Conformance Testing for Cyber-Physical Systems

MATTHIAS WOEHRLE, KAI LAMPKA, and LOTHAR THIELE, ETH Zurich

Cyber-Physical Systems (CPS) require a high degree of reliability and robustness. Hence it is important to assert their correctness with respect to extra-functional properties, like power consumption, temperature, etc. In turn the physical quantities may be exploited for assessing system implementations. This article develops a methodology for utilizing measurements of physical quantities for testing the conformance of a running CPS with respect to a formal description of its required behavior allowing to uncover defects. We present foundations and implementations of this approach and demonstrate its usefulness by conformance testing power measurements of a wireless sensor node with a formal model of its power consumption.

Categories and Subject Descriptors: D.2.5 [**Software Engineering**]: Testing and Debugging—*Testing tools*

General Terms: Algorithms, Verification

Additional Key Words and Phrases: Conformance test, timed testing, cyber-physical systems

ACM Reference Format:
Woehrle, M., Lampka, K., and Thiele, L. 2012. Conformance testing for cyber-physical systems. ACM Trans. Embedd. Comput. Syst. 11, 4, Article 84 (December 2012), 23 pages.
DOI = 10.1145/2362336.2362351 http://doi.acm.org/10.1145/2362336.2362351

1. INTRODUCTION

Cyber-Physical Systems (CPS) are computation systems deeply integrated into and interacting with the environment and its physical processes. Commonly, these systems require a high degree of reliability and robustness due to uncertainties in their physical environment. Hence, one of the main challenges for designing and implementing CPSs is the assertion of their correctness. Correctness does not only encompass algorithmic and functional aspects but also extra-functional properties that are closely related to the inherent interaction with the system environment. Examples of such properties may be temperature, power consumption, and timing. On the other hand, this additional observable behavior of a CPS may also help in assessing the overall correctness as computations not only consume and provide data but also have side-effects that can be exploited. The article addresses this challenge and presents a methodology for utilizing nonintrusive measurements of a physical quantity obtained from a CPS to identify defects in the implementation. This conformance testing between formal specification and implementation is in fact an important issue: (abstract) system designs can carefully be studied on the basis of formal methods. However, the actual implementation is built manually, not automatically generated from the specification model, such that their conformance with a predefined formal model is unknown. Consider an

The work presented here was supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant no. 5005-67322.
Authors' addresses: M. Woehrle, K. Lampka, and L. Thiele, ETH Zurich, Computer Engineering and Networks Laboratory (TIK), ETZ G 75, Gloriastrasse 35,8092 Zurich, Switzerland.

DOI 10.1145/2362336.2362351 http://doi.acm.org/10.1145/2362336.2362351

implementation of a wireless sensor node and measured traces of power consumption. We want to use the measurements of the actual (heterogeneous) hardware to assure that our previous modeling and the energy-efficient design of the system actually hold for the implementation. Visual inspection of such traces or the use of reference traces are not suitable for the large number of tests required for analyzing the various properties of the software in different test environments. Our aim is to exploit formal methods for an automated approach. This is useful since we need to compare the measured trace against all possible traces that can be produced by the specification.

Formal state-based analysis techniques have shown to be of great value when it comes to the verification of systems. However, the expansion of all possible system behaviors may yield an exponential number of system states with respect to a system's concurrently executed activities. This problem, the well-known state space explosion problem, worsens when timing and physical quantities from a continuous domain have to be considered. Hence physical quantities are commonly excluded from any verification process, which is problematic as they are important for guaranteeing correct behavior of CPSs. Nevertheless, formal methods support an expressive, concise, and often compositional formulation of expected system behavior. It is the goal of this article to present a *scalable approach* for detecting implementation defects of a CPS by exploiting some unobtrusively measured physical quantity.

The proposed technique employs formal, state-based models for specifying the expected behavior and for representing a series of measurements of a physical quantity. This approach allows to investigate the conformance of expected and observed behavior by using a timed model checker. The failure of the conformance test provides a diagnostic (debugging) trace to the test engineer, which helps in debugging implementation errors of a CPS's hard- and software. It is interesting to note that the proposed approach is bidirectional. The values of the physical quantity may be outputs produced by the CPS or they may be inputs to the CPS stimulating some behavior or a combination of the two cases.

As we are dealing with complex hard- and software systems deeply integrated into the environment the presented approach has to cope with the following challenges.

(1) As standard instrumentation techniques can falsify the system behavior, nonintrusive techniques are preferable. However, with such techniques the concrete state of the CPS is hidden and cannot be observed directly, for example the current mode of operation.
(2) The individual hardware components may react/contribute differently to the observed quantity, but it is only the overall effect that can be seen. For example, components may have different power modi, but it is only the sum of the individual power consumptions that can be measured.
(3) The observed physical properties are the result of the interaction between the CPS and its environment. This interaction needs to be part of the underlying system model and increases the complexity of the conformance test.
(4) The complexity of systems yields a high-degree of nondeterminism ruling out an exhaustive analysis due to the notorious state space explosion problem.
(5) As with all physical observations, measurements are subject to uncertainty and measurement inaccuracy.

To deal with these aspects, we devise a conformance test, i.e., for a series of measurements and a modeled system, both given as (timed) automata. One central concept is the mapping of physical quantities to a set of distinct, finite intervals in order to: (a) reduce the computational complexity of the conformance test and (b) to enable the use of existing tools that are tailored towards value-discrete timed models, such as

Uppaal [Bengtsson and Yi 2004; Behrmann et al. 2004] and TRON [Larsen et al. 2004; Hessel et al. 2008].

The main contribution of this work can be summarized as follows.

—We present the new approach for automatic conformance testing of CPSs based on timed automata.
—We present an efficient modeling for the composition of physical measurements and a system specification and discuss optimizations towards computational efficiency that are required when dealing with measurements from a real system.
—We demonstrate the feasibility of the conformance tests by testing measured power traces of an implementation of a complex CPS.
—We investigate the computational efficiency of our approach for different verification tools, Uppaal and TRON.

We continue with a presentation of the theoretical background before detailing on our method in Section 3. Section 4 discusses a second approach of our method using an online testing tool. We present a wireless sensor node application in Section 5 and show the results of conformance tests of power consumption measurements in Section 6. We conclude with a summary, including related work and a discussion of other approaches.

2. BACKGROUND THEORY

In the following, we clarify the required notations and theoretical concepts.

Definition 1 (*Timed Trace*). A *timed action* is a pair (t, a) where a is some label and $t \in \mathbb{R}_{\geq 0}$ some nonnegative timestamp. A *timed trace* $\Pi := (t_1, a_1); (t_2, a_2); (t_3, a_3); \ldots$ is a sequence of timed actions ordered by increasing timestamps, with $t_i \leq t_{i+1}$ for $i \in \mathbb{N}$.

Timed automata are used as a formal model in the proposed method. In particular, this work focuses on timed automata as used in the Uppaal model checker and follows the notation of the corresponding literature [Bengtsson and Yi 2004; Behrmann et al. 2004].

Definition 2 (*Timed Automaton Extended with Variables*). A *timed automaton extended with variables* is a tuple $TA = (Loc, l_0, Act, C, V, \hookrightarrow, I)$ where:

—Loc is a finite set of locations.
—$l_0 \in Loc$ is the initial location.
—Act is a set of actions including the internal, unobservable action τ.
—C is a finite set of clocks.
—V is a finite set of (discrete) variables.
—$\hookrightarrow \subseteq Loc \times ClockCons(C) \times VarCons(V) \times Act \times 2^C \times F \times Loc$ is an edge relation, where $ClockCons$ is a set of constraints on clocks and $VarCons$ is a set of constraints on (discrete) variables. These constraints on edges are denoted as guards. F is a set of edge-specific valuation functions on variables.
—I: $Loc \rightarrow ClockCons(C) \times VarCons(V)$ is an invariant assignment function.

Clock and variable constraints are conjunctions of atomic guards of the form $x \bowtie n, x \in C \cup V, n \in \mathbb{N}_0$ where $\bowtie \in \{<, \leq, >, \geq, =\}$. Clocks are assigned to real values using a *valuation function* $u : C \rightarrow \mathbb{R}_{\geq 0}$. Clocks implicitly increase their value as time progresses. $u+d$ denotes that each clock $x \in C$ is mapped to the value $u(x)+d$, i.e., time is increased by $d \in \mathbb{R}_{\geq 0}$. All clocks in the system increase at the same rate. Clocks can only be inspected or reset denoted by $u' = [r \rightarrow 0]u$, which signifies resetting the clocks in $r \subseteq C$. All other (not resetted) clocks agree with the valuation u, i.e., $u' = u$ for all clocks $C \setminus r$. Variable valuations are determined by an edge-specific valuation function $f_e \in F$ with $f_e : \mathbb{D}^{|V|} \rightarrow \mathbb{D}^{|V|}$, where $\mathbb{D}$ is the finite domain of the discrete variables:

$\mathbb{D} \subset \mathbb{N}_0$. Variables are updated on (discrete) transitions, i.e., $v' = f_e(v)$, where v is a vector of variables before the update and v' the corresponding variable vector after the update. Unless specifically indicated, we always refer to a valuation of a clock or discrete variable instead of the variable itself. Furthermore, $u \in g_c$ denotes that a clock valuation u satisfies a clock constraint $g_c \in \mathit{ClockCons}$; analogously $v \in g_v$ states that a variable valuation v satisfies a constraint $g_v \in \mathit{VarCons}$. We also write $(u, v) \in I(l)$ with $l \in \mathit{Loc}$ to state that the valuations of clocks and variables satisfy location invariant $I(l)$. Note that here a set notation on the valuation is used for the following reason: Invariant and guards are constraints on clocks (or variables) that specify a set of valuations allowed in a given location. Hence, a clock (or variable) valuation is valid if it is included in the set of allowed valuations. This notation is used for consistency with literature [Bengtsson and Yi 2004; Behrmann et al. 2004].

