CHAPTER

c0095
MODEL-BASED TESTING OF CYBER-PHYSICAL SYSTEMS 19

A. Aerts*, M. Reniers*, M.R. Mousavi†

Eindhoven University of Technology, Eindhoven, The Netherlands Halmstad University, Halmstad, Sweden†*

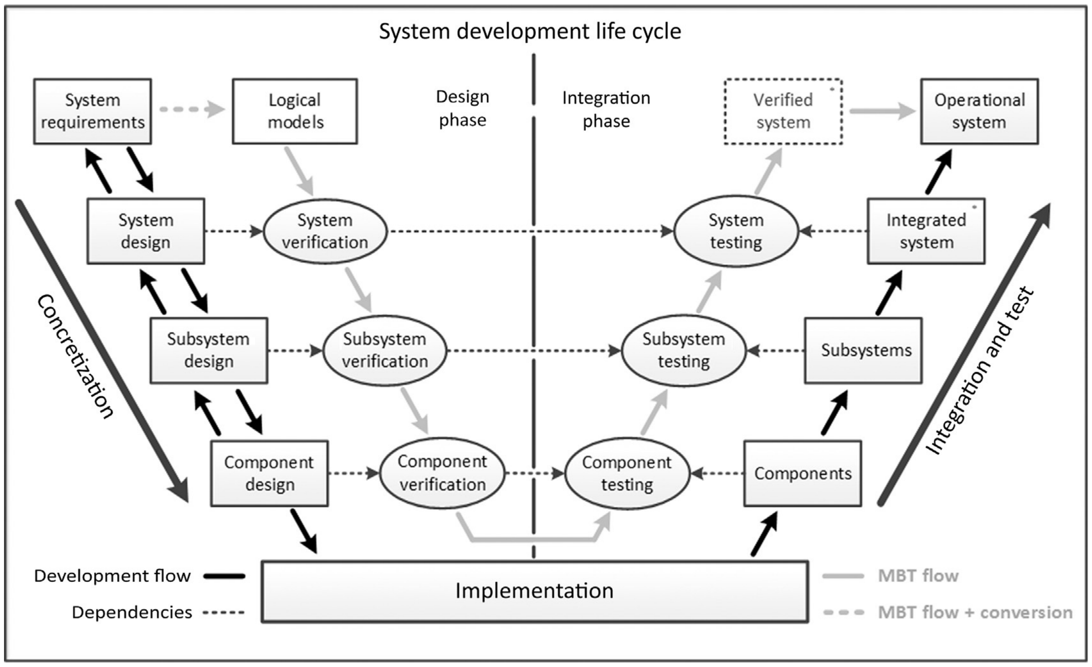
s0010
1 MODEL-BASED TESTING

s0015
1.1 WHY, WHAT, AND WHEN

p0010 The design of a cyber-physical system (CPS) is typically characterized by the use of a system development life cycle (SDLC), such as the one depicted in Fig. 1 (the outer V depicted using solid black lines). This life cycle represents how an abstract artifact such as system requirements is gradually refined into a concrete implementation artifact (the bottom-most part of the V) and then integrated into a part of an operational system.

p0015 Ideally, each of these development activities is accompanied with testing activities, including validation and verification activities (the inner V depicted using solid gray lines in Fig. 1): validation makes sure that the artifacts developed in the activity are in-line with users' intentions and verification ensures that the developed artifacts are consistent with the artifacts of the other activities. Validation often involves users (or user models). Several challenges make validation and verification nontrivial: firstly, it is not always easy to translate user and consistency requirements into concrete test inputs; secondly, it is not straightforward to judge the outcome of tests. The latter problem is often called "the oracle problem" (Binder, 1999); constructing a complete and correct oracle is as difficult Q1 as building a complete system. These two problems are intensified in the context of CPSs due to their huge domain of test inputs (endless interactions with huge parameter space) and complex correctness criteria.

p0020 Model-based testing (MBT) provides solutions to these two problems. Models are exploited to generate test inputs automatically and models also define the correctness (conformance) criteria in order to produce a test verdict after (or while) executing test cases. MBT can be performed at various steps of the CPS development life cycle, as depicted in Fig. 1 in the inner V-path (gray), along the whole development cycle. Applying MBT at early development phases allows for early detection of faults, which often proves to be efficient and cost-effective (Vishal et al., 2012). In addition to the classical benefit of early testing, such as reduced debugging cost and effort (Boehm, 1981), MBT provides a structured and systematic method for testing at various stages of the CPS development life cycle.



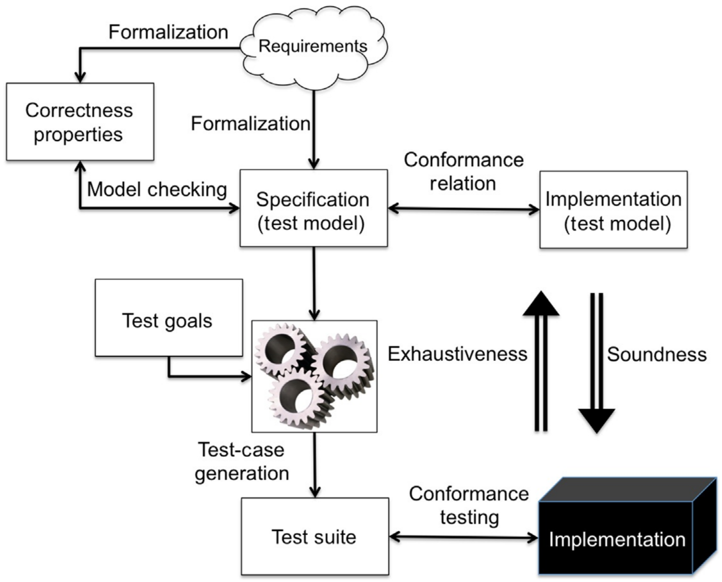
f0010 **FIG. 1**

System development life cycle of cyber-physical systems (V-model), inspired by Firesmith (2014).

s0020 1.2 HOW

p0025 MBT can be applied throughout the V-model, and is hence applicable on different types of artifacts. In the design phase (see Fig. 1), specifications are used in order to test the correctness of the design artifacts (see Section 2). A prerequisite of this verification task is a precise specification of the requirements. This is a highly nontrivial first challenge, ie, incorrect specifications lead to incorrect designs and subsequently to invalid conformance testing verdicts throughout the development process. To overcome this challenge, techniques such as model-checking can be used (Baier and Katoen, 2008; Alur et al., 1995).

