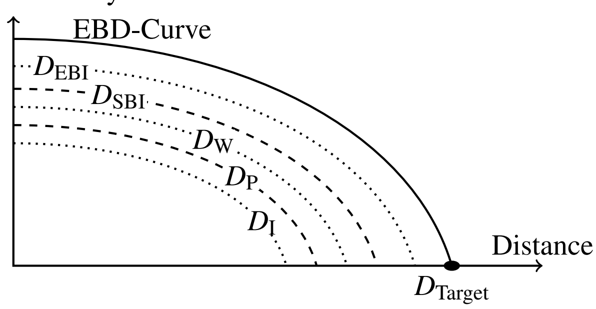


Figure 5. Braking Curves of the TSM

All these triggering conditions depend on the train velocity and track relative distance (position). The conditions can be calculated from the EBD curve, compare Figure 5. The EBD curve (emergency brake deceleration) maps the track relative distance to a velocity, assuming a negative acceleration A_{safe}, such that zero speed is reached at the target position (D_{Target}). The curve can be calculated by calculation of the braking distance s as shown in Figure 1. The value of A_{safe}[3] is a conservative approximation that describes the least deceleration that the emergency brakes of the train achieve when fully activated. Because the full activation of the emergency brakes needs some time, the curve D_{EBI} is shifted by a fixed time delay. If the current distance-velocity-pair describing the train speed and position is right or above this line, the emergency brakes are triggered. In a similar way the other triggering conditions are defined by shifting the EBD cure by fixed time delays.

The *temporal evolution* of our test model is described by the parametric diagram shown in Figure 2. The constraints in this diagram restrict the time-continuous evolution of the trains front position and the velocity. Therefore the acceleration a of the train is restricted to values in the range $[-10, 2]$. If the emergency brakes are activated (the train is in the state machine state EB), the deceleration must be higher than prescribed by A_{safe}. As described in section 4.0.0.2, the discretized versions of the constraints were used.

In total, our test model used for this evaluation contains 52 constraint properties, including those shown in Figure 1,2 and 4.

Test case generation

Table 1 gives a summary of our test case generation. The test case generation was performed on a system with 24 CPU cores running at 2.8 GHz with 16 GB RAM. In the first step of our test case generation 50 abstract test cases were calculated, which took less than a second. Then it took three and a half hours to generate the concrete test cases from the 50 abstract test cases. The main reason, that this computationally expensive task, is still applicable in acceptable time, is that SMT instances for every test case can be solved in parallel. The test case generation was run in parallel and on average 16.7 of the 24 CPUs were utilized.

Given that the results from this TSM model can be generalized to hybrid systems of comparable complexity, the runtime is acceptable. Yet, the results indicate, that the utilization of a different computational backend, e.g. analytical or numerical solvers, should be considered.

For every abstract test case a concrete solution was calculated. Thus, our model abstraction seems appropriate. This might not always be the case. If the test case generation of abstract test cases yields a low percentage of solvable concrete test cases, this might be an indicator for inappropriate abstraction or too restrictive constraints.

number of test cases generated	50
avg. test case length (test steps)	2.84
time for abstract test case generation	$0.8s$
time for calculation of concrete test cases	$3.5h$
memory for test case generation	8.4 GB

Table 1. Results of the Test Case Generation for the TSM Model

[3] For convenience we used a fixed value for A_{safe}. According to the ETCS specification [16] up to seven values can be defined as a step function dependent on the train speed.

6 Conclusion

We have presented a novel approach to automated model-based testing of mixed time-discrete and time-continuous systems. It has been shown that SysML is a suitable formalism for creating test models of this kind: time-discrete control aspects are reflected by state machines, and time-continuous constraints are represented by constraint blocks and parametric diagrams. Concrete test cases were generated in a two-step approach. At first, an abstraction of the concrete system was constructed, and an input equivalence class partition strategy with proven fault detection capabilities was applied to generate a test suite of abstract test cases. The inputs associated with each of these test cases are sequences of input equivalence classes. For the second step, concrete solutions for each of these abstract test cases were generated. For this purpose, each abstract test case is transformed into constraints over concrete model variables, and the additional constraints coming from time-continuous conditions specified in constraint blocks are taken into account as well.

Using an SMT solver that is capable of processing floating point arithmetics, it was shown that the approach is suitable for practical application. To this end, tests for a safety-relevant control function of the ETCS onboard controller have been generated in an automated way. The results show that MBT is feasible for hybrid systems that are comparable to the complexity of this real-world example. The model-based test generation and execution process has been implemented in an experimental version of an industrial-strength testing tool. As mentioned, we are currently exploring numerical solvers or computer algebra systems to get a better performance and to support more complex mathematical expressions.

References

[1] S. Ali, M. Iqbal, A. Arcuri, and L. Briand. A Search-Based OCL Constraint Solver for Model-Based Test Data Generation. In *2011 11th International Conference on Quality Software (QSIC)*, pages 41–50, July 2011.

[2] R. Alur, T. Dang, J. Esposito, R. Fierro, Y. Hur, F. Ivančić, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky. Hierarchical hybrid modeling of embedded systems. *Lecture Notes in Computer Science*, 2211:14–31, 2001.

[3] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, P. McMinn, A. Bertolino, J. Jenny Li, and H. Zhu. An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software*, 86(8):1978–2001, Aug. 2013.

[4] F. Bouquet, C. Grandpierre, B. Legeard, F. Peureux, N. Vacelet, and M. Utting. A Subset of Precise UML for Model-based Testing. In *Proceedings of the 3rd International Workshop on Advances in Model-based Testing*, A-MOST '07, pages 95–104, New York, NY, USA, 2007. ACM.

[5] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[6] D. Harel and A. Naamad. The statemate semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, October 1996.

[7] T. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pages 278–292. IEEE Computer Society Press, 1996.

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

[9] C. Hilken, J. Peleska, and R. Wille. A Unified Formulation of Behavioral Semantics for SysML Models. pages 263–271, Feb. 2015.

[10] W.-l. Huang and J. Peleska. Complete model-based equivalence class testing. *International Journal on Software Tools for Technology Transfer*, pages 1–19, Nov. 2014.

[11] F. Hübner, W.-l. Huang, and J. Peleska. Experimental Evaluation of a Novel Equivalence Class Partition Testing Strategy. In J. C. Blanchette and N. Kosmatov, editors, *Tests and Proofs*, number 9154 in Lecture Notes in Computer Science, pages 155–172. Springer International Publishing, July 2015.

[12] Object Management Group. OMG Systems Modeling Language (OMG SysML$^{\text{TM}}$), Version 1.4. Technical report, Object Management Group, 2015. http://www.omg.org/spec/SysML/1.4.

[13] J. Peleska. Industrial-strength model-based testing - state of the art and current challenges. In A. K. Petrenko and H. 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.

[14] A. Petrenko, A. Simao, and J. C. Maldonado. Model-based testing of software and systems: Recent advances and challenges. *Int. J. Softw. Tools Technol. Transf.*, 14(4):383–386, Aug. 2012.

[15] 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.

[16] UNISIG. *ERTMS/ETCS System Requirements Specification, Chapter 3, Principles*, volume Subset-026-3, chapter 3. February 2012. Issue 3.3.0.

[17] M. Utting, A. Pretschner, and B. Legeard. A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability*, 22(5):297–312, Aug. 2012.