The active location of a (single) timed automaton is the location wherein the execution of the automaton (currently) resides. A state in a timed automaton is defined by an active location and clock and variable valuations $\langle l, u, v\rangle$. The transition relation $\mathcal{T}$ as induced by the edge relation and the advance of time in timed automata can be defined as follows.

Definition 3 (*Transition System of a Timed Automaton*). A transition system $\mathcal{T}'$ as induced by a TA is a tuple $(\mathcal{R}, \rightarrow)$, where:

—$\mathcal{R}$ is the set of (reachable) states; each defined by a triple $\langle l, u, v\rangle$ with $l \in \mathit{Loc}$ as the active location, $u \in \mathbb{R}_{\geq 0}$ as clock valuation, and $v \in \mathbb{D}^{|V|}$ as vector of variable evaluations.

—$\rightarrow \subseteq \mathcal{R} \times (\mathit{Act} \cup \mathbb{R}_{\geq 0}) \times \mathcal{R}$ is a(n) (infinite) transition relation. Its elements are either delay or discrete (edge-induced) transitions (as discussed shortly).

With delay transitions in the preceding definition we refer to the fact that the timed automaton remains in its (currently active) location and time passes. With discrete transitions we refer to the situation that the timed automaton changes its location, i.e., the traversal of an enabled edge takes place which is instantaneously followed by a reset of clocks and an update of variables; an enabled edge is an edge where the clock and variable constraints g_c and g_v, respectively, are satisfied by the current clock and variable valuations. Formally, these two types of transitions are defined as follows.

—*delay transition*

$$\langle l, u, v\rangle \xrightarrow{d} \langle l, u + d, v\rangle \text{ } \mathit{if} \text{ } \forall d' \in \mathbb{R}_{\geq 0} : 0 \leq d' \leq d \text{ implies } (u + d', v) \in I(l)$$

—*discrete transition*

$$\langle l, u, v\rangle \xrightarrow{\alpha} \langle l', u', v'\rangle \text{ } \mathit{if} \text{ } l \xrightarrow{g_c, g_v, \alpha, r, f_e} l' \text{ } \mathit{and} \text{ } u \in g_c \text{ } \mathit{and} \text{ } v \in g_v \text{ } \mathit{and} \text{ } u' = [r \rightarrow 0]u$$
$$\mathit{and} \text{ } v' = f_e(v) \text{ } \mathit{and} \text{ } (u', v') \in I(l'),$$

where $g_c \in \mathit{ClockCons}(C), g_v \in \mathit{VarCons}(V), \alpha \subseteq \mathit{Act}, r \subseteq C, f_e \in F$ as specified before. As a discrete transition is induced by the traversal of an enabled edge of a timed automaton, we will also speak of *executing an edge*, rather than of its traversal.

Delays may be sampled from intervals of $\mathbb{R}_{\geq 0}$, hence the previous transition rules induce an infinite transition system. The infinite set of reachable states was already denoted by us $\mathcal{R}'$, whereas we used the symbol $\rightarrow$ for referring to the infinite transition relation as induced by a timed automaton. However, these sets can be mapped to finite quotient systems $\mathcal{R}$ and $\Rightarrow$, yielding a finite transition system $\mathcal{T}$. On the basis of this finite structure state reachability for timed automata can be decided in finite time; one solely needs to generate the $|\mathcal{R}|$ different system configurations. The obtained transition system is commonly denoted as region graph. For further details on this concept the reader is referred to Alur and Dill [1994].

As the presented approach emphasizes a compositional modeling style, it is required to extend the basic concept to networks of (cooperating) timed automata, where the individual TA are denoted as component automata in the following. The clocks of the individual component automata increase all at the same rate. In the following, a brief overview of the related concepts follows; more details can be found in the Uppaal tutorials [Bengtsson and Yi 2004; Behrmann et al. 2004].

—*Cooperation via shared variables*. Variables can be declared on the level of a network of timed automata, allowing the individual timed automaton to read and manipulate them.

—*Rendezvous mechanisms*. Uppaal implements different mechanisms allowing to jointly traverse (or execute) edges within the different component automata, where such a joint execution is denoted as synchronization. By following Uppaal's nomenclature, the terms channels and signals, as well as sender and receiver are used in the following.

A channel is a unique edge label. Once it occurs in combination with an exclamation mark one speaks of a sending edge. Its occurrence with a question mark is referred to as a receiving edge. The channel of a jointly executed sending and receiving edge must match, where only a single sending edge but a varying number of receiving edges can take part in a synchronization. Before clarifying details with respect to the number of participating receivers, it is important to note that only one edge per component automaton can be executed. Updates on sending edges are performed before updates on receiving edges. The order of the receiving edges is undefined. The following concepts need to be distinguished:

Binary synchronization. One sending and one receiving timed automaton synchronize on the joint execution of dedicated edges: one from the sender, whose edge is labeled by a `channel_id` and an exclamation mark, and one from the receiver, whose edge is labeled by the same `channel_id`, but extended with a question mark (see the `on!` and `on?`-labeled edges in the timed automata of Figure 5 and 6). The pairs of enabled sending and receiving edges belonging to the same channel are selected nondeterministically, i.e., all possible synchronization pairs are considered in the state space exploration.

Broadcast channels. A single sender synchronizes with up to n receivers. This refers to the situation where one timed automaton executes a sending edge, which can be understood as the emission of a signal and where 0 to n receivers execute a receiving edge, which can be interpreted as the instantaneous reception of this broadcast signal. A broadcast requires that each timed automaton that contains (one or more) enabled receiving edges has to execute one of these edges.

Uppaal features the concept of urgent locations. Within urgent locations no time passes. Thus the system has to execute any of its outgoing edge in zero time once the urgent location is entered. If this is not possible, the execution of the timed automaton deadlocks. Figure 5 shows Uppaal urgent locations that are marked with a `'U'` and an initial location that is marked with a concentric circle.

Composed timed automata do not share clocks. Updates of global variables on synchronized edges require special care; they are evaluated sequentially: first the updates on the sending edge are performed, then the one(s) on the receiving edge(s). In case of updates on multiple receiving edges, the resulting update is not well-defined unless the operation is commutative, e.g., incrementing a variable. As the presented conformance testing approach deals with networks of timed automata that are jointly executed, a state of the system can be uniquely defined by a vector of location identifiers, each

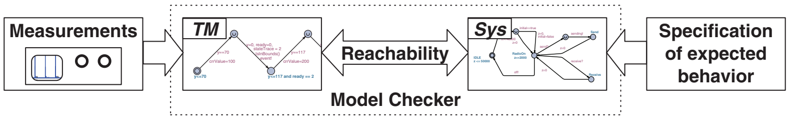

Fig. 1. TTQ overview: TTQ uses: (1) a model of the trace of measurements of a physical quantity (*TM*) and (2) a model of the system (*Sys*). On the composed model *TM*||*Sys*, reachability is checked using a model checker.

referring to the active location held by the respective automaton, and the valuation of all clocks and the vector of values held by the variables. The notation $(\vec{s}, u, v)$ is used for the elements of the set of reachable states $\mathcal{R}$, where $\vec{s}$ refers to the vector of active locations, u refers to the representation of the clock valuations, and v to the values currently held by the variables.

3. THE TTQ-APPROACH

The main goal of this section is to present Timed Testing of a physical Quantity (TTQ). TTQ employs a reachability query in order to decide whether a (finite) timed trace of measurements of a physical quantity is included in the traces of a specification of a CPS. For being applicable in an industrial environment the approach is designed in such a way that it can be implemented on the basis of standard real-time tools. The proposed methodology is illustrated in Figure 1: It relies on models of a timed trace of measurements of a physical quantity and a user-defined model that describes the specification of the system under evaluation. Whereas the model of the timed trace of measurements *TM* is automatically derived from the time series of measurements, the formal model of system behavior *Sys* needs to be (manually) generated from some specification. Having formal models for both the measurements and the expected system behavior, we can specify a conformance test. Both models are composed into a network of timed automata, denoted as *Sys*||*TM*. We formulate the conformance test as a reachability check on the jointly executed network of timed automata

$$Sys \models TM \Leftrightarrow (\vec{s}, u, v) \in \mathcal{R}_{Sys||TM}, \text{where } \vec{s} \text{ contains } l^f_{TM},$$

where l^f_{TM} denotes the final location of *TM* and $\mathcal{R}_{Sys||TM}$ is the set of reachable states for *Sys*||*TM*. Intuitively, l^f_{TM} corresponds to the final measurement in a trace. In order to automatically determine reachability on the composed model, TTQ utilizes a timed model checker. For representing the time series of measurements a suitable model is required, such that: (i) the model provides sufficient expressiveness with respect to the tracked measurements and (ii) the model allows an efficient generation of the set of reachable states of the composed models $\mathcal{R}_{Sys||TM}$. In the following it is shown how timed automata extended with (discrete) variables can be employed for this purpose. We implement the presented approach on top of the timed model checker Uppaal.