p0030 Conformance testing is the process of verifying the correctness of an artifact in the development cycle of a CPS against its model. Mathematically, a notion of conformance is defined as a relation between two models: an abstract one and a concrete one; in order to designate these two models, we refer to the abstract one as the specification and the concrete one as the implementation. (The implementation need not be the actual implementation, but can be any concretized artifact.) Hence, conformance testing relies on a fundamental assumption that the specification and the implementation are both representable by *some* model. Fig. 2 provides an overview of the conformance testing process and its mathematical underpinning (conformance relation, to be discussed in Section 3). The model of the implementation is often too large to be explicitly represented and is often assumed to be only partially known or even completely unknown. Hence, conformance testing, ie, test-case generation (based on the specification model, see Section 4), test-case execution, and conformance analysis (evaluating



f0015 **FIG. 2**

Schematic view of conformance testing and conformance relation, inspired by Utting et al. (2012).

the conformance relation, see Section 3) are used to establish the conformance relation without having direct access to the underlying model of the implementation. Typically the notions of conformance relation and conformance testing are proven to be equivalent through a *soundness* and *exhaustiveness* (also called completeness) theorem. In other words, it is proven that any conforming system passes all the test cases generated in the conformance testing process and any nonconforming system will fail at least one such test case.

p0035 In practice, specification models are often absent or out of date and, hence, model learning techniques (Aarts et al., 2015; Meinke et al., 2012) can be used to learn abstract models through interactions with the implementation under test. As stated before, one must ensure that the model does indeed represent the desired behavior described by the user or system requirements. To this end, models are formally checked (or tested) against more abstract properties or manually reviewed by the domain experts for confirmation or adjustments.

p0040 The application of MBT to support a V-model SDLC (see Fig. 1) can be interpreted as part of a systems engineering framework (Blanchard et al., 1990), which provides means for system development using a SDLC. Particularly, the domain of model-based systems engineering (MBSE), which includes MBT, is becoming increasingly popular. For an overview of MBSE methodologies, see Estefan et al. (2007). To support this discipline, the Systems Modeling Language (SysML) (Object Management Group, 2014) was created which, amongst others, provides verification and validation functionality for CPSs (Friedenthal et al., 2014). MBT of CPSs as considered in the remainder of this chapter can be seen as part of the verification activities. However, in order for SysML to utilize such techniques, transformations are required to exploit external (model) simulation environments such as Modelica (Modelica Association et al., 2005).

s0025
1.3 RUNNING EXAMPLE: THERMOSTAT

p0045 To illustrate the concepts introduced in this chapter, we use the following example. The example is a system that comprises a thermostat device located in a room with a window. Because the thermostat can be either fully ON or switched OFF, and no accurate (feedback) control is applied, the system is considered unregulated. Instead an acceptable temperature interval is specified by which the thermostat switches accordingly (ON/OFF). In addition, the window provides an input to the system by influencing the temperature increase/decrease in the room, hence making the entire system input dependent.

p0050 This example will be used in the remainder of the chapter in order to illustrate the core concepts in the field of MBT.

s0030
1.4 ORGANIZATION OF THE CHAPTER

p0055 In the remainder of this chapter, we discuss different aspects of MBT for CPSs. To start with, in Section 2, we review the different types of models that can be used for the purpose of testing CPSs. In Section 3, we discuss how a mathematical definition of a CPS-conformance relation can be devised. Then, in Section 4, we study how such a conformance relation can be tested through test-case generation and test-case execution. In Section 5, we provide an overview of the existing tooling for this purpose and finally, in Section 6, we conclude the chapter by presenting some of the remaining challenges in the field of MBT for CPSs.

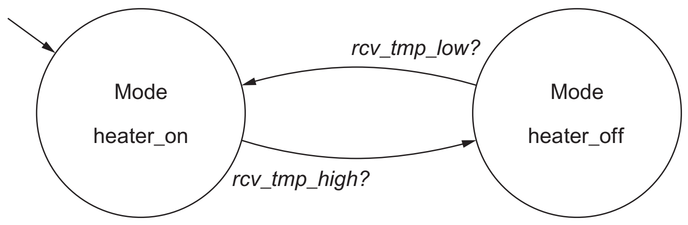
s0035
2 MODELS

p0060 In this section, we review the different types of models used for capturing the specification (and type of the purported model of the implementation) for different notions of conformance testing.

s0040
2.1 STATE-MACHINE-BASED MODELS

p0065 Finite state machines (FSMs) or finite automata have been used traditionally in hardware modeling and hardware testing (Lee, 1996). Extensions of such models with variables and data, eg, extended FSMs and class FSMs, were also proposed for software testing (Hierons et al., 2009; Hong, 1995). In an FSM the Q2 history of interactions with the system (eg, the current valuations of variables or signals) is represented by the system state and the transitions capture the system output as the result of providing an input to the system. Labeled transition systems (LTSs) (Uselton and Smolka, 1994) are variants of FSMs that remove the finiteness assumption and also relax the tight coupling between input and output (eg, a sequence of several inputs may lead to a single output in an LTS while in FSMs inputs and outputs are paired). IOLTSs (Fernandez et al., 1996) and I/O automata (Lynch and Tuttle, 1987) are useful extensions of Q3 LTSs and automata, respectively, that allow for explicit modeling of discrete input and output actions.

Q4 p0070 *Example*. In Fig. 3, the thermostat example introduced in Section 1.3 is modeled using an I/O automaton. Two states, namely "heater_on" and "heater_off," represent the switching behavior of the system based on a temperature reading from the room (rcv_tmp_low? and rcv_tmp_high?). Intuitively, such a model is a coarse abstraction of reality. Particularly, this model does not specify the system dynamics regarding temperature and the rules governing them. Hybrid (I/O) automata, introduced next, are often used to model CPSs and their system dynamics.



f0020 **FIG. 3**

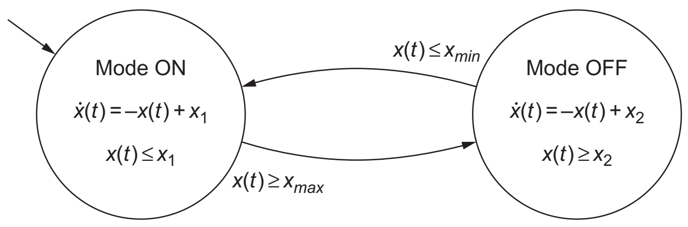
Thermostat FSM.

p0075 In order to model CPSs, it is often necessary to enrich (extended) FSMs with models of system dynamics, such as differential equations or piecewise continuous trajectories. Examples of such enriched models are hybrid automata (Alur et al., 1993) and hybrid LTSs (Cuijpers et al., 2002).