To this end, this section describes two fundamental contributions for rendering a conformance test using a model checker feasible.

—The modeling of the composition of the system model *Sys* with the trace model *TM*. This must allow: (a) the nondeterminism inherent in *Sys*, (b) a compositional modeling of the system model *Sys*, and (c) independent progress for *TM* and *Sys*.
—We present an optimization on the size of the trace model *TM* that considerably reduces the input size of the model checking problem.

3.1. Timed Automaton Models Employed in TTQ

In the following, the timed automata models for the measurements (TM) and for the system specification are presented (Sys).

3.1.1. Model of Timed Measurements (TM). The formal model of a time series of measurements is a timed automaton TM. It contains a variable p that denotes the measurement of a physical quantity and a single clock c_{TM} that denotes the clock valuations at a change of p. Formally TM is defined as follows. We have

$$TM = \left(Loc_{TM}, Loc^0_{TM}, Act_{TM}, u_{TM}, v_{TM}, \hookrightarrow_{TM}, I_{TM}\right),$$
$$Act_{TM} = \{\tau\}, u_{TM} = \{c_{TM}\}, v_{TM} = \{p\},$$

where we denote $l_{TM} \in Loc_{TM}$ as a location in TM. $l^f_{TM} \in Loc_{TM}$ is the final location of TM.

TM is constructed by generating a location in the trace for each pair in the timed trace of measurements. A clock is used to stay in a given location exactly for the time between the current and the previous measurement, i.e., for $(t_1, a_1); (t_2, a_2)$ and assuming the trace starting at time 0, the time in location l_1 is t_1 and for the subsequent location l_2 it is $t_2 - t_1$. However, the size of a trace of measurements is typically very large as: (a) typical lab instruments provide a high resolution, e.g., in the order of milliseconds (ms) down to nanoseconds (ns) for power consumption, and (b) the measurement noise leads to frequent changes of the measurement value resulting in frequent state transitions. As a result, there is a need for compressing millions of data points into a processable number of locations in TM. This is described at the end of this section (refer to Section 3.3.2), as necessary technical concepts need to be presented first.

3.1.2. System Specification Sys. At first, it is assumed that Sys is a single timed automaton. This means that for simplicity of presentation, it is initially ignored that the model may consist of a set of cooperating timed automata. The locations of Sys are annotated with invariants on clocks and variables. This results in a description of the system with respect to timing and the physical quantity. For Sys, the following notation is used.

$$Sys = (Loc_s, Loc^0_s, Act_s, u_s, v_s, \hookrightarrow_{Sys}, I_s), \{h_{low}, h_{up}, p\} \subseteq v_s$$

In the following, some aspects of Sys are detailed.

(1) Set of variables $\{h_{low}, h_{up}, p\} \subseteq v_s$. The model of the trace TM communicates with the system model Sys via a shared variable p that holds the values of the measured physical quantity. For dealing with deviations in the measurements, Sys employs a pair of variables h_{low}, h_{up}, which take values in $\mathbb{D}$. Pairs of these variables specify upper and lower bounds to be respected by the measurement p, i.e., $p \in [h_{low}, h_{up}]$. As pointed out earlier, the labels of the variables refer here to their valuations with respect to a system state $(\vec{s}, u, v)$. It is interesting to note that Sys only reads p, whereas TM updates p to signify changes in the measured physical quantity. The variables h_{low}, h_{up} are manipulated when traversing a respective edge of Sys. Sys may contain additional variables, but such additional variables are irrelevant for the discussion to follow.

(2) Set of locations Loc_s. This set consists of the partitions $\mathcal{M}$ and $\mathcal{N}$:
Set of system modes $\mathcal{M}$
A location $m \in \mathcal{M}$ represents a mode of operation that possesses a fixed lower and upper bound h_{low}, h_{up} on power consumption p. Corresponding location invariants allow to invalidate pairs of locations and measured values, namely if $p \notin [h_{low}, h_{up}]$. Hence, it is straightforward to define location invariants that assert that a provided

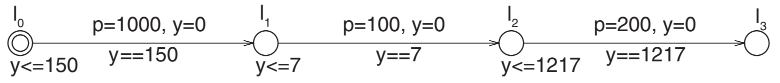

Fig. 2. A short excerpt of an exemplary *TM* in Uppaal. In this implementation, the physical quantity, power consumption, is annotated as p. The clock is specified as a clock variable y. A value of $p = 1000$ is initially measured for the first 150 time units.

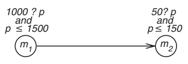

Fig. 3. System mode transitions without update locations are not possible.

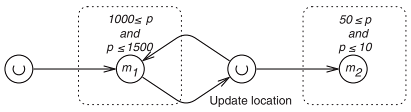

Fig. 4. The update mechanism of *Sys* as implemented in Uppaal. System modes have an intermediate urgent update location without any invariant.

measurement held by variable p is conformant with the modeled system.[1] The upper and lower bounds are updated when *Sys* executes a transition. However, in case variable p requires an update to a value which does not agree with the currently provided power interval $[h_{low}, h_{up}]$ a violation of a location invariant occurs; this violation terminates the currently explored execution path. As an example consider *TM* in Figure 2 in combination with the system model *Sys* in Figure 3: while residing in location $l_0 \in TM$ and $m_1 \in Sys$, *TM* cannot traverse into location l_1 as this violates the invariant $I(m_1)$. Once clock y reaches 150 this situation leads to a deadlock of the composite network of timed automata, ultimately terminating the current branch of execution. Consequently such a naive modeling prohibits updates of p. For resolving this problem we introduce update locations within the system model *Sys* and avoid the potential deadlocks illustrated before.

Set of update locations $\mathcal{N}$

Update locations $n \in \mathcal{N}$ are urgent locations without any invariants. They are artifacts of the presented approach as they allow the update of the shared variable p in another automaton, i.e., by executing a transition of *TM*. Updates on p in system modes may not be possible, as they may violate the location invariant $p \in [h_{low},\ h_{up}]$. Hence update location and system mode locations are interleaved. As an example consider *TM* in Figure 2 but now in combination with the system model *Sys* in Figure 4. At time $y = 150$, *Sys* can traverse into the update location embedded between m_1 and m_2. This can immediately be followed by letting *TM* traverse into location l_1. Without delay *Sys* may either go back to the previously held location m_1 or a newly reached location m_2, depending on the validity of the associated location invariants. In the example only a traversal into location m_2 is possible. However, in case both locations are valid target locations, a choice is taken nondeterministically, where due to the exhaustiveness of (timed) verification both model futures have to be explored.

[1]As a consequence, the presented approach requires measurements that do not refer to a physical quantity that is an integral over time. The physical quantity must be an instantaneous measurement that can be determined by only looking at the current state and does not require any history. It is exactly this assumption that allows the conformance test to stay within the class of timed automata, rather than being forced to make use of linear priced timed automata [Behrmann et al. 2005] or other implementations of hybrid automata [Henzinger et al. 1997].

With the pattern depicted in Figure 4 we implement the following semantics. We have

$$\langle m, u_s, \{h_{low}, h_{up}, p\}\rangle \xrightarrow{\tau}_{Sys} \langle n, u_s, \{h_{low}, h_{up}, p'\}\rangle \xrightarrow{\alpha_s}_{Sys} \langle m', u'_s, \{h'_{low}, h'_{up}, p'\}\rangle;$$
$$\text{with } m, m' \in \mathcal{M}; n \in \mathcal{N}; I(n) = (\emptyset, \emptyset); (u_s, \{p, h_{low}, h_{up}\}) \in I(m);$$
$$(u'_s, \{p', h'_{low}, h'_{up}\}) \in I(m'); u'_s = [r_s \to 0]u_s$$

where in the following we write for simplicity

$$\langle m, u_s, \{h_{low}, h_{up}, p\}\rangle \xrightarrow{\alpha_s}_{Sys} \langle m', u'_s, \{h'_{low}, h'_{up}, p'\}\rangle.$$

Note that x' denotes a change of x after taking a transition, i.e., either a location change or a change in the valuation. $u'_s = [r_s \to 0]u_s$ denotes that a subset of clocks $r_s \subseteq u_s$ in *Sys* may be reset on a transition to a new system mode. Delay transitions in update locations are not possible, since update locations are urgent locations. Delay transitions in system modes follow from the definition of timed automata.

3.2. Reachability Check for Verifying TTQ Conformance

The trace model *TM* and the system model *Sys* are executed concurrently referred to by *TM*||*Sys*. *TM* and *Sys* only cooperate via a shared variable p, i.e., no rendezvous takes place between them, such that $Act_{TM} \cap Act_s = \emptyset$, $v_{TM} \cap v_s = \{p\}$ holds.

The global variable p is updated when transitions of *TM* take place. Updates of p may potentially invalidate location invariants of *Sys*, which is the reason for using update locations in *Sys*. This specific modeling of *Sys* results in the transition relations defined next.

3.2.1. Definition of Transition Rules. With the presence of mode and update locations in *Sys*, the following transition rules are obtained. Note that in order to distinguish transitions in *TM* and *Sys* from the transitions in a composed model, a subscript is used on the transition relation: $\hookrightarrow_{TM}$ for *TM* and $\hookrightarrow_{Sys}$ for *Sys*. For all transition rules, it holds that $m \in \mathcal{M}$ and $l_{TM} \in Loc_{TM}$.

(a) Delay transition. *Sys* and *TM* stay in their current location and increase their clocks with the same duration.

$$\frac{\begin{array}{c}\langle l_{TM}, u_{TM}, \{p\}\rangle \xrightarrow{d}_{TM} \langle l_{TM}, u_{TM} + d, \{p\}\rangle \quad \wedge \\ \langle m, u_s, \{h_{low}, h_{up}, p\}\rangle \xrightarrow{d}_{Sys} \langle m, u_s + d, \{h_{low}, h_{up}, p\}\rangle\end{array}}{\langle (l_{TM}, m), (u_{TM}, u_s), \{h_{low}, h_{up}, p\}\rangle \xrightarrow{d} \langle (l_{TM}, m), (u_{TM} + d, u_s + d), \{h_{low}, h_{up}, p\}\rangle};$$

with $d \in \mathbb{R}_{\geq 0}$ and

$$\forall d' \in \mathbb{R}_{\geq 0}, 0 \leq d' \leq d : (u_{TM} + d', v_{TM}) \in I(l_{TM}) \wedge (u_s + d', v_s) \in I(m)$$

(b) Current power consumption is accepted by different system locations. *Sys* may traverse to another system mode that also accepts the current power consumption.

$$\frac{\langle m, u_s, \{h_{low}, h_{up}, p\}\rangle \xrightarrow{\alpha_s}_{Sys} \langle m', u'_s, \{h'_{low}, h'_{up}, p\}\rangle}{\langle (l_{TM}, m), (u_{TM}, u_s), \{h_{low}, h_{up}, p\}\rangle \xrightarrow{\alpha_s} \langle (l_{TM}, m'), (u_{TM}, u'_s), \{h'_{low}, h'_{up}, p\}\rangle};$$
$$\text{with } (h_{low} \leq p) \wedge (p \leq h_{up}) \wedge (h'_{low} \leq p) \wedge (p \leq h'_{up})$$

(c) Current system location accepts value of power consumption before and after update. *TM* updates the power consumption, yet *Sys* stays in the current system mode, since this system mode accepts the power measurement before and after the update.

$$\frac{\langle l_{TM}, u_{TM}, \{p\}\rangle \xrightarrow{\tau}_{TM} \langle l'_{TM}, u'_{TM}, \{p'\}\rangle}{\langle (l_{TM}, m), (u_{TM}, u_s), \{h_{low}, h_{up}, p\}\rangle \xrightarrow{\tau} \langle (l'_{TM}, m), (u'_{TM}, u_s), \{h_{low}, h_{up}, p'\}\rangle};$$
$$\text{with } u'_{TM} = [r_{TM} \to 0]u_{TM}, (h_{low} \leq p) \wedge (p \leq h_{up}) \wedge (h_{low} \leq p') \wedge (p' \leq h_{up})$$

(d) Mode change: *Sys* needs to enter a new system mode to accept a change in power consumption in *TM*. This will be denoted as a mode change in the following.

$$\frac{\begin{array}{c}\langle l_{TM}, u_{TM}, \{p\}\rangle \xrightarrow{\tau}_{TM} \langle l'_{TM}, u'_{TM}, \{p'\}\rangle \ \wedge \\ \langle m, u_s, \{h_{low}, h_{up}, p\}\rangle \xrightarrow{\alpha_s}_{Sys} \langle m', u'_s, \{h'_{low}, h'_{up}, p'\}\rangle\end{array}}{\langle (l_{TM}, m), (u_{TM}, u_s), \{h_{low}, h_{up}, p\}\rangle \xrightarrow{\alpha_s} \langle (l'_{TM}, m'), (u'_{TM}, u'_s), \{h'_{low}, h'_{up}, p'\}\rangle};$$

with $u'_{TM} = [r_{TM} \to 0]u_{TM}, (h_{low} \leq p) \wedge (p \leq h_{up}) \wedge (h'_{low} \leq p') \wedge (p' \leq h'_{up})$

Note that an actual mode change only occurs if it holds that $\neg(h'_{low} \leq p) \wedge \neg(p \leq h'_{up}) \wedge \neg(h_{low} \leq p') \wedge \neg(p' \leq h_{up})$, i.e., the current system mode does not accept the future power consumption and the new system mode does not accept the current power consumption. Our modeling enables a traversal back to the system mode if a mode change is not required. This is significant when *Sys* is composed of a network of timed automata as explained shortly.

3.2.2. Modeling the Composition. The composition of *TM* and *Sys* necessitates careful modeling. As an example, transition rules (a) and (b) necessitate that *TM* and *Sys* may traverse independently at any point in time. There are also special considerations for implementing mode changes for a model *Sys* composed of a network of timed automata. Each individual component timed automata model in *Sys* that reads the variable p may feature mode changes. On a change of p in *TM* only a subset of these component timed automata may require a mode change of some components. Other component timed automata may stay in their given system mode. However, all component timed automata must intermittently transfer into an update location without an invariant on p to enable a mode change. For this reason there is a back edge from update locations to the corresponding system mode as shown in Figure 4. Note that we also experimented with an alternative composition of *TM* and *Sys* using synchronization via a dedicated broadcast channel. However, the model using synchronization performed worse with respect to state space size and conformance test runtime.

The transition rules are implemented using the update process shown in Figure 4. The automaton, which could be an excerpt of a system model *Sys*, can unconditionally traverse into an update location. The update location allows *TM* to execute a transition that assigns a new value to p. When leaving the update location, *Sys* updates upper and lower bounds, i.e., assigning new values to h_{low} and h_{up}. The location invariant of potential target mode locations depends on these new values. If the location invariant does not hold, the corresponding transition is invalidated. This implies that the overall model *TM*||*Sys* deadlocks in an update location if the location invariant does not hold for any of the potential target mode locations.

The coupling of *TM* and *Sys* allows to check if the finite timed trace as produced by *TM* is included within the set of traces that can be produced by *Sys*. A standard timed model checker like Uppaal can be used to formulate a reachability query for the final location l^f_{TM} of *TM*.

3.2.3. Counterexample. If *TM* models a behavior not explained by *Sys*, the concurrently executed model *TM*||*Sys* does not reach a state where location l^f_{TM} is marked as active.

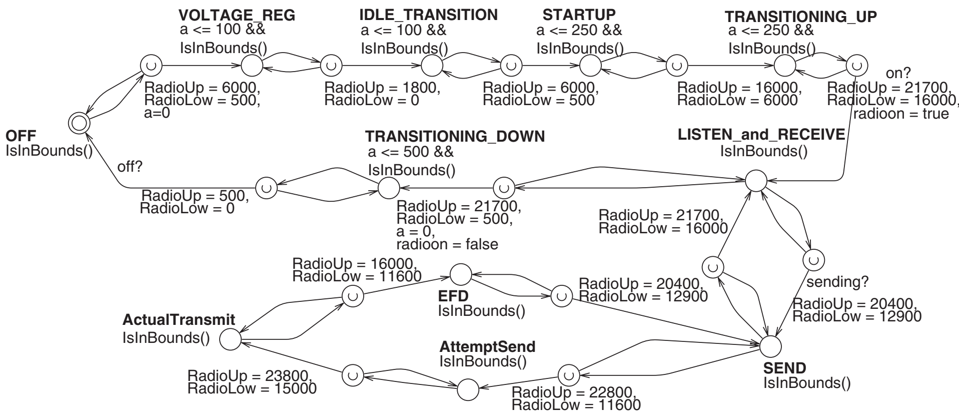

Fig. 5. Uppaal model of the radio hardware component. Lower and upper bounds on power consumption are annotated as `RadioLow` and `RadioUp` (in μA). A clock `a` controls transition times between system modes. `IsInBounds` is the location invariant on power consumption of the comprehensive system model as described in Section 3.3.1.

The last transition that lets a property ultimately fail might be related to the underlying cause. In the employed reachability property for TTQ, this "deadly transition" is given by the last state reached on the longest path of $\mathcal{T}_{TM||Sys}$ (longest with respect to time). A possible method to receive this information is to first label the locations of *TM*, e.g., by indexing them. Subsequently, one can repeatedly check for the reachability of a location based on the label using some search strategy such as binary search. Iterative calls to Uppaal on reachability of annotated index labels are employed to determine the location in *TM* where the conformance test fails.

3.3. Compositional Modeling of the System Model

For simplicity, the explanations given before ignored the fact that a system model *Sys* might be built in a compositional manner, where different timed automata represent individual components of the system. In the following, these basic building blocks of the overall system model will be addressed as component timed automata. As an example, a hardware component timed automaton may describe a certain piece of hardware such as a microcontroller with different modes, e.g., performing a computation task or residing in a low-power mode. A software component timed automaton may represent some piece of software controlling some hardware components, where in particular these software components are time-driven. Shared variables or rendezvous mechanisms can be used for coordinating interactions among the component timed automaton as explained in Section 2. Figure 5 shows that the radio model for a sensor node is periodically turned on for listening on the channel via the label `on`. The corresponding software model is shown in Figure 6: it specifies that the radio must periodically exit its power-off state (with an invariant on the `IDLE` location).

As a major difficulty, such a compositional approach has to cope with the fact that individual hardware components may contribute differently to the power consumption, i.e., each system mode of a hardware component timed automaton may contribute differently. In particular, each hardware component timed automaton has two variables for describing the allowed power consumption in a specific system mode. The overall allowed power consumption is defined by the set of system modes the hardware components of *Sys* are residing in.