p0080 *Example*. In Fig. 4, a hybrid automaton model of the thermostat example is shown. The dynamics $\dot{x}(t)$ of each mode represent the temperature behavior in the room with the $x(t)$ variable modeling temperature and the x_1/x_2 variables being input-dependent temperature constants linked to the window position (input-dependent system). The temperature is regulated between $x_{\min}$ and $x_{\max}$ degrees leading to the corresponding guard conditions in Fig. 4. Hence, by opening and closing (to different extents) the window the dynamics of the system are affected.

s0045 2.2 LOGICAL MODELS

p0085 Logical formulae are often used to specify an abstract constraint on the behavior of dynamical systems. Such formulae often refer to qualitative or quantitative time, eg, the relative order of events or specific moments of time when certain events may occur or system dynamics may take specific values. Temporal logics are a class of logical formalisms that allow for such timed specifications (Manna and Pnueli, 1992). Q5



f0025 **FIG. 4**

Thermostat state-model with $x_{\min} \leq x(t) \leq x_{\max}$ as the configured desired room temperature, $\dot{x}(t)$ as the room (temperature) dynamics, x_1 and x_2 as temperature (input) variables that influence the dynamics and mode-dependent invariants on room temperature.

p0090 Classical temporal logics, such as Modal mu-Calculus (Kozen, 1983), computational tree logic (Emerson and Halpern, 1986), and linear temporal logic (LTL) (Pnueli, 1977) allow for specifying behavioral constraints using a qualitative notion of time. Such logics, in addition to standard logical connectives from propositional logic, provide modalities to specify, for example, that certain formulae will always hold, will hold eventually, or in the next state.

p0095 *Example*. The following LTL formula specifies that whenever the system is in the "On" mode, it will eventually reach the "Off" mode. The fact that this formula should always hold is specified by the modality $\boldsymbol{G}$; intuitively, $\boldsymbol{G}\varphi$ means that formula $\boldsymbol{\varphi}$ holds in all reachable states of the system. Logical implication is denoted by $\Rightarrow$ and eventuality is denoted by modality $\boldsymbol{F}$. Formula $\boldsymbol{F}\varphi$ denotes that $\boldsymbol{\varphi}$ will eventually hold in the future:

$$\varphi_1 = \boldsymbol{G}(\mathrm{On} \Rightarrow \boldsymbol{F}\ \mathrm{Off}).$$

p0100 The above-given formula specifies that in each state of the system, whenever On holds, eventually in the future Off will hold, ie, Off is an inevitable consequence of On.

p0105 For CPSs, it is often useful to refer to quantitative time and also to the evolution of system dynamics. Examples of temporal logics that allow for specifications with reference to quantitative time and systems dynamics are metric temporal logic (MTL) (Koymans, 1990) and signal temporal logic (STL) (Maler and Nickovic, 2004), respectively. In particular, MTL extends LTL modalities with the possibility of specifying the time interval within which the ensuing property is satisfied. Additionally, using STL one can also refer to values.

p0110 *Example*. The following MTL formula specifies that within the first 100 units of time from the start of the system, it always holds that when the system is in the "On" mode, it will be in the "Off" mode within 5 units of time (relative to the moment when On holds):

$$\varphi_2 = \boldsymbol{G}_{[0,\,100]}\left(\mathrm{On} \Rightarrow \boldsymbol{F}_{[0,\,5]}\ \mathrm{Off}\right).$$

p0115 Similarly, the following STL formula specifies that the first 100 units of time, whenever the temperature falls below a certain threshold, it will again be above the threshold within 5 units of time:

$$\varphi_2 = \boldsymbol{G}_{[0,\,100]}\left(x(t) < x_{\min} \Rightarrow \boldsymbol{F}_{[0,\,5]} x(t) \geq x_{\min}\right).$$

s0050 2.3 STATE-BASED MODELS

p0120 State-based models typically specify a system in terms of a relation between valuation of model variables before and after different operations (state transformers). Examples of state-based modeling languages include Z (Spivey, 2008), B method and Event B (Abrial, 2010), and action systems (Back, 2003). Q6 Q7 Q8

B978-0-12-803801-7.00019-5, 00019

p0125 *Example*. The following action system specifies the discrete behavior of the thermostat system:

```
p0130  Thermostat ≜ |[
p0135                  var mode*: {on, off} •
p0140                      mode := off;
p0145                  do
p0150                   rcvTmpLow = true → mode := on;   []
p0155                   rcvTmpLow = false → mode := off;
p0160                  od
p0165              ]| : rcvTmpLow
```

p0170 The syntax is selfexplanatory: first a state variable (*mode*) is declared (the asterisk marks a variable that is exported, ie, can be referenced in another action system). Subsequently, the state variable is initialized. Finally, it is specified how the state variable evolves by specifying a number of alternative guarded assignments. Finally, it is specified that *rcvTmpLow* is an imported variable.

p0175 State-based models have been extended to represent system dynamics, see, eg, (Rönkkö et al., 2003; Back et al., 2001; Banach et al., 2015). Below we specify our thermostat example in the continuous action system style (Back et al., 2001).

p0180 *Example*. The following continuous action system specifies the discrete and dynamic behavior of the thermostat system:

```
p0185  ThermostatDynamics ≜
p0190              |[
p0195                  var mode*: {on, off},
p0200                      x: $Real_{+} \to Real$   •
p0205                  mode := off;
p0210                  reset(x);
p0215                  do
p0220                   $x.\ now \le x_{min} \to mode := on;$
p0225                       $x = \lambda t\ .\ x.\ now + {(t-now)^2}/{2} + x_1(t - now);$   []
p0230                   $x.\ now \ge x_{max} \to mode := off;$
p0235                       $x = \lambda t\ .\ x.\ now - {(t-now)^2}/{2} + x_2(t - now);$
p0240                  od
p0245              ]|
```

p0250 The syntax of the above-specified action system is almost identical to the discrete specification. The main difference is that the system dynamics (variable x) is specified in terms of a trajectory (a real-valued function). The value of x is subsequently specified using lambda expressions (an expression $x = \lambda t \cdot f$ can be read as the equation $x(t) = f$).

s0055 3 CONFORMANCE RELATION

p0255 In MBT, models and implementations are either compared directly or indirectly with one another. These types of conformance are named conformance relation and conformance testing respectively.

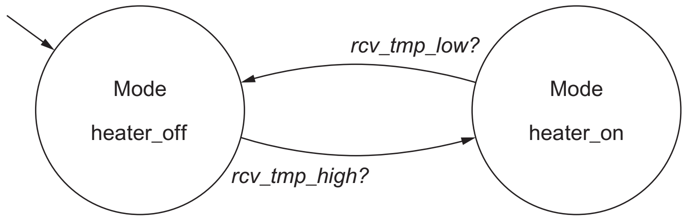
p0260 *A conformance relation is* a well-defined mathematical relation between a specification and an implementation model. For this it is essential that both the specification and implementation can be represented by formal models. Of course, the formal model underlying the implementation may not be available. Nevertheless, it is assumed that the behavior of the implementation may be represented by some (possibly unknown) model. This assumption is called the *test assumption* (ISO/IEC JTC1/SC21 WG7, ITU-T SG 10/Q.8, 1996). For discrete event systems, examples of conformance relations are defined in Lee (1996) and Tretmans (1996). Such definitions are extended to systems with data, eg, in Frantzen et al. (2006) and with time, eg, in Briones and Brinksma (2005). For surveys of such conformance relations, we refer to Hierons et al. (2009) and Broy et al. (2005).

p0265 Typically a conformance relation requires that the set of outputs produced by the implementation is a subset of those allowed by the specification. However, practically it is customary to utilize partial test models, which focus on certain aspects of the system behavior. In such a case, the conformance relation should be adapted so that it concerns only a subset of implementation's behavior, namely the subset that is included in the specification, eg, see Tretmans (1996).

p0270 In order to establish the conformance relation, a *conformance testing* technique is used. Such a technique obtains a test suite from the specification and executes the test cases in that test suite on the implementation. Subsequently, conformance testing establishes whether or not the implementation passes the test cases and the test suite (as expected). Hence, it should be clearly defined what it means for an implementation to pass a test case (and hence a test suite). A conformance testing technique is *sound* if any conforming implementation passes all the obtained test cases (and hence, the test suite) and it is *complete* when every nonconforming implementation fails at least one test case. (We refer to Fig. 2 for a schematic presentation of this process.)

p0275 *Example*. Consider the I/O automaton of the thermostat example of Fig. 3 as a specification model. An implementation model for this system is provided in Fig. 5. This implementation model does not conform to the specification since initially (assuming that "heater_on" is the initial mode), it is in mode "heater_off" while according to the specification the mode "heater_on" should be active. Also, after the rcv_tmp_high? input, the specification only allows for the "heater_off" mode while the implementation is in mode "heater_on."

p0280 In order to test CPSs, it is often essential to exploit models incorporating system dynamics and adapt the notions of conformance relation and conformance testing for such models. Involving system dynamics, one concern is how to compare the output behavior (trajectories) of specification and implementation. One may distinguish between approaches where behaviors of specification need to

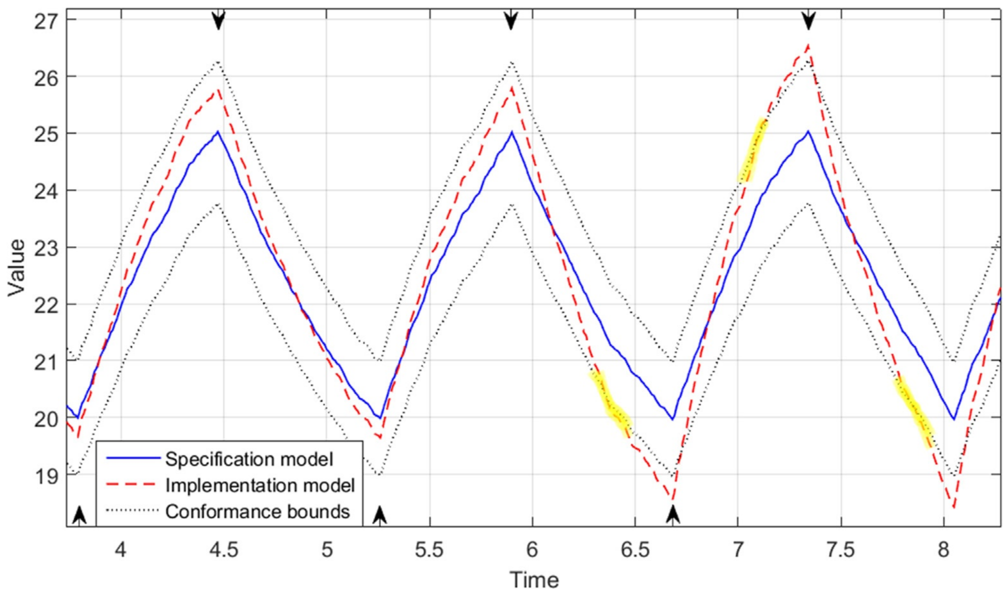


f0030 **FIG. 5**

Implementation model of a thermostat example in the form of an I/O automaton (deduced from the specification model in Fig. 3; however, note that the modes are switched).

be mimicked precisely by the implementation or only approximately. The hybrid input-output conformance relation defined by Osch (2006) is an example of the former, whereas the approximate notion of conformance proposed in Abbas et al. (2014a,b) is an example of the latter. Concerning the latter relation, a notion of closeness between two output sequences is defined that allows for deviations in value and time between outputs up to the specified values of ε and τ, respectively. The conformance degree, given some allowed time deviation τ between two systems, is defined as the minimal value for ε such that the systems are (τ, ε)-conformant. Based on this notion of closeness the authors of this chapter developed a prototype tool for MBT of CPSs (Aerts et al., 2015). In the literature one may find other works on approximate equivalence of hybrid systems, such as Girard and Pappas (2011) and Julius and Pappas (2006). We refer to Khakpour and Mousavi (2015) and Mohaqeqi et al. (2014) for a comparison of the abovementioned notions.

p0285 *Example*. Given the hybrid automaton (specification) model of the thermostat example in Fig. 4, an exemplary steady-state behavior for such a system is shown in Fig. 6 (using the solid line). In addition, there is also a steady-state behavior for a unique thermostat implementation model shown (dashed line). The output or temperature behavior of the specification model (solid line) is surrounded by conformance bounds (dotted line). The conformance bounds allow for deviations (of the implementation) from the specification in such a way that the room temperature still remains approximately between 20 and 25 degrees. In the same figure, we have indicated state transitions of the underlying hybrid automaton by solid arrowheads (upwards = "On" → "Off", downwards = "Off" → "On", see Fig. 4).



f0035 **FIG. 6**

Steady-state behavior of a thermostat example (see Fig. 4) of a specification and implementation model, visualization of a conformance relation.