As an example, Figure 5 shows the component timed automaton modeling the radio: It features 11 system modes and the corresponding power consumption bounds (here

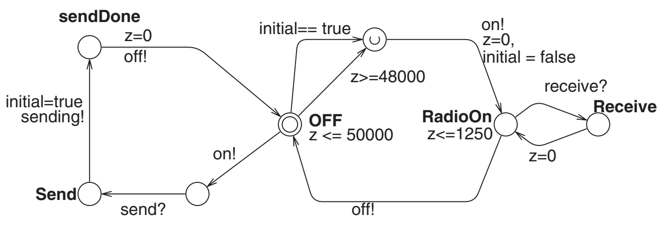

Fig. 6. Uppaal model of the radio software. The initial state of the software is the IDLE location. This location must be periodically exited, at least every 50,000 times units as seen by the invariant on the clock `z`: `z <= 50000`. The software model uses binary synchronization (channels `on`, `off`, and `sending`) to synchronize with the radio component. Note that `receive?` and `send?` synchronize with a testcase model that specifies when packets may be received and sent.

described with the variables `RadioLow` for the lower bound and `RadioUp` for upper bound). Each system mode is annotated with its individual power consumption bounds on the incoming edge. In a powered-off state (`OFF`), power consumption, or equivalently in this case current draw, is lower bounded by $0mA$ and upper bounded by $0.5mA$.

3.3.1. Interval Composition. Each component timed automaton i has its own set of variables $\{h^i_{low}, h^i_{up}\}$ that indicate the currently accepted interval. As a system state $(\vec{s}, u, v)$ contains the active location of each component, the interval bounds of the overall system model Sys can be obtained by adding the lower and upper bounds associated with the location of each n component timed automata: $h^1_{low}, \ldots, h^n_{low}$ and $h^1_{up}, \ldots, h^n_{up}$. The sums are assigned to the global variables h_{low} and h_{up} as introduced earlier: $h_{low} := \sum_{i=1}^{n} h^i_{low}$ and $h_{up} := \sum_{i=1}^{n} h^i_{up}$. Since the bounds h_{low} and h_{up} are additively composed, the power intervals described by the bounds may not be disjoint. All possible intervals of power consumption can be computed offline given the system model Sys.

As an example, the case study in Section 5 uses two components: a microcontroller as well as a radio. The implementation in Uppaal is displayed shortly for the bounds of the radio, `RadioLow` and `RadioUp`, and the bounds of the microcontroller, `MCLow` and `MCUp` and the consumption `crrValue`.

```
bool IsInBounds(){
        int h_up = RadioUp+MCUp;
        int h_low = RadioLow+MCLow;
        if(crrValue > h_up or crrValue < h_low) return(false);
        return(true); }
```

3.3.2. Reducing Power Trace Size. As discussed in Section 3.1.1, the size of TM is a major obstacle. A timed trace obtained from some measurement can include millions of measurements. Given that the input size of models for standard model checkers is constrained, a considerable reduction must be achieved. In order to reduce the number of locations in the sequential TM, we can exploit the computed set of intervals of Sys: We segment the value range of power measurements into intervals. The use of intervals, i.e., valuations of h_{low} and h_{up}, on power consumption in Sys allows TTQ to abstract from the measurements. For keeping the number of power intervals as small as possible, TTQ exploits the concept of Greatest Common Intervals (GCIs) as described in the following.

3.3.3. Greatest Common Intervals. In order to determine a minimal representation of intervals, TTQ exploits the concept of a GCI partitioning [Strehl 2000], whereby a set of nondisjoint intervals, such as the set of bounds H_{Sys}, is transformed into a minimal size set of disjoint intervals H_{GCI}. Since GCIs are disjoint, a single value can be used for the representation of a single GCI: the average value of the interval. Each of these

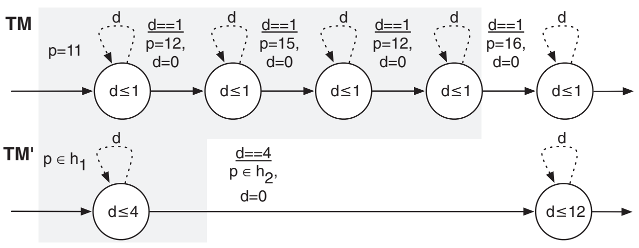

Fig. 7. Construction of *TM*′ from *TM* by looking at the GCI-valuation-equivalence. Here $h_1 = [10, 15]$ and $h_2 = [16, 18]$.

disjoint intervals represents an equivalence class with respect to the measurement of the physical quantity. Different measurements inside a GCI cannot be distinguished. This is used in TTQ for reducing the number of locations of *TM*.

3.4. Trace Automaton Optimization

The trace reduction described before must preserve the timed behavior with respect to the physical quantity. The set of GCIs can be computed (offline) from the system specification model *Sys*. As a result, the construction of a compressed trace of measurements is in principle straightforward. When the compressed trace is constructed, a respective timed automaton can be derived, which is denoted as quotient timed automaton *TM*′ in the following. Shortly, it will be shown that *TM* and *TM*′ are equivalent with respect to the timed sequence of GCI-visits, st.

$$Sys \models TM \Leftrightarrow Sys \models TM'$$

For automating the generation of the quotient timed automaton *TM*', TTQ employs an algorithm that creates a list of locations l' in TM' given the measurement samples and the endpoints of the GCIs as presented in Woehrle et al. [2009]. In a nutshell, a location is created for *k* consecutive measurements when the value of the measurement is within the same GCI.

The process of generating the compressed trace and its timed automaton-based representation *TM*′, is exemplified in Figure 7.

Definition 4 (*Residence Time*). Each location $l \in TM$ has a distinct residence time $r \in \mathbb{R}_{\geq 0}$ that is given by the time between two differing measurements. This is modeled by an invariant on a location $d \leq r$ with $\langle l, u, v\rangle \xrightarrow{r} \langle l, u+r, v\rangle$ *where* $\forall d \in \mathbb{R}_{\geq 0} : 0 \leq d \leq r \Rightarrow (u + d, v) \in I(l)$. The location invariant and a corresponding guard on the outgoing transition ensure that the transition to the next location has to be taken exactly after r time units have passed.

The example in Figure 7 shows a typical trace of measurements *TM* with equidistant measurements featuring a residence time of one for each location $l \in TM$.

THEOREM 1. *The quotient timed automaton generation algorithm (as described in Woehrle et al. [2009]) produces a reduced automaton TM′ that is equivalent with respect to the timed sequence of GCI-visits to the automaton TM derived from the original timed trace of measurements.*

PROOF. *Assumption*: A location $l' \in TM'$ is equivalent to a location $l \in TM$ with respect to its inclusion in a GCI iff for a location $l \in TM$ the valuation of p is in a specific GCI h_l, i.e., $p \in h_l, h_l \in H_{GCI}$ and the valuation of p' in the location $l' \in TM'$ lies inside the same interval: $p' \in h_l$. This is denoted as GCI-valuation-equivalent in the following. The automaton TM' is GCI-valuation-equivalent to TM iff there is a GCI-valuation-equivalent sequential location in TM' for each location in TM.

Basis ($i := 0$): Let us assume that we are at a location l_0 of TM that features a measurement of p_0. The power consumption p_0 is in a given GCI $p_0 \in h_l, h_l \in H_{GCI}$, i.e., the system model *Sys* is indifferent to the specific value within h_l. Hence, l_0 is equivalent with respect to the GCI of the quotient TM' that resides in its initial location l'_0 with $p'_0 \in h_l$.

Inductive step ($i \to i+1$): Let us assume we are at a location l_i with measurement p_i with $p_i \in h_l, h_l \in H_{GCI}$. There are two possible options for successor states l_{i+1}.

(1) $p_{i+1} \in h_m, h_m \in H_{GCI}, m \neq l$. A new location l'_{i+1} is generated in TM' with $p'_{i+1} \in h_m$ and residence time $r'_{i+1} = r_{i+1} = 1$. Hence, l'_{i+1} is GCI-valuation-equivalent to l_{i+1} at least for time r'_{i+1}.
(2) $p_{i+1} \in h_l$. No additional location is added for TM', but the residence time is increased for l'_i st. $r'_{i+1} = r'_i + r_{i+1} = r'_i + 1$. l'_i features the same valuation with respect to the GCI for both l_i and l_{i+1}. Hence, at each instant in time TM and TM' provide the same valuation with respect to the currently visit GCI.