p0290 As illustrated in Fig. 6, for approximately 6.4 time units, the implementation model remains within the conformance bounds of the specification model. However, beyond this point, nonconforming behavior is detected, ie, the implementation repeatedly violates the conformance bounds as indicated in Fig. 6 by the highlighted areas.

p0295 Conformance relations for CPSs may be parameterized by a conformance (robustness) degree. Such a degree specifies a bound on the deviation of the implementation from the specification, and has been defined in Abbas et al. (2014a,b) for behavioral models (akin to sampling of hybrid automata). In addition, a robustness degree for temporal logic specifications has been introduced in Donzé and Maler (2010) to measure the "satisfaction degree" of temporal logic specifications (as specification model) on implementation models. The latter is particularly useful in test-case generation and is discussed in the next section.

s0060 4 CONFORMANCE TESTING

s0065 4.1 TEST-CASE GENERATION

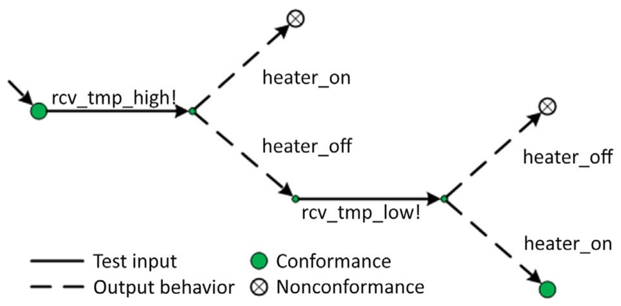
p0300 In Section 2, an overview of the modeling languages used for MBT is presented. Assuming the presence of a test model in one such modeling language, the next step in a MBT workflow is to extract a number of test cases from the model. The test cases are subsequently applied to the implementation to assign a verdict that indicates whether the implementation conforms to the specification or not (ie, whether the underlying model of the implementation is related to the specification model by the conformance relation). The process of extracting test cases is called *test-case generation*.

p0305 Two important attributes of a test-case generation algorithm are its soundness and exhaustiveness (Tretmans, 1996). *Soundness* means that a conforming implementation is never rejected. In other words, if the implementation conforms to the specification, then indeed the implementation passes all test cases that can be generated. *Exhaustiveness* of test-case generation signifies that all faulty implementations (possibly with respect to a given fault model) are rejected by some generated test case. Exhaustiveness is hard to obtain in practice, particularly without a well-defined fault model, because it usually requires an unbounded number of test cases, ie, an infinite test suite.

p0310 Given the size of the systems to which one wishes to apply MBT, manual definition of test cases is an approach that is generally too labor-intensive. A strength of MBT is that sound test-case generation algorithms are devised for various types of models, eg, Hierons et al. (2009), Broy et al. (2005), and Abbas et al. (2013).

p0315 For deterministic models, test cases are often expressed as sequences of inputs and expected outputs. For nondeterministic systems, test cases can be represented as trees. Each edge in the sequence or tree either provides an input (allowed by the specification) to the system under test and/or observes an output from the system in order to evaluate it using the allowed outputs by the specification.

p0320 *Example*. Consider the I/O automaton of Fig. 3 as a specification model, and the automaton model of Fig. 5 as an implementation model. Based on the specification model, a test tree can be generated as shown in Fig. 7. This test tree depicts the test cases for the implementation under test, and specifies conforming and nonconforming behavior.



f0040 **FIG. 7**

Test tree for the thermostat FSM example (based on Fig. 3).

p0325 In the case of the thermostat FSM example, when executing this test tree, the implementation will fail after the first test case. The implementation will be in mode “heater_on” as response to the test input rcv_tmp_high! and hence, the test case is terminated because of nonconforming behavior. Note that one can perform online and offline test-case generation, which mainly differs in the fact that online test-case generation will typically result in a sequence of test cases, ie, the next step is determined depending on the behavior observed so far, while in offline test-case generation one may have to deal with all possible outcomes of every test input (which typically results in a test tree).

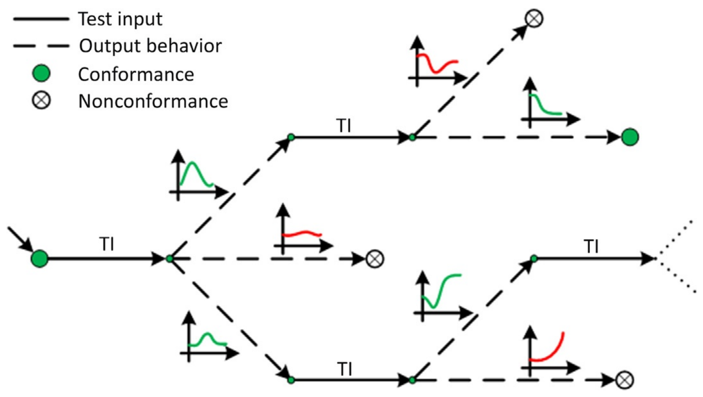
s0070 4.2 TEST-CASE SELECTION AND COVERAGE

p0330 CPS models typically have an infinite state space and hence, the sampling strategy in this infinite state space can have a major impact in the effectiveness of the generated test cases. In order to steer the sampling process, a notion of model coverage can be used. Traditional examples of such coverage metrics are branch coverage criteria in control-flow graphs and predicate and condition coverage in state-machine based models (Ammann and Offutt, 2008).

p0335 In Dang and Nahhal (2009), the authors present a test coverage measure that is based on the notion of star discrepancy (Beck and Chen, 1988). This coverage notion is used to quantify the extent to which a test suite covers the state space of the system. This coverage measure is then used to guide a method for test-case generation based on rapidly-exploring random trees (RRT) into areas of the state space where coverage is low.

p0340 For the domain of CPSs, a method to develop structural coverage based on symbolic computation of conditions on the inputs of the system (specified by means of differential dynamic logic) is to drive the specification in a certain structural region of its behavior (Zhang et al., 2013). This method uses counter-examples from model-checking algorithms to obtain test cases for specific structural regions.

p0345 In recent work by Dreossi et al. (2015) a novel approach for MBT of CPSs was proposed in which robustness analysis and coverage were combined. The authors propose an approach in which the



f0045 **FIG. 8**

Test tree for the thermostat hybrid automaton example (Fig. 4).

system under test is modeled as a (nonautonomous, discrete-time) hybrid dynamical system and the requirements imposed on the system are formalized as formulas stated in STL (Donzé and Maler, 2010; Donzé et al., 2013). The algorithm used for hybrid test-case generation is an adapted version of the previously discussed RTT algorithm. In particular, decisions on what part of the state space to explore further, that are taken randomly in the original RTT algorithm, are instead based on a robustness analysis of the satisfaction of the requirement formulas. Q9

p0350 *Example*. Given the hybrid automaton model of the thermostat example in Fig. 4, a test suite (in the form of a tree) can be generated as shown in Fig. 8. This test suite comprises a representation of a set of pass (solid) and fail (crossed) verdict nodes corresponding to correct and incorrect observed output trajectories. Note that the visualized output trajectories in Fig. 8 should not be taken literally, in fact each observed trajectory represents a subset of allowed or not allowed output trajectories. In practice testing based on these trajectories should allow for certain error margins both in timing and in the valuation of the variables in output trajectories. We refer to Abbas et al. (2013, 2014a) for examples of testing notions that accommodate such error bounds.

s0075 5 TOOLING

p0355 In order to apply MBT methodologies in practical cases, mechanized tools are required. In this section, we briefly review the existing tools that can be used to mechanize different stages of MBT. There is ample room for more mature tooling in this area and we hope to see more stable and robust tools that are based on published research results in this area.

p0360 One of the few test-case generation tools for hybrid systems is HTG (Dang, 2011). In this tool, hybrid automata models or SPICE netlists serve as input for coverage-guided test-case generation based on the star discrepancy notion from statistics and the RRTs algorithm from robotic motion planning. The generated test cases can be visualized with viewers such as Matlab or the GNU plot tool. Q10

p0365 Another tool that includes the conformance testing functionality as explained in Section 3, is S-TaLiRo (Annpureddy et al., 2011). This tooling, designed in Matlab, utilizes arbitrary Matlab or Simulink models in order to perform conformance testing using stochastic techniques or Monte-Carlo methods for black-box test-case generation (no model knowledge is used). Hence, only coverage metrics defined over the input space are applicable. Moreover, the tooling is also compatible with logical models, in specific the falsification of MTL formulae on implementation models.

p0370 The Matlab toolbox RRT-REX (for RRT Robustness-guided EXplorer) (Dreossi et al., 2015) implements a "white-box testing" approach for logical models specified in terms of STL formulae (ie, model knowledge is used for the test-case generation process). This method, as explained in Section 4, guarantees a global minimum of the robustness degree of its temporal logic formulae. However, at the moment of writing, the current version of the tooling does not incorporate conformance analysis.

p0375 Regarding conformance testing of CPSs, only a few tools or toolboxes exist. One example worth mentioning is a tool prototype developed in Matlab (Aerts et al., 2015) to work with CPS models (and implementations) in Acumen (Taha et al., 2012). In this tool, the authors implemented a modified version of the conformance notion as presented in Abbas et al. (2014b), and proposed a white-box test-case generation algorithm based on hybrid automata models for CPS.

p0380 In order to provide a comparison of these tools, an overview can be found in Table 1, which lists all of the above tools and their respective features.

p0385 Regarding model-verification, several tools exists. Here, we briefly mention three examples of such tools. Firstly, the tool "SpaceEx" (Frehse et al., 2011) serves as a verification tool for continuous as well as hybrid dynamical systems. Secondly, "Strong" (Deng et al., 2013) is a trajectory-based verification toolbox for hybrid systems in Matlab, which utilizes a combination of simulation and formal verification algorithms. Lastly, "Breach" (Donzé, 2010) is a Matlab/C++ toolbox for simulation of hybrid systems, verification of temporal logic properties on hybrid systems, and reachability analysis of dynamical systems represented by ordinary differential equations or by external tools such as Matlab Simulink.

t0010 **Table 1 Tool Overview**

		Tools			
		HTG	**RRT-REX**	**S-TaLiRo**	**"Tool Prototype"**
1	Conformance testing	No	No	Yes	Yes
2	Temporal logic falsification	No	Yes	Yes	No
3	White-box test generation	Yes	Yes	No	Yes
4	Black-box test generation	No	No	Yes	No

s0080 ## 6 CHALLENGES

p0390 Several challenges lie ahead regarding the applicability of MBT of CPSs. We categorize such challenges into theoretical challenges and practical applicability challenges; we refer to Khakpour and Mousavi (2015) for a more detailed description of these challenges.

p0395 Regarding theory, despite the existing literature, we still need more research concerning scalable conformance relations and conformance testing methods. Below we list a few challenging issues in designing such a conformance relation and conformance testing method:

- u0010 Component-wise and step-wise conformance: existing conformance relations are mostly noncompositional, ie, composition of components that conform to their respective specifications may result in a system that does not conform to the composition of the specifications. Also many of the existing conformance relations are not transitive and hence, do not provide a natural means for step-wise refinement.
- u0015 Expressiveness of models: often conformance relations are defined in terms of very restrictive modeling assumptions. For example, models are assumed to be deterministic or full (ie, at all states and all possible inputs are defined).
- u0020 Verified and robust discretization: for practical implementations inputs and outputs are discretized samples of their corresponding continuous trajectories in the model. There is very little work on providing verified discretization schemes that do not miss the relevant events and cover the model with respect to a given notion of coverage.

p0415 Applicability of sound scientific theories involves the integration of these scientific methods and techniques in stable and well-supported tools. If such tools are present, the proposed framework in Section 1, where MBT is positioned in a V-model SDLC, can be applied to adopt such an integrated MBT process. We envisage the following issues regarding practical applicability and tooling:

- u0025 User-friendliness: Tools should be intuitive to use for engineers of various backgrounds. Therefore, a comprehensible user interface and a simple input language should be present in which the tool configuration and operation can take place, including the specification of (formal) specifications. Up-to-date models are hardly present in practice and also often engineers do not feel comfortable with the abstractions in generic modeling formalisms. Hence, domain-specific languages can play an important role in developing models for such tools.
- u0030 Accuracy: There is a gap between mathematical theories and actual implementations of systems dynamic models. Having a reliable and verified basis for the calculations of numeric approximations of systems dynamics is an important challenge in tools for MBT.
- u0035 Real-time adapters: MBT tools often produce abstract test cases that need to be transformed into concrete test cases using the right data formats and protocols. Such adapters are often very complex and verifying their correctness is not a trivial task. Hence, providing reliable techniques for developing or extending adapters is an important issue in tooling. Accurate test execution and observations of system dynamics also requires real-time adapters with an accurate approximation of time, which is an extra challenge in tooling.
- u0040 Traceability: Many certification trajectories and standards (such as the ISO26262 for automotive and ARP-4761 and DO-178C/ED-12C for avionic systems) require traceability among

certification artifacts (eg, from requirements to test cases). The traceability path passes through models in MBT and requires additional machinery in the tooling.

u0045 • White-box versus black-box testing: MBT is often used as a black-box testing method, ie, the structural information of the model and the implementation are often not exploited in the process of test-case generation and execution. However, white-box (glass-box) testing is a viable alternative and can potentially improve the effectiveness of the test process in order to avoid gaps and redundancies. Further research is required in order to build in such techniques in the MBT trajectory of CPSs.