The induction scheme given before proves that we can generate a quotient timed automaton TM' for any time series of measurements. The constructed quotient timed automaton is GCI-valuation-equivalent with respect to the trace model TM for the following reasons: Let TM' be in any location l'_i corresponding to the location sequence $l_p, \ldots, l_q$ in TM. By construction of TM', it holds that the residence time in l'_i is $r'_i = \sum_{j=p}^{q} r_j$. Note that l'_i is GCI-valuation-equivalent to $l_p, \ldots, l_q$. Hence it follows that TM and TM' are equivalent for the time t: $\delta_i \leq t \leq \delta_i + r'_i$ with respect to their GCI-valuation, where $\delta_0 = 0$ and $\delta_j = \sum_{j=0}^{i-1} r'_j$. As TM and TM' may only reside in a single location at any period in time, the induction over i covers all points in time of the original time series of measurements and yields again that TM and TM' are GCI-valuation-equivalent. □

LEMMA 1. *If TM' is an GCI-valuation-equivalent automaton to TM, then $Sys \models TM \Leftrightarrow Sys \models TM'$ holds.*

PROOF. It only depends on the timed sequence of GCIs whether the final location l^f_{TM} of trace automaton TM or TM' is reachable within the composed model $Sys||TM$. As this is the same for TM and TM', the lemma holds. □

4. ANALYZING A CPS WITH TRON

TRON is a model-based tool that tests the conformance of an implementation and its timed automata-based specification, where conformance is tested with respect to timed input/output behavior. Analogously to TTQ, TRON employs timed automata for specifying the desired behavior of an implementation, i.e., only behavior that is allowed by the specification may be seen in any run. TRON uses a relativized timed input/output conformance relation `rtioco` introduced by Larsen et al. [2004] as defined shortly. Note that in this section the notations of the original publication are used. In particular, Larsen et al. [2004] use a different notation for timed traces; in their work, and hence in the following explanations, a timed trace is a sequence of labels. The labels may include actions A and (time) delays $d \in \mathbb{R}_{\geq 0}$. Hence a timed trace $\sigma \in (A \cup \mathbb{R}_{\geq 0})^*$ is of the form $\sigma = d_1a_1d_2a_2 \ldots d_ka_k$ with $d_i \in R_{\geq 0}$ and $a_i \in A$, i.e., labels and delays are concatenated.

Definition 5 (*rtioco*).

Let *imp* and *s* be $\mathcal{TIOTS}$*(timed I/O transitions systems)* :

imp `rtioco` $s \iff \forall \sigma \in Ttraces(e) : out(\langle imp, e\rangle \; after \; \sigma) \subseteq out(\langle s, e\rangle \; after \; \sigma)$

where

—$\mathcal{TIOTS}$ is a timed I/O transitions system. For details on $\mathcal{TIOTS}$, the reader is referred to the original publications of TRON [Larsen et al. 2004], since it is not necessary for the understanding of the application of TRON. It only should be noted that $\mathcal{TIOTS}$ have labels L that include distinct inputs L_i, outputs L_o as well as timed transitions ($d \in \mathbb{R}_{\geq 0}$) and an unobservable internal action label τ, i.e., $L = L_i \cup L_o \cup \{\tau\} \cup \mathbb{R}_{\geq 0}$.
—$\langle imp, e\rangle$ is the composition of implementation *imp* with the environment e. $\langle s, e\rangle$ denotes the composition of specification s with e.
—σ is an observable timed trace, i.e., $\sigma \in (L_i \cup L_o \cup \mathbb{R}_{\geq 0})^*$, i.e., a trace containing inputs, outputs, and (time) delays.
—*after* σ denotes the set of possible states a system may be in after executing a timed trace σ.
—$Ttraces(e)$ is the set of all possible timed traces of the environment e.
—$out(K)$ is the set of outputs ($L_o \cup \mathbb{R}_{\geq 0}$) a $\mathcal{TIOTS}$ may produce from a set of states K.

Intuitively, `rtioco` denotes that when providing the same environment to specification and implementation and executing any timed trace from the environment, the outputs that one may see from the implementation model must be a subset of the specification model. Hence, the implementation has only behavior that is allowed by the specification. Delays and outputs are only allowed if they are specified in s. TRON verifies that the outputs of the implementation $out(\langle i, e\rangle)$ *after* σ are included in the outputs allowed by the specification $out(\langle s, e\rangle)$ *after* σ. The discussion with respect to TTQ only focuses on outputs of a system.

For determining the outputs of the implementation, TRON needs a connection to the specific implementation. For offline testing as considered in this work, a test adapter is provided that reads a textual trace. For accessing the timed automaton-based system description, TRON needs a sampler process to be provided by the test engineer. This sampler process uses dedicated signals and variables, allowing a comparison of input/output values of implementation and specification. In this work, the sampler process outputs the physical quantity as produced by the system at a given time, allowing its comparison to the respective value in the trace of measurements at this instant. In order to understand the testing procedure, trace adapter and sampler process are briefly discussed in the following.

4.1. Trace Adapter

TRON needs a connector to the implementation to read its outputs. For TTQ, traces are already available from a given execution. For such offline testing, TRON provides a *trace adapter* that accepts a textual representation of the timed trace. Input actions, output actions, and delays are defined. As previously mentioned, each action may have variables attached that are compared to the specification when the action is triggered, but not in any other case.

Listing 1 shows an exemplary trace used in the case study, where the physical quantity represents current draw under constant supply (i.e., power consumption). Lines 1 and 2 show the declaration of input and outputs actions and the associated (integer) variables: there are no input actions and only a single output action (`report`) with an associated variable for defining the power consumption (`crrValue`). Lines 3

```
 1  input ;
 2  output report(crrValue);
 3  precision 20;
 4  timeout 20000000;
 5
 6  output report1(0);
 7  delay @6.0;
 8  output report1(400);
 9  delay @8.0;
10  output report1(100);
11  ...
```

Listing 1. TRON trace format with a single output variable `crrValue` describing measured power consumption.

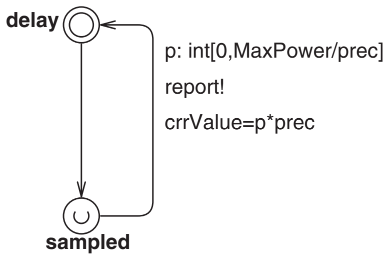

Fig. 8. TRON sampler process for producing the output (`crrValue`) from the Uppaal model using a nondeterministic selection `p` with granularity `prec`.

and 4 are declarations for TRON parameters that include the `precision` of individual time units (here specified as $20\mu s$.), and a `timeout` that is used to signal when to stop the testing process[2]. The actual trace, which starts at line 6, is specified as a list of delays and output actions with a corresponding value. Delays are specified with respect to time units (`delay @6.0;`). Output actions are specified with the value of the associated variable at that moment in time (`output report1(400);`). The trace that follows describes that initially the current draw is $0\mu A$. After $120\mu s$, the current draw changes to $400\mu A$. Finally at $160\mu s$, the implementation draws $100\mu A$. For TRON such an execution trace for a given trace of measurements is generated based on the optimization described in Section 3.4.

4.2. Sampler Process

TRON uses timed automata models of the specification and employs the Uppaal verification engine [Mikucionis et al. 2004] to compute the future state space of the specification. The *sampler process* is a model that specifies input and output actions of the specification. Figure 8 shows the *sampler process* used in the case study with a single output. The sampler process models the time when an output (`report`) may be generated. Additionally, it defines the range of possible values for the output variable `crrValue`, i.e., it specifies all values that may be output. These are the same output action and variable as specified in Listing 1, i.e., trace and sampler process synchronize via `report` and `crrValue`.

On the transition, the current valuation of the power consumption variable `crrValue` is selected nondeterministically from the interval $[0, \texttt{MaxPower}]$, which is achieved by using Uppaal's selections (here `p`). Selections nondeterministically assign an integer

[2]The length of the test is determined in the offline case of TTQ by the length of the power trace previously measured.

value within the given interval. To reduce granularity of possible values for `crrValue`, we first divide by a constant `prec` before the selection and then multiply by `prec` after the selection. Hence, `prec` defines the granularity of allowed power consumption values. An evaluation of `prec` is detailed in the case study (refer to Section 6.4.2). `MaxPower` is a constant, defining the maximum value for the power consumption of the complete system. Note that Uppaal's selections require constants in their ranges, i.e., the bounds of the interval need to be fixed at compile time. The sampler process shown in Figure 8 does not describe any temporal behavior for the output; since power may be measured (and change) at any point in time, a change in power consumption is allowed at any time by the model. The sampler process is composed with the system specification model *Sys*. *Sys* is identical to the TTQ-based approach.

4.3. TTQ Execution with TRON

For a better understanding, we may assume that a timed trace σ, as provided through the trace adapter, is executed. Let us further assume that from the last observation at time t_0 neither an input $i \in L_i$ nor an output $o \in L_o$ is provided by the implementation for δ time units ($\delta \in \mathbb{R}_{\geq 0}$). After this δ time units delay, the implementation produces a specific output $o \in L_o$. Remember, that the outputs of a $\mathcal{TIOTS}$ in states K are defined as $out(K)$ and that the set of outputs includes $L_o \cup \mathbb{R}_{\geq 0}$. Formally, this means that $\delta \in out(\langle i, e\rangle \textit{ after } \sigma)$ and $o \in out(\langle i, e\rangle \textit{ after } \sigma\delta)$, where $\sigma\delta$ denotes the concatenation of time delay δ to the timed trace σ. For resolving this situation, TRON performs a state space exploration of the system model $\langle s, e\rangle$ and computes the largest delay d_{max} that is possible starting at t_0: $d_{max} = \max(d \in \mathbb{R}_{\geq 0} | d \in out(\langle s, e\rangle \textit{ after } \sigma))$. Given d_{max}, it can determined whether the delay δ is acceptable. If $d_{max} < \delta$ the delay is not acceptable and the test fails. In case $d_{max} \geq \delta$, the delay is allowed by the specification and TRON needs to check the respective target states for the possible output o. Once again this is done by state space exploration and the check whether $o \in out(\langle s, e\rangle \textit{ after } \sigma\delta)$ holds. If this is true the delay and output are appended to the timed trace: $\sigma' = \sigma\delta o$ and the exploration proceeds with all valid target states. This means that TRON needs to keep all potential target states in memory, such that it is capable of exploring all potentially valid traces. This yields that the reachable (valid) states are visited in a breadth-first search and stepwise manner. Contrary to this, the reachability check as performed by TTQ can be organized in an arbitrary order, e.g., depth-first search.