REFERENCES

Aarts, F., Jonsson, B., Uijen, J., Vaandrager, F., 2015. Generating models of infinite-state communication protocols using regular inference with abstraction. Form. Method. Syst. Des. 46 (1), 1–41.

Abbas, H., Fainekos, G., Sankaranarayanan, S., Ivancic, F., Gupta, A., 2013. Probabilistic temporal logic falsification of cyber-physical systems. ACM Trans. Embed. Comput. Syst. 12 (S2), 1–95. 30.

Abbas, H., Hoxha, B., Fainekos, G., Deshmukh, J., Kapinski, J., Ueda, K., 2014a. Conformance testing as falsification for cyber-physical systems. arXiv. preprint arXiv:1401.5200.

Abbas, H., Mittelmann, H., Fainekos, G., 2014b. Formal property verification in a conformance testing framework. In: IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE), 2014 Twelfth ACM. IEEE, pp. 155–164.

Aerts, A., Mousavi, M., Reniers, M., 2015. A tool prototype for model-based testing of cyber-physical systems. In: Proceedings of the 12th International Colloquium on Theoretical Aspects of Computing (ICTAC 2015). Springer, Berlin.

Alur, R., Costas, C., Henzinger, T., Ho, P., 1993. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In: Hybrid Systems, vol. 736. Springer, Berlin, Heidelberg, pp. 209–229.

Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.-H., Nicollin, X., et al., 1995. The algorithmic analysis of hybrid systems. Theor. Comput. Sci. 138 (1), 3–34.

Ammann, P., Offutt, J., 2008. Introduction to Software Testing. Cambridge University Press, Cambridge.

Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S., 2011. S-TaLiRo: a tool for temporal logic falsification for hybrid systems. In: Aziz Abdulla, P., Rusten, K., Leino, M. (Eds.), Tools and Algorithms for the Construction and Analysis of Systems. Springer, Berlin, pp. 254–257. Q11

Back, R.-J., Petre, L., Porres, I., 2001. Continuous action systems as a model for hybrid systems. Nord. J. Comput. 8 (1), 2–21.

Baier, C., Katoen, J., 2008. Principles of Model Checking. MIT Press, Cambridge, MA.

Banach, R., Butler, M., Qin, S., Verma, N., Zhu, H., 2015. Core hybrid event-B I: single hybrid event-B machines. Sci. Comput. Program. 105, 92–123.

Beck, J., Chen, W., 1988. Irregularities of Distribution. Cambridge University Press, Cambridge.

Blanchard, B.S., Fabrycky, W.J., Fabrycky, W.J., 1990. Systems Engineering and Analysis. vol. 4, Prentice Hall, Englewood Cliffs, NJ.

Boehm, B., 1981. Software Engineering Economics. Prentice Hall, Englewood Cliffs, NJ.

Briones, L., Brinksma, E., 2005. A test generation framework for quiescent real-time systems. In: Formal Approaches to Software Testing—Proceedings of the 4th International Workshop (FATES 2004). Springer, Berlin, Heidelberg.

Broy, M., Jonsson, B., Katoen, J., Leucker, M., Pretschner, A., 2005. Model-Based Testing of Reactive Systems: Advanced Lectures (vol. 3472 of Lecture Notes in Computer Science). Springer, Berlin.

Cuijpers, P., Reniers, M., Heemels, W., 2002. Hybrid Transition Systems. TU/e, Eindhoven.

Dang, T., 2011. Model-based testing of hybrid systems. In: Zander, J., Schieferdecker, I., Mosterman, P.J. (Eds.), Model-Based Testing for Embedded Systems. CRC Press, Boca Raton, FL, pp. 383–424.

Dang, T., Nahhal, T., 2009. Coverage-guided test generation for continuous. Form. Method. Syst. Des. 34 (2), 183–213.

Deng, Y., Rajhans, A., Julius, A., 2013. STRONG: a trajectory-based verification toolbox for hybrid systems. In: Joshi, K., Siegle, M., Stoelinga, M., D'Argenio, P. (Eds.), Proceedings of the 10th International Conference on Quantitative Evaluation of Systems (QEST 2013). Springer, Berlin, Heidelberg, pp. 165–168.

Donzé, A., 2010. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P. (Eds.), Proceedings of the 22nd International Conference on Computer Aided Verification (CAV 2010). Springer, Berlin, Heidelberg, pp. 167–170.

Donzé, A., Maler, O., 2010. Robust satisfaction of temporal logic over real-valued signals. In: Proceedings of the 8th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2010). Springer, Berlin, Heidelberg, pp. 92–106.

Donzé, A., Ferrère, T., Maler, O., 2013. Efficient robust monitoring for STL. In: Proceedings of the 25th International Conference on Computer Aided Verification (CAV 2013). Springer, Berlin, Heidelberg, pp. 264–279.

Dreossi, T., Dang, T., Donzé, A., Kapinski, J., Jin, X., Deshmukh, J.V., 2015. Efficient guiding strategies for testing of temporal properties of hybrid systems. In: Proceedings of the 7th International Symposium on NASA Formal Methods (NFM 2015). Springer International, pp. 127–142.

Emerson, A.E., Halpern, J.Y., 1986. "Sometimes" and "not never" revisited: on branching versus linear time temporal logic. J. ACM 33 (1), 151–178.

Estefan, J.A., et al., 2007. Survey of Model-Based Systems Engineering (MBSE) Methodologies, vol. 25. Incose MBSE Focus Group. Q12

Fainekos, G.E., Pappas, G.J., 2006. Robustness of temporal logic specifications. In: Havelund, K., Núñez, M., Roşu, G., Wolff, B. (Eds.), Formal Approaches to Software Testing and Runtime Verification. Springer, Berlin, pp. 178–192. Q13

Fainekos, G.E., Anand, M., Lee, I., Pappas, G.J., 2007. Robust test generation and coverage for hybrid systems. In: Proceedings of the 10th International Workshop on Hybrid Systems: Computation and Control (HSCC 2007). Springer, Berlin, Heidelberg, pp. 329–342. Q14

Fernandez, J., Jard, C., Jéron, T., Viho, C., 1996. Using on-the-fly verification techniques for the generation of test suites. In: Proceedings of the 8th International Conference on Computer Aided Verification (CAV96). Springer, Berlin, pp. 348–359.