As pointed out earlier, TRON features the concept of a future size [Larsen 2009] that is used to limit the precomputation to δ_{max} time units. TRON features the command line option `-F` for specifying the future size δ_{max} (in time units). For TTQ, this is important, since it determines the computational overhead of determining (future) outputs.[3] Consider the previous example of determining $\sigma' = \sigma\delta o$. Let us assume a delay transition of $\delta = 3000$. If $\delta_{max} = 1000$, TRON would need 3 separate precomputation steps and arrive at a trace: $\sigma' = \sigma\delta_{max}\delta_{max}\delta_{max}o$. Choosing $\delta_{max} = 5000$ reduces the computational overhead by pre-computing the state space, since it uses just one exploration step: $\sigma' = \sigma\delta o$. The effect of the future size on checking power traces is explored in Section 6.4.1.

5. APPLICATION OF TTQ

In the case study we use an actual implementation of a CPS developed in our group: a wireless sensor node interacting with the environment. We select power consumption as the physical quantity to be observed. Testing for power consumption is challenging, as it requires fine-grained measurements resulting in traces with hundreds of thousands of measurement samples. We measure the power consumption of a

[3]The future size also plays a role in the testcase generation.

typical wireless sensor node, running a standard sensor network application, called Harvester [Lim et al. 2009]. Harvester collects data from individual sensor nodes to one or more data sinks. Harvester is an open-source project running on the TinyOS 2 operating system. It targets energy-aware collection of environmental data in order to be able to monitor environmental phenomena for multiple years. Hence, power consumption for such a system is of utmost importance and the software implementation needs to guarantee an energy-saving operation, i.e., it requires extensive usage of the hardware's low-power modes. The power-aware programming of such systems is complex and error-prone: Each hardware component features its own characteristic set of power modes, activities can be executed concurrently and contribute differently to the level of power consumption. Asserting the correctness of such systems includes the assertion of functional and nonfunctional properties such as power consumption as well as real-time properties, as complex communication tasks must be guaranteed.

Harvester is based on the standard TinyOS multihop routing protocol and an adapted, synchronized Low-Power Listening (LPL) MAC protocol. In this case study we focus on an implementation on the Tmote Sky sensor node, featuring a TI MSP430 low-power microcontroller and a packet-based TI CC2420 radio.

A simplified version of Harvester is considered, which does not use the analog-to-digital converter to read sensor values but simply sends a given value periodically. Thus the components to be modeled can be reduced to the two main contributors, the microcontroller and the radio. Other components such as LEDs, sensors, or the external flash are persistently powered off and are not included within the analysis. In our TTQ the power consumption of a sensor node is monitored by the voltage drop across a MHP201R0F 1Ω (±1% tolerance) shunt resistor measured by a digital multimeter (Agilent 34411A). We measure current consumption under a constant supply voltage and use current draw and power consumption interchangeably in the following. A node is monitored for 20s, due to the limited storage depth of samples by the multimeter at 10^6 data points. Harvester is configured with a wake-up period of 1000 binary ms or approximately 0.977s.

The Harvester is a complex system running an intricate SW stack on top of a CPS. We manage the complexity of the system model through an abstract representation of expected behavior and component decomposition. In particular, we differentiate between *HW-oriented models* capturing the low-level behavior of individual HW components and the corresponding low-level SW, *SW-oriented models*, representing the application-level SW as well as the environment and testcase. For details on the models, we refer to the original report [Woehrle et al. 2009].

6. EMPIRICAL EVALUATION: RESULTS AND BENCHMARKS

In this section, we present experimental results on the case study. Firstly, we describe the impact of our optimizations on input problem sizes. Secondly, we detail on the experimental results for the implementations using TRON and Uppaal. The results for Uppaal are updated from Woehrle et al. [2009] with respect to two improvements. We refined the system model and used a newer version of Uppaal to remove the input size limitation to 65,536 locations.

6.1. Power Trace Models

Table I presents a summary of the five testcases from Woehrle et al. [2009] with respect to the original power trace sizes in number of individual measurements, and the effect of the quotient transition system optimization concerning the resulting number of locations in Uppaal. As can be seen, for a typical example the optimization results in a compression of locations by at least an order of magnitude.

Table I. Measurement Samples for Each Individual Testcase and Corresponding Location Count for the Quotient Transition System Using the Optimization Described in Section 3.3.3

Trace	Model				
	Wake-up	Inject	Complex	MC state	Specification
Samples	1,000,000	990,000	310,000	1,000,000	1,000,000
Locations	1,141	1,087	23,418	1,293	1,336

6.2. Experimental Setup

All conformance tests were performed on a Sun-Fire-X2200-M2-64 blade running Linux. It features 2 dual-core 64-bit AMD Opteron processors, i.e., 4 cores, running at 2.6 GHz. The blade is equipped with 8GB RAM. The command-line verifier of Uppaal 4.1.2 is used (`verifyta`). In particular, `verifyta` is run with the `-u` option to obtain information about explored and stored states. For TRON version 1.4b5 is used, since tests with version 1.5 showed considerable degradations in performance. TRON runs with a verbosity level of 8, in order to backup the state set and allow for final diagnostics in case of a failed conformance test. A logical (simulated or virtual) time clock (`-Q log`) is set. Unless otherwise noted, TRON uses a future size of 50,000 (`-F 50000`, refer to Section 4.3).

The main performance metric of these experiments is the execution time of the conformance test. The generation of the models from raw measurements is out-of-scope of this evaluation, but in the same order of magnitude for both tools. The models for Uppaal and TRON are the same for the system specification *Sys*; the obvious difference is that Uppaal includes the trace model *TM*, while TRON features the Sampler process and an input trace as described in Section 4. For measuring time, the unix `time` facility is used and user times reported. Unless otherwise noted, the quotient transition system optimizations are performed as described in Section 3.4.

6.3. Uppaal Results

In the following different design decisions are evaluated based on: (i) the representation of time and (ii) the representation of data values.

6.3.1. Measurement Timing. The first experiments investigate whether the representation of time in the power trace makes a difference for Uppaal. For each power trace location, an invariant on the time a specific value was measured is used. In Woehrle et al. [2009], the trace model *TM* in Uppaal uses relative durations for each measurement location. However, also an absolute time scale without resetting clocks after each power trace location can be employed. The runtime of the relative approach is shorter. However, the size of the explored state space is comparable, indicating that Uppaal can internally better process short intervals rather than intervals with a large offset. Our experiments on all testcases show that using relative times performs generally better.

6.3.2. Measurement Granularity. We also performed experiments concerning different measurement granularities. Initially a resolution for 1 μA of measurements was used. As a second step, a restricted granularity of 100 μA was employed, since the quotient transition system optimization allows for a granular representation of power consumption values. This experiment tries to explore whether the size of the value domain has an impact on the size of the state space and runtime in Uppaal. The results for the size of the state space are the same and runtime results are comparable.

6.4. TRON Results

In a similar vein, the performance in TRON for measurement granularity and representation of time was explored.

Table II. TRON: Comparison of Different Measurement Granularities Using a Future of 50,000

Model	Granularity		
	$1\ \mu A$	$100\ \mu A$	GCI-based
Wake-up	$456s$	$254s$	$210s$
Inject	$214s$	$120s$	$92s$
Complex	$26,482s$	$9,620s$	$13,263s$
MC state	$240s$	$131s$	$98s$
Specification	$223s$	$122s$	$93s$

6.4.1. Measurement Timing. For representation of time, the effect of different future sizes was explored. The future option in TRON (`-F`) describes the number of time units the state space of the specification is precomputed. If this future window is short, TRON has to go through multiple state space explorations as explained in Section 4.3. If the future size parameter is too large, too much of the future state space is explored; an output may be previously seen from the implementation. Note that the Harvester model is periodic with 50,000 time units due to its wake-up behavior, i.e., there is always an output within an interval less than this period. The results show all future sizes perform equally as long as the state space precomputation is larger than the periodicity of the software, i.e., $\geq$50,000 time unites. Larger future size values do not result in an expensive exploration: As the models have a periodic behavior, the state space exploration is finished after one wake-up period. Hence, increasing the future size value larger than the period does not result in any exploration overhead. However, there is a penalty for shorter values, since the state space exploration is cut into smaller pieces. This generates a considerable overhead in terms of individual iterations of explorations (cf. Section 4.3).

6.4.2. Measurement Granularity. Table II presents the results for TRON concerning measurement granularity ($1\mu A$ and $100\mu A$) for the different testcases. Note that in this case, the value domain makes a significant difference. The difference stems from the coupling through the sampler process: TRON needs to compute the future state space of the specification and then performs a comparison with the implementation trace. The sampler process allows for selecting any integer value within the given bounds. Hence, the finer the granularity, the larger the sets to be compared. The interface between the Uppaal exploration and TRON needs to compare each possible output with the trace output of the implementation. This comparison creates a considerable overhead.

In order to allow for the largest minimization of the number of measurement values, a trace abstraction on GCI intervals was additionally performed: Each measured power consumption value is annotated only with its associated GCI in the trace (refer to Section 3.3.3). In turn, the specification needs to be extended to associate GCI intervals with system states. To this end, the invariant function of the specification model is changed as shown in Listing 1. The number of possible values in the sampler can be reduced from 262 (for 100 μA granularity) down to 42 (GCI) values. This allows for some further improvements as shown in the right column of Listing 2. 'Complex' apparently does not benefit from a reduced representation in the sampler process, probably due to the large degree of nondeterminism. It also seems that computing the value of `actual` for each location invariant creates some overhead.