Firesmith, D., 2014. Common System and Software Testing Pitfalls: How to Prevent and Mitigate Them: Descriptions, Symptoms, Consequences, Causes, and Recommendations. Addison-Wesley Professional. Q15

Frantzen, L., Tretmans, J., Willemse, T.A., 2006. Symbolic framework for model-based testing. In: Proc. of FATES/RV 2006. Lecture Notes in Computer Science, vol. 4262. Springer, Berlin, pp. 40–54. Q16

Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., et al., 2011. SpaceEx: scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (Eds.), Proceedings of the 23rd International Conference on Computer Aided Verification (CAV 2011). Springer, Berlin, pp. 379–395.

Friedenthal, S., Moore, A., Steiner, R., 2014. A Practical Guide to SysML: The Systems Modeling Language. Morgan Kaufmann, San Francisco, CA.

Girard, A., Pappas, G.J., 2007. Approximation metrics for discrete and continuous systems. IEEE Trans. Autom. Control 52 (5), 782–798. Q17

Girard, A., Pappas, G., 2011. Approximate bisimulation: a bridge between computer science and control theory. Eur. J. Control. 17 (5–6), 568–578.

Hierons, R., Bogdanov, K., Bowen, J., Cleaveland, R., Derrick, J., Dick, J., et al., 2009. Using formal specifications to support testing. ACM Comput. Surv. 41 (2), 1–76.

ISO/IEC JTC1/SC21 WG7, ITU-T SG 10/Q.8, 1996. Information retrieval, transfer and management for OSI; framework: formal methods in conformance testing. ITU-T.
Julius, A., Pappas, G., 2006. Approximate equivalence and approximate synchronization of metric transition systems. In: 2006 45th IEEE Conference on Decision and Control. IEEE, pp. 905–910.
Khakpour, N., Mousavi, M., 2015. Notions of conformance testing for cyber-physical systems: overview and roadmap. In: Proceedings of the 26th International Conference on Concurrency Theory (CONCUR 2015), vol. 42. LIPIcs–Leibniz International Proceedings in Informatics, Madrid, Spain, pp. 18–40.
Koymans, R., 1990. Specifying real-time properties with metric temporal logic. Real-Time Syst. 2 (4), 255–299.
Kozen, D., 1983. Results on the propositional mu-calculus. Theor. Comput. Sci. 27 (3), 333–354.
Lee, D.a., 1996. Principles and methods of testing finite-state machines. Proc. IEEE 84 (8), 1089–1123.
Lynch, N.A., Tuttle, M.R., 1987. Hierarchical correctness proofs for distributed algorithms. In: Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing. ACM, New York, pp. 137–151.
Maler, O., Nickovic, D., 2004. Monitoring temporal properties of continuous signals. In: Proceedings of the Joint International Conferences on Formal Modeling and Analysis of Timed Systems (FORMATS 2004) and Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT 2004). Springer, Berlin, pp. 152–166.
Manna, Z., Pnueli, A., 1992. The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer, New York. Q18
Meinke, K., Niu, F., Sindhu, M.A., 2012. Learning-based software testing: a tutorial. In: International Workshops on Leveraging Applications of Formal Methods, Verification, and Validation, vol. 336. Springer, Berlin, pp. 200–219.
Modelica Association et al., 2005. Modelica—a unified object-oriented language for physical systems modeling. Language Specification, Version 2.
Mohaqeqi, M., Mousavi, M., Taha, W., 2014. Conformance testing of cyber-physical systems: a comparative study. In: Proceedings of the 14th International Workshop on Automated Verification of Critical Systems (AVOCS 2014), vol. 70.
Object Management Group, 2014. OMG Systems Modeling Language (OMG SysML™), V1.4. Opgehaald van http://www.omg.org/spec/SysML/1.4/PDF.
Osch, M.V., 2006. Hybrid input-output conformance and test generation. In: Formal Approaches to Software Testing and Runtime Verification. Springer, Berlin, pp. 70–84.
Pnueli, A., 1977. The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science. IEEE, pp. 46–57.
Rönkkö, M., Ravn, A., Sere, K., 2003. Hybrid action systems. Theor. Comput. Sci. 290 (1), 937–973.
Taha, W., Brauner, P., Zeng, Y., Cartwright, R., Gaspes, V., Ames, A., et al., 2012. A core language for executable models of cyber physical systems (preliminary report). In: Proceedings of the 32nd International Conference on Distributed Computing Systems Workshops (ICDCSW 2012). IEEE CS, Macau, China, pp. 303–308.
Tretmans, J., 1996. Test generation with inputs, outputs and repetitive quiescence. Softw. Concepts Tools 17, 103–120.
Uselton, A.C., Smolka, S.A., 1994. A compositional semantics for statecharts using labeled transition systems. In: Proceedings of the 5th International Conference on Concurrency Theory (CONCUR ’94). Springer, Berlin, Heidelberg, pp. 2–17.
Utting, M., Pretschner, A., Legeard, B., 2012. A taxonomy of model-based testing approaches. Softw. Test. Verif. Reliab. 22 (5), 297–312.
Vishal, V., Kovacioglu, M., Kherazi, R., Mousavi, M., 2012. Integrating model-based and constraint-based testing using SpecExplorer. In: Proceedings of the 4th Workshop on Model-based Testing in Practice (MoTiP 2012). IEEE CS, Dallas, pp. 219–224.
Zhang, L., He, J., Yu, W., 2013. Test case generation from formal models of cyber physical system. Int. J. Hybr. Inform. Technol. 6 (3), 15–24.

Non-Print Items

Abstract

Cyber-physical systems (CPSs) are the result of the integration of connected computer systems with the physical world. They feature complex interactions that go beyond traditional communication schemes and protocols in computer systems. One distinguished feature of such complex interactions is the tight coupling between discrete and continuous interactions, captured by hybrid system models.

Due to the complexity of CPSs, providing rigorous and model-based analysis methods and tools for verifying correctness of such systems is of the utmost importance. Model-based testing (MBT) is one such verification technique that can be used for checking the conformance of an implementation of a system to its specification (model).

In this chapter, we first review the main concepts and techniques in MBT. Subsequently, we review the most common modeling formalisms for CPSs, with focus on hybrid system models. Subsequently, we provide a brief overview of conformance relations and conformance testing techniques for CPSs.

Keywords: Cyber-physical systems, V-model, Model-based testing, Conformance, Test-case generation, Test coverage