6.5. Uppaal versus TRON Comparison

Lastly, the runtime of the best TRON version with the best Uppaal version is compared to give an overall impression of their relative performance. The results are shown in Table III. Uppaal outperforms TRON by at least an order of magnitude.

```
bool IsInBounds(){
    power_t actual = (intervals[crrValue]+intervals[crrValue+1])/2;
        if(actual > RadioUp+MCUp or actual < RadioLow+MCLow) return(false);
        return(true); }
```

Listing 2. Power consumption invariant for GCI-based description of trace. `crrValue` denotes the GCI interval associated with the measurement. An associated power consumption `actual` is computed online in the invariant function.

Table III. Comparison of the Best Runtimes for Uppaal and for TRON

Tool	Model				
	Wake-up	Inject	Complex	MC state	Specification
Uppaal	$3s$	$2s$	$755s$	$3s$	$3s$
TRON	$210s$	$92s$	$9,620s$	$98s$	$93s$

While initially better performance using TRON was expected, its performance can be explained by its typical usage as an *online* testing tool: (1) In offline testing, one exactly knows the time of the next event and can in turn do a limited exploration for exactly this duration. Additionally, in offline testing one knows exactly what the next state must be and performs an exploration only for this particular future state. In contrast, TRON has no knowledge of future states and must perform a (more expensive) complete exploration. In a second step it must perform a comparison between the (large) set of explored, possible states and match them with the set of states allowed by the trace. This happens at each synchronization point, i.e., when there is an output in the timed trace of the implementation. (2) TRON naturally finds the last, "deadly" transition (refer to Section 3.2.3). Hence, there is no need to perform a search as in the Uppaal-based version. Uppaal approximately needs $log_2(n)$ runs of the model checker, where n is the number of locations in TM, to determine the failing location. This results in about 11 to 15 runs for the traces used in the case study.

7. CONCLUSION

The approach presented concerns the conformance testing of an implementation based on measurements of a physical quantity. Before concluding the work with conclusions, we present related work concerning the conformance test and our specific approach.

7.1. Related Work

To the best of our knowledge, TTQ is the first approach to exploit a physical quantity of a CPS to test for conformance to a specification. Typically these physical quantities, such as power and energy consumption, are rather explored in simulation than in real tests, e.g., for sensor networks in Shnayder et al. [2004].

TTQ is based on conformance testing. There is related work concerning conformance relations both untimed [Tretmans 1994] and timed [Larsen et al. 2004] and even for hybrid system descriptions [van Osch 2006]. Most closely TTQ is related to the work by Bohnenkamp and Stoelinga on quantitative testing [Bohnenkamp and Stoelinga 2008]. The main differences is that in quantitative testing the uncertainty of the measurement is constant throughout the specification, i.e., there is a behavior at a distance of at most x, $x \in \mathbb{R}_{\geq 0}$. However, hardware component states may have differing uncertainties depending on the system mode. Hence, a mode-based uncertainty as employed in TTQ by using intervals on individual system modes of hardware components is preferable. Moreover, for quantitative testing there is currently no tool support. The presented evaluation includes TRON [Larsen et al. 2004], since it is an available, maintained tool and allows a direct comparison with an Uppaal-based approach. The use of hybrid systems for testing physical quantities is described in detail in the discussion that follows.

Obvious choices for trace verification are *hybrid model checkers* such as HyTech [Henzinger et al. 1997]. The traces of measurements can be better described by a hybrid automaton that allows models to use continuous variables. For the measurements of a physical quantity a representation as a continuous variable is suitable. However, the system specification abstracts away from the continuous properties of measurements by using bounds. Additionally, the transitional characteristics of physical quantities are often not of interest. Rather the steady-state physical quantity in a given system mode needs to be checked. The formulation as a hybrid automaton does not provide any benefit in modeling when using bounds on system modes. In contrast, model checking of hybrid automata is a more difficult problem than model checking of timed automata. In conclusion, TTQ does not benefit from employing hybrid model checkers.

Another option is the discretization of time. Fundamentally, since we deal with a microprocessor that is synchronous with its clock cycle (for sensor networks in the order of tens of μs), we can safely abstract away from continuous time to discrete time. In turn, the problem is discretized and model checkers for untimed automata may be used. Compared to the blow-up introduced by discretization of time, the representation of time in Uppaal using symbolic methods seems rather efficient and is not perceived as the major issue of the limited performance as discussed in the following.

7.2. Conclusions

This work presents a novel method to test a CPS utilizing measurements of a physical signal. By introducing various optimizations we showed how to use standard timed verification techniques and tools for carrying out the conformance testing of CPS; we employed (and benchmarked) the timed model checker Uppaal [Bengtsson and Yi 2004; Behrmann et al. 2004], and the timed online testing tool TRON [Larsen et al. 2004]. But there are limitations to the presented approach: (a) For applying the proposed technique it is required to obtain measurements and hence necessitates the availability of a deployed system or prototype implementation. (b) The method requires a formal model of the system, and creating the model is a considerable, one-time overhead, but can be reused for other validation methods, e.g., model checking. (c) The approach belongs to the domain of test methods, as it is based on finite execution traces of deployed systems. Unlike model checking techniques the method is therefore not exhaustive. However, this nonexhaustiveness relaxes the size and complexity limitations as imposed by the notorious state space explosion problem inherent to standard state-based verification techniques.

In this article we use the devised conformance testing approach for testing a wireless sensor node implementation utilizing nonintrusively collected power measurements and can identify diverse software defects. Our case study comprises numerous experiments that investigate the effect of different modeling concepts on the efficiency of the method. The experiments highlight the effectiveness of our approach.

REFERENCES

Alur, R. and Dill, D. L. 1994. A theory of timed automata. *Theore. Comput Scie. 126*, 183–235.

Behrmann, G., David, A., and Larsen, K. G. 2004. A tutorial on uppaal. In *International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM-RT'04)*. 3185. Lecture Notes in Computer Science, vol. 3185, Springer, 200–236.

Behrmann, G., Larsen, K. G., and Rasmussen, J. I. 2005. Priced timed automata: Algorithms and applications. In *Proceedings of the Conference on Formal Methods for Components and Objects (FMCO'04)*. 162–182.

Bengtsson, J. and Yi, W. 2004. Timed automata: Semantics, algorithms and tools. In *Proceedings of the Lecture Notes on Concurrency and Petri Nets*. Lecture Notes in Computer Science, vol. 3098, Springers.

BOHNENKAMP, H. AND STOELINGA, M. 2008. Quantitative testing. In *Proceedings of the 8th ACM International Conference on Embedded Software (EMSOFT'08)*. 227–236.

HENZINGER, T. A., HO, P.-H., AND WONG-TOI, H. 1997. Hytech: A model checker for hybrid systems. In *Proceedings of the 9th International Conference on Computer Aided Verification (CAV '97)*. 460–463.

HESSEL, A., LARSEN, K. G., MIKUCIONIS, M., NIELSEN, B., PETTERSSON, P., AND SKOU, A. 2008. Testing real-time systems using UPPAAL. In *Formal Methods and Testing*, Lecture Notes in Computer Science, vol. 4949. 77–117.

KIM G. LARSEN, MARIUS MIKUCIONIS, B. N. 2009. *Uppaal Tron User Manual*. CISS, BRICS, Aalborg University, Aalborg, Denmark.

LARSEN, K. G., MIKUCIONIS, M., AND NIELSEN, B. 2004. Online testing of real-time systems using uppaal. In *Proceedings of the 4th International Workshop Formal Approaches to Software Testing (FATES'04)*. 79–94.

LIM, R., WOEHRLE, M., MEIER, A., AND BEUTEL, J. 2009. Poster abstract: Harvester - energy savings through synchronized low-power listening. In *Adjunct Proceedings of the 6th European Workshop on Sensor Networks (EWSN'09)*. 29–30.

MIKUCIONIS, M., LARSEN, K. G., AND NIELSEN, B. 2004. T-uppaal: Online model-based testing of real-time systems. In *Proceedings of the 19th IEEE International Conference Automated Software Engineering (ASE'04)*. 396–397.

SHNAYDER, V., HEMPSTEAD, M., RONG CHEN, B., ALLEN, G. W., AND WELSH, M. 2004. Simulating the power consumption of large-scale sensor network applications. In *Proceedings of the 2nd International Conference on 6th Embedded Networked Sensor Systems (SenSys'04)*. 188–200.

STREHL, K. 2000. Symbolic methods applied to formal verification and synthesis in embedded systems design. Ph.D. thesis, ETH Zurich.

TRETMANS, J. 1994. A formal approach to conformance testing. In *Proceedings of IFIP TC6/WG6.1 6th Interantional Workshop on Protocol Test Systems VI*. 257–276.

VAN OSCH, M. 2006. Hybrid input-output conformance and test generation. In *Formal Approaches to Software Testing and Runtime Verification*, 70–84.

WOEHRLE, M., LAMPKA, K., AND THIELE, L. 2009. Exploiting timed automata for conformance testing of power measurements. In *Proceedings of the 7th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'09)*. 275–290.

Received February 2010; revised August 2010; accepted December 2010